Provable and Practical Security for Database Outsourcing

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Matthias Christoph Huber

aus Achern

Tag der mündlichen Prüfung:	22.07.2016
Erster Gutachter:	Prof. Dr. Jörn Müller-Quade
Zweiter Gutachter:	Prof. Dr. Ralf Reussner

Acknowledgement

I am fortunate to have had the support of outstanding people in accomplishing this work.

First an foremost, I would like to thank my advisor Jörn Müller-Quade for his support, his guidance, his optimism, and his kindness. I am very grateful for the inspiring discussions we had over the years. Moreover, I would like to thank Ralf Reussner, the co-referee of my thesis, for his valuable counsel. Furthermore, I want to thank all the other professors who took interest in this work for their valuable feedback: Bernhard Beckert, Klemens Böhm, Hannes Hartenstein and Achim Streit.

I was glad to work with many wonderful colleges, co-authors, and students who made my time at work memorable. In particular, I want to thank my colleges and roommates Erik Burger, Dirk Achenbach, Jochen Rill, and Patrik Scheidecker for their support and the many happy memories over the years. Moreover, I very much enjoyed working with my long time friend Nico Döttling. I also want to thank Matthias Gabel, Gunnar Hartung, Christian Henrich and Tobias Nilges for the inspiring discussions.

I also want to thank my friends and everyone I did not mention explicitly and who contributed, directly or indirectly, to this work.

Finally and most importantly, I am deeply grateful for my family and my girlfriend for their love and their support during hard times. Mama, Papa, Julia, Benjamin, and Maria thank you.

Abstract

Provable security is one of the greatest achievements of modern cryptography. When proving the security of a cryptographic scheme, its security properties are reduced to problems known or assumed to be hard to solve. Therefore, breaking the security of such a scheme in its security model involves solving the corresponding problem which is deemed infeasible for sufficiently large instances.

Provable security has many benefits. For example, it allows to design schemes that provide security against attacks that have not yet been conceived. Furthermore, formal security guarantees allow to compare the security of schemes without comparing the schemes itself which can be a tedious task.

A central focus of cryptographic research is to conceive more and more stronger security notions and to find schemes that fulfil such notions. For example, the gold standard for encryption schemes is the notion of *semantic security*, where an adversary is not allowed to learn even one bit about the plaintext. For many complex applications such as database outsourcing classical, strong security requirements imply methods with large overheads. On the other hand, there are more practical data outsourcing schemes that intuitively provide some security but do not have any proven security properties at all.

Relaxing the classical security requirements potentially allows for more efficient schemes while maintaining provable security properties. This implies a trade-off: The resulting security notion should be meaningful in the context of the application while at the same time it should allow for efficient schemes. A field where such weak security notions play an essential role is database privacy. While the intention of security notions for encryption schemes is to hide all information of the plain text from the adversary, so-called privacy notions describe a trade-off between the confidentiality of the original database and the usefulness of the disclosed database.

Database privacy is closely related to data outsourcing. Efficient data outsourcing schemes imply leakage of information about the data to the server. Therefore, a security notion for efficient data outsourcing schemes also describes a trade-off, here, between the confidentiality of data and queries and the efficiency of schemes fulfilling this notion.

The goal of this thesis is to bridge the gap between practical methods for data outsourcing and the field of cryptographic research that is concerned with formal security notions.

Starting in the field of data privacy, we provide a framework for defining and reasoning about privacy notions. In contrast to existing notions and frameworks, our framework allows for an intuitive definition of privacy notions by allowing to define sensitive predicates and explicitly limiting what an adversary is allowed to learn about them from a release. Furthermore, we provide meta notions for different privacy goals of data outsourcing and establish their relations.

As a main contribution of this thesis, we provide a meaningful security notion for database outsourcing and a practical scheme fulfilling this notion as well as implementations that demonstrate the viability. Therefore, we capture database outsourcing in a formal model and define our scheme with the means of this model. We prove the security of our scheme by a reduction to the security of an internally used encryption scheme. Additionally, we examine the design space for this scheme by discussing extensions and optimisations of this scheme for performance as well as for security.

Zusammenfassung

Die Beweisbarkeit der Sicherheit ist einer der größten Erfolge der modernen Kryptographie. Um die Sicherheit eines Verfahrens zu beweisen, werden seine Sicherheitseigenschaften auf ein Problem reduziert, von dem man weiß oder annimmt, dass es schwer zu lösen ist. Daraus folgt, dass die Sicherheit eines solchen Verfahrens innerhalb des Modells zu brechen das Lösen des zugehörigen Problems involviert. Für hinreichend große Instanzen des Problems wird dies als nicht machbar angesehen.

Beweisbaren Sicherheit hat viele Vorteile. Beispielsweise erlaubt sie die Konstruktion von Verfahren, die auch gegen unbekannte Angriffe sicher sind. Darüber hinaus erlauben formale Sicherheitsbegriffe den Vergleich der Sicherheit von Verfahren ohne die Verfahren selbst zu betrachten, was sehr aufwändig werden kann.

Ein Schwerpunkt der Forschung in der Kryptographie ist es immer stärkere Sicherheitsbegriffe und Verfahren zu finden, die diese erfüllen. Beispielsweise darf ein Angreifer bei der *semantischen Sicherheit*, dem Goldstandard der Sicherheitsbegriffe für Verschlüsselung, nicht ein Bit an Information über den Klartext lernen. Für viele komplexe Anwendungen, wie beispielsweise das Auslagern von Datenbanken, implizieren solche starken Sicherheitsanforderungen Verfahren mit hohen Kosten. Andererseits gibt es praktikablere Verfahren für das Auslagern von Datenbanken, welche intuitiv ein gewisses Sicherheitsniveau bieten, jedoch keine bewiesene Sicherheitseigenschaften haben.

Eine Lockerung der klassichen, staken Sicherheitsanforderungen könnte effizientere Verfahren zulassen, ohne jedoch beweisbare Sicherheitseigenschaften zu verlieren. Dies impliziert ein Kompromiss: Die resultierende formale Sicherheitseigenschaft muss für die Anwendung sinnvoll sein und gleichzeitig effiziente Verfahren zulassen. Bei der Privatheit von Datenbanken spielen solche schwachen Sicherheitsbegriffe eine essentielle Rolle. Während Sicherheitsbegriffe für Verschlüsselung darauf abzielen, jegliche Information vor dem Angreifer zu verstecken, beschreiben sogenannte Privatheitseigenschaften einen Kompromiss zwischen der Vertraulichkeit der Originaldatenbank und der Nützlichkeit der veröffentlichten Informationen.

Die Privatheit von Datenbanken ist eng verwandt mit dem Auslagern von Datenbanken. Effiziente Verfahren für das Auslagern von Datenbanken implizieren einen Abfluss von Informationen über die Originaldatenbank zum Server. Folglich beschreibt ein Sicherheitsbegriff für das Auslagern von Datenbanken einen Kompromiss zwischen der Vertraulichkeit der Originaldatenbank und der Effizienz der Verfahren, die diesen Begriff erfüllen.

Das Ziel dieser Dissertation ist zwischen praktikablen Verfahren für das Auslagern von Datenbanken und formalen Sicherheitsbegriffen aus der Kryptographie eine Brücke zu schlagen.

Ausgangspunkt ist hierbei die Privatheit von Datenbanken. Wir definieren ein Rahmenwerk, welches es erlaubt formale Privatheitsbegriffe zu definieren und zu analysieren. Im Gegensatz zu existierenden Begriffen und Rahmenwerken erlauben wir eine intuitive Definition von Privatheitsbegriffen aufgrund der Definition sensibler Aussagen und einer expliziten Begrenzung, was ein Angreifer aufgrund einer Veröffentlichung über diese Aussagen lernen darf. Darüber hinaus definieren wir abstrakte Begriffe für die unterschiedlichen Privatheitsziele beim Auslagern von Daten und zeigen, wie sie zusammen hängen.

Hauptbestandteile dieser Dissertation sind ein Sicherheitsbegriff, der für das Auslagern

von Datenbanken sinvoll ist, ein praktikables Verfahren, das diesen Begriff erfüllt, sowie Implementierungen anhand derer wir die Realisierbarkeit des Verfahrens zeigen. Dazu formulieren wir das Auslagern von Datenbanken in einem Modell und definieren das Verfahren darin. Wir beweisen die Sicherheit des Verfahrens durch eine Reduktion auf die Sicherheit eines intern genutzen Verschlüsselungsverfahrens. Zusätzlich zeigen wir durch eine Diskussion von Erweiterungen und Optimierungen für die Sicherheit als auch für die Effizienz des Verfahrens weitere Varianten für unser Verfahren auf.

Contents

1. Preamble

1. Introduction

One of the greatest achievements of modern cryptography is the provable security methodology. In order to prove the security of cryptographic schemes their security properties are precisely formulated in a mathematical model. These formalisations are called security notions. Depending on the scheme, one can prove information theoretic security or reduce the security properties to a computational hardness assumption. Since information theoretic security harder to achieve, the latter case, which is also known as computational security, is more common. A computational hardness assumption is an assumption about the computational complexity of a problem. Noted hardness assumptions are for example integer factorisation, the discrete logarithm problem, and the shortest vector problem. Reducing the security of a cryptographic scheme to such an assumption implies that successful attacks involve the solution of the corresponding problem.

The benefits of the provable security methodology are widespread. The reduction to a hardness assumption provides that the security properties hold against any adversary for which this assumption holds: in general adversaries with a computational power that is polynomially restricted in the size of the problem. This also affects unknown adversaries and, therefore, even attacks that have not yet been conceived, as long as they are conducted in the formal model. Therefore, a formal security proof excludes huge classes of attacks. Furthermore, security notions allow to compare different cryptographic schemes on an abstract level. They allow the discussion and examination of the security properties of a scheme without the need to consider the scheme itself. This abstraction eases the definition of more complex provably secure, cryptographic schemes that use simpler ones as building blocks [Can01]. Classical examples for cryptographic schemes with provable security and formal security notions are encryption schemes, signature schemes, and secure multiparty computations. Because of the benefits of provable security, it is desirable to apply this approach to more and more complex applications.

A central focus of cryptographic research is to conceive more and more stronger security notions and to find schemes that fulfil such notions. For example, the gold standard for encryption is the notion of semantic security [GM82], where an adversary is not allowed to learn even one bit of information about the plaintext. For many complex applications, such strong security requirements imply methods with large overheads. Fully homomorphic encryption schemes [Gen09] or secure multi party computation schemes [GMW87], for example, allow to realise certain data and computation outsourcing schemes with strong security properties. Such schemes, however, imply a huge overhead that, in most cases, cancels the benefits of outsourcing. On the other hand, there are more practical data outsourcing schemes with the goal to provide some security but without any proven security properties at all.

This implies the question, if there is something in between. Can schemes for complex

applications be found that are more practical that classical cryptographic schemes, but at the same time maintain meaningful provable security properties? There are, for example, no efficient single server private information retrieval (PIR) schemes [SC07]. Deliberately allowing for some information to leak to the adversary, results in a relaxed security notion. This information may be useful in order to design more efficient schemes. This implies a trade-off: The resulting security notion should be meaningful in the context of the application while at the same time it should allow for efficient schemes.

A field where weak security notions play an essential role is database privacy. Database privacy deals with the problem of disclosing a database for subsequent analysis in a privacy preserving way. Therefore, the database is transformed prior to publication. While the intention of security notions for encryption schemes is to hide all information of the plaintext from the adversary, privacy notions describe a trade-off between the confidentiality of the original database and the usefulness of the disclosed database. Because of trends such as Cloud Computing, Machine Learning and Big Data, recently, database privacy has received much attention. The current gold standard for data privacy is the privacy notion Differential Privacy [Dwo08a]. If a mechanism fulfils differential privacy, its output is only marginally affected by whether an individual is represented in the original database or not. Privacy notions can seen as weak security notions that either implicitly or explicitly model the information that leaks to the adversary.

Therefore, privacy notions are a subject worthwhile to be researched, not only in the context of database privacy. The approach to explicitly model information that leaks to the adversary, and, in a second step, exploit this information in order to build efficient schemes is promising. Secure database outsourcing deals with the problem encrypting a database and storing it on a server in a way that it still can be queried efficiently. This field of research is closely related to database privacy, private information retrieval, and searchable encryption and, therefore, benefits from researching privacy notions in its context. Encrypting the database as a whole with a, for example, IND-CCA secure encryption scheme results in strong security guarantees. Most queries, however, can now only be executed by downloading end decrypting the whole database. Consequently, more efficient solution must involve partial execution of the query on the server. This, however, implies leakage of information to the server. Privacy notions seem like an appropriate tool to express this leakage and to formalise the security properties of database outsourcing schemes.

As mentioned above, there are practical database outsourcing schemes that intuitively provide some privacy but do not have any formal security notion. For database outsourcing, this arises the research question if there are secure database outsourcing schemes that support efficient execution of queries yet have meaningful and provable security properties.

2. Contribution and Structure of this Thesis

The goal of this thesis is to bridge the gap between practical methods for data outsourcing and the field of cryptographic research that is concerned with formal security notions. In this work, we show that there are methods for database outsourcing that are practical yet have provable and meaningful security properties. These properties define a trade-off between security and practicability with, in contrast to classical cryptographic methods, weaker, application specific security. Starting point are notions from the field of data

privacy. Then, we define privacy goals of data outsourcing and a security notion for database outsourcing as well as a scheme fulfilling this notion. The contributions of this work are:

- A framework for the definition and analysis of statistical and computational privacy notions. Furthermore, results concerning the composability of privacy notions.
- Meta notions that address different privacy goals of data outsourcing as well as results about their relations. These notions allow for formalising the security of encryption schemes as well as the security of data outsourcing schemes during usage.
- A security notion for database outsourcing and an efficient scheme for database outsourcing that fulfils this notion.
- Theoretical performance evaluations of this scheme as well as benchmarks of two implementations that serve as a validation of the applicability of our work. Furthermore, we provide optimisations for security in the presence of queries as well as performance optimisations of the data structures.

In the following, we provide a brief summary of the chapters of this thesis.

- In Chapter 2, we provide the technical background for the following chapters. We present notations and definitions used throughout this thesis.
- In Chapter 3, we introduce into data privacy and privacy notions. We present the concept of database anonymisation and different privacy notions that can be found in literature. Furthermore, we motivate semantic privacy notions by presenting and discussing weaknesses of syntactic notions.
- In Chapter 4, we present the Bayes Privacy framework that, similar to the Pufferfish framework presented in [KM12], allows for modelling privacy notions. The Bayes Privacy framework, while encompassing the Pufferfish framework, is a different approach to privacy notions by explicitly limiting privacy breaches. Furthermore, we provide results regarding the composability of privacy notions.
- In Chapter 5, we formalise data outsourcing an provide meta notions for different privacy goals of data outsourcing. We show how these goals are related and categorise existing notions from literature regarding these goals. Furthermore, we present generalisations of the meta notions that allow the definition of application specific notions that deliberately leak information to the adversary.
- In Chapter 6, we present two static security notions and a notion that provides data privacy if the adversary observes the execution of queries. We show that these notions are instances of the meta notions from Chapter 5. Furthermore, we discuss how the basic notion defined in this chapter is meaningful for database outsourcing.
- In Chapter 7, we provide schemes that fulfil the notions from Chapter 6. We present two implementations of our scheme and provide benchmarks. Furthermore, we discuss optimisations and provide benchmarks for selected optimisations.

2. Foundations

In addition to the foundations in this chapter, there are chapter specific preliminaries in each chapter.

1. Notations

We denote the *assignment* of a value a to a variable v with the notation $v \leftarrow b$. In order to assign a uniformly distributed random element of a set B to a variable v, we use the notation $v \leftarrow S$. For *conditional statements* we use the clause *If* in a standard way. We denote the concatenation of two strings a and b by $a|b$, the i-th element of a list or a set c with c_i, and the number of elements on that two given sets a and b differ with $a \Delta b$. If not stated otherwise, the characters i, j, k, l, m, n are natural numbers.

Throughout this work, we will use the letter k as a security parameter and the letter K as a cryptographic key. Furthermore, we will us the term *id* in order to denote the identity function.

If $G(\cdot, \cdot)$ is an algorithm or an oracle with two parameters, $G(\cdot, K)$ is the same algorithm or oracle where the second parameter is hard-wired to K.

1.1. O Notation and Negligibility

Definition 1 (O). *Let f and g be functions $\mathbb{N} \to \mathbb{R}$ and $k \in \mathbb{N}$. We write $f(k) \in O(g(k))$ or $f(k) = O(g(k))$ if there exists a constant $c > 0$ and a threshold $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for all $n > n_0$.*

Definition 2 (poly). *Let f be a function $\mathbb{N} \to \mathbb{R}$ and $k \in \mathbb{N}$. We write $f(k) \in \text{poly}(k)$ or $f(k) = \text{poly}(k)$ if there exists a constant $c > 0$ such that $f(k) \in O(k^c)$.*

Definition 3 (Negligibility). *Let f be a functions $\mathbb{N} \to \mathbb{R}$. We say f is negligible if for every constant $c > 0$ if there exists a threshold $n_0 \in \mathbb{N}$ such that $f(n) < n^{-c}$ for all $n > n_0$.*

2. Probability Theory and Statistics

Definition 4 (Conditional Probability). *Let A and B be events. The* conditional probability $\Pr[A|B]$ *of A given B is defined as:*

$$\Pr[A|B] := \frac{\Pr[A \cap B]}{\Pr[B]}$$

Lemma 1 (Bayes' Theorem and the Law of Total Probability)**.** *Let A and B be events. Then the following holds:*

$$\Pr[A|B] = \frac{\Pr[B|A]\Pr[A]}{\Pr[B]} \quad \textit{(Bayes' Theorem)} \tag{2.1}$$

$$\Pr[A] = \sum_j \Pr[A, B_j]\Pr[B_j] \quad \textit{(law of total probability)} \tag{2.2}$$

$$\Pr[A] = \Pr[A, B]\Pr[B] + \Pr[A, \neg B]\Pr[\neg B] \quad \textit{(law of total probability for binary events)} \tag{2.3}$$

This lemma can be proven directly with Definition 4.

3. Data Sets and Databases

Definition 5 (Data Set)**.** *A* data set $d \in D$ *is an element of a domain D.*

We define a database, a special case of a data set, as a multiset of tuples:

Definition 6 (Database)**.** *A* database $d = \{d_1, d_2, \dots, d_n\}$ *is a finite multiset of tuples from the same set $A_1 \times A_2 \times, \dots, A_m$. We call the number of tuples of d the size of d and denote it with $|d|$ We call the set $\{A_1, A_2, \dots, A_m\}$ the attributes of d.*

Let $i \in \{1, \dots, n\}$ and $j \in \{1, \dots m\}$. For an arbitrary but fixed order of d, we call the tuple $d_i \in d$ row i of d and denote it with $d(i, \cdot)$. We call the j-th element of row i of d the value of attribute A_j in row i of d and denote it with $d(i, j)$. We call the multiset of all values of attribute A_j of d column j of d and denote it with $d(\cdot, j)$.

We call the set of all databases DB.

Throughout this work, we use the terms tuple and row interchangeably. Please note that, since we defined a database as a multiset, there is no specific order of its elements. Since a finite set of tables always can be normalised to a single table, we do not distinguish between a table and a database.

Definition 7 (Selection)**.** *A* selection $\sigma_{A\theta v}(d)$ *or* $\sigma_{A\theta B}(d)$ *is a unary operation where*

- *d is a database*
- *A and B are attributes of d*
- *v is a value of the domain of A*
- *θ is a binary operation from the set $\{\leq, \leq, =, \geq, \geq, \neq\}$*

and the result of the selection $\sigma_{A\theta v}(d)$ are the tuples of d where $A\theta v$ holds and the result of $\sigma_{A\theta B}(d)$ are the tuples of d where $A\theta B$ holds. We call $A\theta v$ or $A\theta B$ the condition *of the selection and the attribute A the attribute of the selection.*

Definition 8 (Generalised Selection)**.** *A* generalised selection *is a unary operation $\sigma_c(d)$ where c is a propositional formula of conditions and the logical operators $\vee$, $\wedge$, and $\neg$. The result of $\sigma_c(d)$ are the tuples of d for which c holds. We call c the* conditions *of the selection.*

A selection is a special case of a general selection. In this work, we use the term selection for both.

Definition 9 (Projection)**.** *Let d be a database of size n with attributes $\{A_1, A_2, \dots, A_m\}$ and $l \leq m$. A projection is a unary operation $\Pi_{A_{\Pi_1},\dots A_{\Pi_l}}(d)$ with:*

$$\Pi_{A_{\Pi_1},\dots A_{\Pi_l}}(d) := (d(i,j) \mid \forall j \in \{\Pi_1, \dots \Pi_l) \mid \forall i \in \{1, \dots n\}$$

The result of a selection is a subset of the columns of the original database.

4. Cryptographic Mechanisms and Notions

Definition 10 (probabilistic polynomial time (PPT))**.** *An algorithm or Touring machine is a* probabilistic polynomial time (PPT) *algorithm or Touring machine if it is in* $\mathrm{poly}(k)$ *for a security parameter k.*

Definition 11 (Symmetric Encryption Scheme)**.** *A symmetric encryption scheme is a tuple* (Gen, Enc, Dec) *of three PPTs such that:*

- $\mathsf{Gen} : 1^k \to \{0,1\}^n$
- $\mathsf{Enc} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^*$
- $\mathsf{Dec} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^*$
- $\forall x \in \{0,1\}^*, K \in \{0,1\}^n : \mathsf{Dec}(\mathsf{Enc}(x,K),K) = x$

We call Gen *the key-generation mechanism,* Enc *the encryption mechanism,* Dec *the decryption mechanism, K the encryption key, k the security parameter, the output of* Enc *as well as the first input of* Dec *the ciphertext, and the output of* Dec *as well as the first input of* Enc *the plaintext.*

Definition 12 (Block Cipher)**.** *A block cipher is a symmetric encryption scheme* (Gen, Enc, Dec) *with:*

- $\mathsf{Enc} : \{0,1\}^b \times \{0,1\}^n \to \{0,1\}^b$
- $\mathsf{Dec} : \{0,1\}^b \times \{0,1\}^n \to \{0,1\}^b$

We call $\{0,1\}^b$ a ciphertext or plaintext block, respectively, and b the block length.

Definition 13 (Cipher Block Chaining (CBC))**.** *Let P be a vector of plaintext blocks with block length b. The Cipher Block Chaining (CBC) mode is a mode of operation of a block cipher* (Gen, Enc, Dec) *with:*

- $C_0 := IV$
- $C_i := \mathsf{Enc}(P_i \oplus C_{i-1}, k)$

where $IV \in \{0,1\}^b$ an initialisation vector.

This definition implies, that $P_0 = IV$ and $P_i = \mathsf{Dec}(C_i, K) \oplus C_{i-1}$.

4.1. Game-Based Security Notions

Game-based security notions are a standard way of defining security for cryptographic schemes. They are modelled as an interactive protocol between an experiment and an adversary $\mathcal{A}$, where the adversary is presented with a challenge [Nao03; Pas11]. We call this protocol the security game. In this protocol, the experiment as well as the adversary receive the length 1^k of the security parameter k. In the following rounds of interaction, the adversary may receive additional information, access to oracles and a challenge. In the end, the experiment computes an output bit. We say, the adversary wins the experiment if the bit is 1 and looses the experiment if the output bis is 0. We consider a scheme as secure, if the success probability of any PPT adversary is at most negligibly better than the success probability of guessing the result. A well-known game-based security notion for encryption schemes is IND-CPA, which is defined in the following.

4.2. IND-CPA Security

Definition 14.

Security Game 1 ($\text{IND-CPA}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \text{Gen}(1^k)$.
2. *The adversary* $\mathcal{A}$ *is given input* 1^k *and oracle access to* $\text{Enc}(\cdot, K)$.
3. $\mathcal{A}$ *outputs two plaintexts* m_0 *and* m_1 *with the same length.*
4. *The experiment chooses a random bit* $b \leftarrow \{0, 1\}$.
5. $\mathcal{A}$ *is given* $\text{Enc}(m_b, K)$.
6. $\mathcal{A}$ *outputs a guess* b' *for* b.

The result of the experiment is 1 *if* $b' = b$ *and* 0 *else.*

Definition 15 (Indistinguishability under Chosen Plaintext Attacks)**.** *An encryption scheme* (Gen, Enc) *has* Indistinguishability under Chosen Plaintext Attacks *IND-CPA if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function negl such that:*

$$\Pr[\textit{IND-CPA}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}(k) = 1] \leq \frac{1}{2} + \textit{negl}(k)$$

Informally, this means that a scheme provides IND-CPA it the adversary has only negligible advantage over guessing in the security game $\text{IND-CPA}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}(k)$. An extension of the IND-CPA-experiment is the IND-MULT-CPA-experiment, where the adversary may choose two vectors of plaintext.

Definition 16.

Security Game 2 ($\text{IND-MULT-CPA}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \text{Gen}(1^k)$.
2. *The adversary* $\mathcal{A}$ *is given input* 1^k *and oracle access to* $\text{Enc}(\cdot, K)$.

3. *$\mathcal{A}$ outputs two vectors of plaintexts $M_0 = (m_0^0, \dots, m_n^0)$ and $M_1 = (m_0^1, \dots, m_n^1)$ where m_i^0 and m_i^1 have the same length for $i = 0..n$.*
4. *The experiment chooses a random bit $b \leftarrow \{0, 1\}$.*
5. *$\mathcal{A}$ is given $\mathsf{Enc}(M_b, K) = (\mathsf{Enc}(m_0^b, K), \dots, \mathsf{Enc}(m_n^b, K))$.*
6. *$\mathcal{A}$ outputs a guess b' for b.*

The result of the experiment is 1 if $b' = b$ and 0 else.

Definition 17 (Indistinguishability under Multiple Chosen Plaintext Attacks)**.** *An encryption scheme* (Gen, Enc) *has* Indistinguishability under Multiple Chosen Plaintext Attacks *IND-CPA if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that:*

$$\Pr[\textit{IND-MULT-CPA}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}(k) = 1] \leq \frac{1}{2} + \mathit{negl}(k)$$

The security notions IND-CPA and IND-MULT-CPA imply each other:

Theorem 1 (IND-CPA $\Leftrightarrow$ IND-MULT-CPA)**.** *Any private-key encryption scheme that has IND-CPA security also has IND-MULT-CPA security and vice versa.*

Proof. The direction IND-MULT-CPA $\Rightarrow$ IND-CPA is trivial: The adversary $\mathcal{A}_{\text{IND-CPA}}$ can directly be used in the experiment $\text{IND-MULT-CPA}_{(\mathsf{Gen},\mathsf{Enc})}$ and we inherit its success probability. A proof for the direction IND-CPA $\Rightarrow$ IND-MULT-CPA can be found in [KL07]. □

3. An Introduction to Privacy

This chapter is partially based on work already published in [Hub+11; HM11; Hub10] and in [HMN14].

1. What is Privacy?

In their article *The Right to Privacy* [WB90], Samuel Warren and Louis Brandeis define privacy as *the right to be left alone*. They argued that there is value in the prevention of publication of *thoughts, sentiments, and emotions*. They analysed the laws at that time and came to the conclusion that these laws did not protect this value. The notion of privacy was born. This notion lead to modern privacy laws such as the German Bundesdatenschutzgetz [Bun78], one of the most progressive laws that not only protects thoughts, sentiments, and emotions, but all data related to a single individual. It forbids processing data related to single individuals without a legal basis. Furthermore, it even defines a class of particularly sensitive data (*besondere Arten personenbezogener Daten*) and the principle of data minimisation (*Datensparsamkeit*) that states that you may process just as little data related to single individuals as necessary.

In classical cryptography, on the other hand, privacy is often used as a synonym for confidentiality. In the context of encryption, for example, privacy means the confidentiality of messages. A voting scheme that provides privacy for the voters [Dem+13; Lan+09] guarantees the confidentiality of single votes. A mix network that provides privacy guarantees the confidentiality of the origin of a message [BDG13]. It hides the identity of the sender of a message from the recipient.

Recently, the social and judicial idea of privacy has also been considered from a cryptographic point of view in the field of database privacy. Here, the goal is to disclose a database, for example, to third parties that want to analyse it [Swe02]. Applications that combine and analyse large amounts of data from different sources promise benefits for science and economy. Since a database can contain sensitive information that must not be disclosed, it has to be sanitised or anonymised prior to disclosure. The goal of the sanitation or anonymisation is to prevent the disclosure of sensitive data. Consider an institution (e.g. the census bureau or a hospital organisation) wanting to disclose a database in order to allow for data mining by third parties. Since the privacy of the individuals of the database has to be protected, the database has to be anonymised prior to disclosure (cf. Figure 3.1). On the other hand, the disclosed database still has to hold information. Otherwise disclosing and analysing it would be pointless. Consequently, a trade-off between the protection of sensitive data and the usefulness of the anonymised database – a trade-off between privacy and utility – has to be found.

Another application related to database privacy is secure database outsourcing. Here, a client wants to outsource a database to a not necessarily trustworthy party. For reasons of efficiency, this party should be able to execute queries on the database on behalf of the client while learning as little as possible about the data [HIM02; Pop+11; Hub+13;

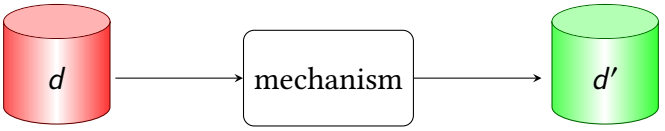


Figure 3.1.: Database privacy: In order to protect the sensitive data in the original database d, a mechanism transforms d to a result d' that is disclosed. In order to enable further processing, the utility of the original data has to be preserved while sensitive data has to be changed or removed.

AGH11]. Again, there is a trade-off between privacy and utility. Here, the utility is needed in order to support efficient execution of queries. Therefore some structure of the original database has to be preserved. We will examine secure database outsourcing in this work in Chapters 6-7.

As mentioned above, in contrast to confidentiality for message encryption, in database privacy, leakage of some data is desired – even necessary [Dwo08a] – while the privacy of individuals has to be protected. An adversary should not be able to learn sensitive information from the disclosed database. On the other hand it should still have utility in the sense that the disclosed data is useful, e. g. for data mining. This trade-off between utility and privacy is described by formal privacy notions. Privacy notions allow for a fine grained definition of privacy. Privacy notions allow to distinguish between sensitive data that should be protected (e. g. sensitive data about individuals) and data that is not sensitive and therefore can be disclosed. Furthermore, there are privacy notions that even introduce a measure for the magnitude of the disclosure of sensitive data.

Privacy notions can be classified into syntactic and semantic notions. In order to provide a motivation for semantic notions, that are designed with adversaries in mind, we present a discussion of weaknesses of syntactic notions found in literature.

Structure of this Chapter Section 2 introduces the concepts of database anonymisation and of privacy notions. Furthermore, we present and discuss syntactic and semantic privacy notions and frameworks for privacy notions found in literature. In Section 3, we present attacks on and subliminal channels of syntactic privacy notions.

2. Anonymisation and Privacy Notions

This section presents the concepts of *database anonymisation* and *privacy notions*. In this work, database anonymisation is used in two scenarios, privacy preserving database disclosure and secure database outsourcing, which will be introduced in this section. There is a rich body of literature on privacy notions, which also will be discussed in this section.

2.1. Database Anonymisation

Database anonymisation is the transformation of a database by an mechanism, if either the mechanism itself or the result of the transformation fulfils a privacy notion (cf. Section 2.2). The goal of database anonymisation is to protect sensitive data from the public or an untrusted party while preserving as much information of the original database as possible.

Note that this sensitive data must not necessarily be data about individuals. We will use examples with data about individuals but also databases that contain intellectual property, for example sensitive measurement values of a production line, can be transformed in order to protect the sensitive data. In both cases, we will call this process anonymisation. There are two important scenarios for database anonymisation. The first scenario is *privacy preserving database disclosure*, the second scenario is *secure database outsourcing*. In the following, these scenarios will be presented.

2.1.1. Privacy Preserving Database Disclosure

In privacy preserving database disclosure, a database containing sensitive data is transformed in order to be disclosed. Therefore, the database is anonymised prior to disclosure.

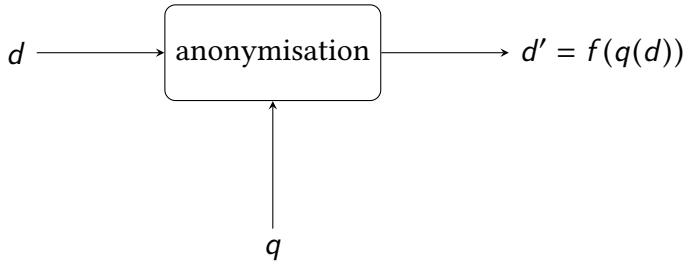
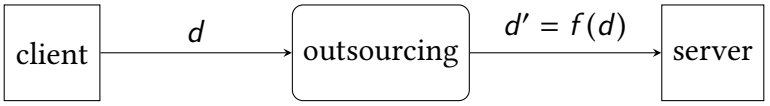


Figure 3.2.: Privacy preserving database disclosure: An anonymisation mechanism transforms a database d to an anonymised result d'. Then, the result d' is disclosed. The goal is to protect sensitive data in d, for example sensitive data about individuals, while preserving the utility of the data, in order to enable further processing. In some cases, clients may specify a function q that is applied to d prior to anonymisation.

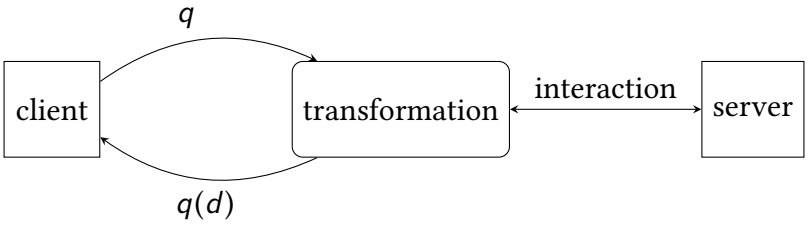
An *anonymisation mechanism* is applied to the database. This scenario is depicted in Figure 3.2: An anonymisation mechanism transforms a database d into the result d'. In some scenarios, clients are allowed to specify a function q that is applied to d prior to anonymisation. This scenario is called *online privacy preserving database disclosure*. It is mainly used in the context of Differential Privacy [Dwo08a], where clients are allowed to issue queries to a curator that can query the original database and anonymises the result of the query. The anonymisation mechanism f can be a probabilistic algorithm and not necessarily is a function. For example, since the optimal k-anonymity problem is hard [MW04], in practice polynomial-time heuristics are used. Differential Privacy and k-anonymity will be discussed in detail in Section 2.2.

2.1.2. Secure Database Outsourcing

Another scenario for database anonymisation is *secure database outsourcing*. Here, a client wants to outsource a database to a not necessarily trustworthy server. In contrast to privacy preserving database disclosure, the client wants to be able to access the data in the outsourced database efficiently. Therefore, the server has to be able to execute queries efficiently on behalf of the client. On the other hand, the server should learn as little as possible about the original database. Therefore the original database d is transformed. This scenario is depicted in Figure 3.3. In order to query the outsourced database, a query q intended for the original database d has to be transformed in order to execute it on



3.3.1: secure database outsourcing: outsourcing step



3.3.2: secure database outsourcing: query handling

Figure 3.3.: Secure database outsourcing: An outsourcing mechanism transforms a database d to an result d'. Then, the result d' is stored on a not necessarily trustworthy server (Figure 3.3.1). In order to query the database stored on the server, queries intended for the database d have to be transformed into an interactive protocol (Figure 3.3.2). The Server has to be prevented from learning the original database d, however, the client has to be able to query the outsourced database efficiently. Therefore, the server has to be able to support execution of queries on d' on behalf of the client.

the outsourced database. In general, a query is transformed to an interactive protocol between the client and the server. Again, there is a trade off between privacy and utility. Here, the utility affects the efficient support for execution of queries. Additionally, there is a requirement to the anonymisation mechanism, that is not necessarily needed in privacy preserving database disclosure: The anonymisation mechanism has to be reversible. Since the client wants to use the result of the anonymisation as it would use the database d, the client needs a trapdoor to reconstruct the original database d from the anonymised database d'.

This chapter as well as Chapter 4 focus on privacy preserving database disclosure, while Chapters 5 and 7 focus on privacy for database outsourcing.

2.2. Privacy Notions

In order to reason about the security properties of an anonymisation mechanism or its result, we need a formal description. A privacy notion is such a formal description. The intention of privacy notions is to precisely describe the level of privacy provided either by an anonymisation mechanism or by an anonymised release.

Privacy notions can be classified into *structural* or *syntactic notions* and *semantic* or *cryptographic notions* [De +12]. Structural privacy notions formalise syntactic requirements for the result of the anonymisation process. Cryptographic privacy notions are designed with an adversary in mind and therefore describe semantic guarantees. They can be compared with security notions for encryption. But, in contrast, cryptographic privacy notions do not try to prevent leakage of information completely. In order to maintain the utility of the anonymisation result, they allow for leakage of some information.

Examples for structural notions are k-anonymity [SS98] and notions based thereon.

They introduce constraints for the anonymisation result. Such notions do not consider adversaries that try to break the anonymisation directly. This will be further discussed in Section 3.

An example for a cryptographic notion is Differential Privacy [Dwo08a]. Such notions introduce constraints for the anonymisation process with with respect to adversaries. These adversaries can be computationally bounded [Mir+09] or unbounded [Dwo08a]. Computationally bounded adversaries are interesting for notions that describe mechanisms based on computational problems. In Chapter 7 we will present methods for secure data outsourcing that involve encryption.

Privacy notions do have standard methods that are used to achieve them (e. g. adding noise for Differential Privacy, or coarsening attribute values for k-anonymity – cf. Sections 2.2.1 and 2.2.2). Nevertheless, we stress that we have to distinguish between methods and guarantees. While Differential Privacy is often achieved by the addition of Laplace noise, another approach to achieve Differential Privacy, for example, is presented in [Bha+11]. This approach does not add noise to the release, but uses the uncertainty of the adversary about the original database or the intrinsic entropy of the original database to construct privacy preserving mechanisms. On the other hand, the addition of Laplace noise does not yield differentially private methods in general, but only under certain assumptions about the distribution of attribute values.

A privacy notion is a guarantee that can be achieved by an anonymisation mechanism. We use the word mechanism instead of function since a mechanism can also be a probabilistic algorithm. If a privacy notion is achieved by a mechanism, we say that this mechanism fulfils this privacy notion.

Privacy notions can be used to reason about anonymisation mechanisms. For example, if a mechanism fulfils the privacy notions A and B, the mechanism can be considered to be more secure than another mechanism that only fulfils privacy notion A. The Bayes Privacy framework that we will present in Chapter 4 allows for modelling and for the comparison of privacy notions without explicitly considering mechanisms.

2.2.1. Structural Privacy Notions

The first attempt to formally capture database privacy resulted in the concept of *k-anonymity* by Samarati and Sweeney [SS98; Swe02]. The idea of this notion is to achieve anonymity by letting each individual blend into a crowd of at least k individuals. The notion k-anonymity implicitly assumes that sensitive traits are rare and that an individual can not be identified by the sensitive trait itself. Then, we can classify attributes about individuals into *quasi-identifier* and *sensitive information*. The quasi-identifier $\mathcal{QI}$ is the set of attributes that allow, linked with external data, to uniquely identify at least one individual of a table. Formally, k-anonymity is defined as follows (taken from [Mac+07]):

Definition 18. *A table T satisfies k-anonymity if for every tuple $t \in T$ there exist $k-1$ other tuples $t_{i_1}, t_{i_2}, \ldots t_{i_{k-1}} \in T$ such that $t[C] = t_{i_1}[C] = t_{2_1}[C] = \ldots = t_{i_{k-1}}[C]$ for all C in $\mathcal{QI}$.*

In this definition, $t[C]$ denotes the projection of the tuple t to the attributes C. This definition implies, that the result has blocks of rows, where the attribute values of the quasi-identifier are identical. In literature, the k-anonymity property is achieved by *supression* and *generalisation*. Attribute values of the quasi-identifier attributes are

either suppressed or generalised until the table fulfils k-anonymity. An example for an 4-anonymous table is depicted in Figure 3.4.2.

	Non-Sensitive			Sensitive
	Zip Code	Age	Nationality	Condition
1	13053	28	Russian	Heart Disease
2	13068	29	American	Heart Disease
3	13068	21	Japanese	Viral Infection
4	13053	23	American	Viral Infection
5	14853	50	Indian	Cancer
6	14853	55	Russian	Heart Disease
7	14850	47	American	Viral Infection
8	14850	49	American	Viral Infection
9	13053	31	American	Cancer
10	13053	37	Indian	Cancer
11	13068	36	Japanese	Cancer
12	13068	35	American	Cancer

3.4.1: A medical table with the sensitive attribute *condition* and the non-sensitive attributes *Zip Code*, *Age*, and *Nationality*

	Non-Sensitive			Sensitive
	Zip Code	Age	Nationality	Condition
1	130**	≤ 30	*	Heart Disease
2	130**	≤ 30	*	Heart Disease
3	130**	≤ 30	*	Viral Infection
4	130**	≤ 30	*	Viral Infection
5	1485*	≥ 40	*	Cancer
6	1485*	≥ 40	*	Heart Disease
7	1485*	≥ 40	*	Viral Infection
8	1485*	≥ 40	*	Viral Infection
9	130**	3*	*	Cancer
10	130**	3*	*	Cancer
11	130**	3*	*	Cancer
12	130**	3*	*	Cancer

3.4.2: A 4-anonymous version of the Table in 3.4.1. The attributes *Zip Code* and *Age* were generalised while the attribute *Nationality* was suppressed.

Figure 3.4.: An example for a medical table and a 4-anonymous version of it (Taken from [Mac+07]). The rows {1,2,3,4}, {5,6,7,8}, and {9,10,11,12}, each, form a bucket. In each bucket, the values of the attributes of the quasi identifier are identical.

Since this notion has several shortcomings, it was iteratively improved upon in order to remove the inherent problems, e.g. by Machanavajjhala et al. [Mac+07] in the form of *l-diversity* and by Li et al. [LLV07] in the form of *t-closeness*. The l-diversity principle requires diversity of the values of the sensitive information attribute within each block. Formally, l-diversity is defined as follows (taken from [Mac+07]):

Definition 19. *A $q*$-block is l-diverse if contains at least l* well-represented *values for the sensitive attribute S. A table is l-diverse if every $q*$-block is l-diverse.*

For different instantiations of well-represented, please refer to [Mac+07]. The intention of this definition is to prevent *homogeneity attacks* [LLV07]. If the values of a sensitive attribute in a $q*$-block are similar, identical, or have the same meaning, a homogeneity attack allows to determine the value of the sensitive attribute for an individual.

Since l-diversity still had some well-known problems, a lot of variations and improvements followed [MW04; LDR05; BA05; TZ11]. Other approaches involve disassociation [Ter+12] or fragmentation [De +10; De +13] in order to achieve privacy. Here, coarsening is used to hide sensitive relations. The approach in [Tao+10] introduces an *independence masking* operator with the intention to completely eliminate correlations between two sets of attributes.

One problem, however, was not solved: The combination of anonymised data sets can lead to a deanonymisation of at least a part of the database, as was shown by Ganta et al. [GKS08]. Even today, most privacy notions suffer from this problem. It even has been proven that for any mechanism that has some utility, there is an adversary that can break any privacy notion [KM11]. Therefore, frameworks that also specify assumptions and restrictions about adversaries are needed. In contrast to semantic notions discussed in the next section, the notions above do not give a guarantee about the process, but describe the result of an process.

2.2.2. Cryptographic Privacy Notions and Frameworks

While the intuition behind k-anonymity and, therefore, its variations is reasonable, the formal instantiation as a syntactic notion allows for a variety of attacks. As it turns out, in general, the actual goal should not be to achieve anonymity in a crowd, but rather to directly hide the association of an individual with certain sensitive values from potential adversaries. An extreme method is to hide the sensitive value itself. In her seminal work Dwork [Dwo08a] introduced the notion of *Differential Privacy*, which is the first privacy notion defined with a cryptographic flavor. In contrast to previous work, Differential Privacy gives a guarantee that for any two databases that differ in a single entry the result of the anonymisation process cannot be distinguished except by a small factor. The intuition of Differential Privacy is that the adversary can not learn much about individuals if the fact that a single individual is present in the original database does not change the anonymisation result. Implicitly, Differential Privacy assumes that the probability of each tuple is independent. Formally Differential Privacy can be described as follows (taken from [Dwo08a]):

Definition 20 (ϵ-Differential Privacy)**.** *A randomized function $\mathcal{K}$ gives* Differential Privacy *if for all data sets D_1 and D_2 differing on at most one element, and all $S \subseteq Range(\mathcal{K})$*

$$\Pr[\mathcal{K}(D_1) \in S] \leq e^{\epsilon} \times \Pr[\mathcal{K}(D_2) \in S]$$

This can be achieved by adding noise from a Laplacian distribution to the actual result. In contrast to the approaches approaches based on k-anonymity that release the complete anonymised database, the definition of Differential Privacy does not restrict the range of the anonymisation mechanism to databases. Many mechanisms fulfilling Differential Privacy assume a trusted curator algorithm that answers queries about the data without releasing a completely anonymised database. Based on the concept of Differential Privacy, follow-up work has been presented, see [Dwo+06c; Dwo+06a; GLP11; Geh+12].

Differential Privacy is the current gold standard notion in privacy research, although there are some drawbacks. In [GLP11] Gherke et. al. show that mechanisms that achieve Differential Privacy when the probability of each tuple is independent can fail to achieve Differential Privacy for databases, where this is not the case. In social networks, for example, the probability of the presence of an individual can depend on the probability of the presence of the social cluster the individual is part of. Therefore, they define the notion *Zero Knowledge Privacy* which is even stronger as Differential Privacy.

Kifer and Machanavajjhala, on the other hand, show in [KM11] that even the definition of Differential Privacy is too strict in some cases and thereby diminishes the utility of the obtained data severely. The term utility in this context means the degree of distortion of the actual result. The main reason for this is that only worst case scenarios are considered and thus the amount of noise needed to achieve the desired privacy is very large. Two different solutions were proposed to circumvent this problem. One proposition is to use the noise that is implicit in the data such that the utility is not reduced by the addition of noise. The other approach is to weaken the definition such that a privacy breach is not impossible, but unlikely [Dua09; Bha+11].

Concerning the first approach, the assumption is made that an adversary does not have the complete knowledge of a database except for a single entry. It is assumed that the adversary's knowledge has a given uncertainty which can be described as a probability distribution on the attribute values. From an adversary's point of view this means that

the data has implicit noise. It was shown by Duan [Dua09] that this uncertainty can be enough to achieve Differential Privacy for sum queries. Additional privacy notion similar to Differential Privacy based on the implicit noise of the data were proposed [Bha+11]. Albeit, to maintain privacy for several queries it is necessary that some fraction of the database is updated between the queries, otherwise the adversary can learn the complete database over time. The aforementioned work also incorporates the second idea, namely a weakening of the privacy guarantee by allowing a privacy breach with small probability.

One of the first relaxations of Differential Privacy was put forward by Dwork et al. [Dwo+06c] and is called (ϵ, δ)-*Differential Privacy*, where δ signifies the probability that the factor between the two distributions is greater than e^{ϵ}. Other notions [Blu+05; CM06; EFW10; BLR08; GLP11] also seize this idea, although in somewhat different contexts. Before the upcoming of Differential Privacy, Chaudhuri and Mishra [CM06] used an (ϵ,δ)-privacy notion of, where they showed that a random subset of a database can be released privately if no values with low occurrence are in this subset. The work by Blum et al. [BLR08] is based on a concept that has proven to be of increasing interest. The knowledge of the adversary is modelled as a probability distribution of possible databases which is a generalisation of uncertainty of database entries. They release information for a class of queries where the utility of the information has only been reduced by a (fixed) small amount.

This concept was taken a step further by the recent results of Kifer and Machanavajjhala [KM12], who introduced the *Pufferfish* framework. Instead of a strict and static guarantee, they allow to specify certain predicates for which the privacy is enforced. This was already previously used by Blum et al. [Blu+05], where a priori and a posteriori beliefs over binary predicates were considered, and by Evfimievski et al. [EFW10] for the case where a change in the probability of a predicate indicates a privacy breach. The idea of Pufferfish is to specify two mutually exclusive predicates. One of the predicates is the sensitive predicate to be hidden. After anonymisation, an adversary should not be able to determine whether the first or the second predicate is true. Formally, a privacy notion in Pufferfish is defined as follows (taken from [KM12]):

Definition 21. *Given a set of potential secrets $\mathbb{S}$, a set of discriminative pairs $\mathbb{S}_{pairs}$, a set of data evolution scenarios $\mathbb{D}$, and a privacy parameter $\epsilon > 0$, a (potentially randomised) algorithm $\mathfrak{M}$ satisfies ϵ-PufferFish ($\mathbb{S}$, $\mathbb{S}_{pairs}$, $\mathbb{D}$) privacy, if*

- *for all possible outputs $\omega \in range(\mathfrak{M})$*
- *for all pairs $(s_i, s_j) \in \mathbb{S}_{pairs}$*
- *for all distributions $\theta \in \mathbb{D}$ for which $P(s_i|\omega) \neq 0$ and $P(s_j|\omega) \neq 0$:*

$$P(\mathfrak{M}(\mathfrak{Data}) = \omega|s_i, \theta) \leq e^{\epsilon} P(\mathfrak{M}(\mathfrak{Data}) = \omega|s_j, \theta)$$
$$P(\mathfrak{M}(\mathfrak{Data}) = \omega|s_j, \theta) \leq e^{\epsilon} P(\mathfrak{M}(\mathfrak{Data}) = \omega|s_i, \theta)$$

This definition of privacy notions on the basis of not necessarily exhaustive $\mathbb{S}_{\text{pairs}}$ might not seem intuitive at first. It, however, allows for mechanisms that release the original database where all sensitive predicates are false. If the $\mathbb{S}_{\text{pairs}}$ were exhaustive (i.e. for each $(s_i, s_j) \in \mathbb{S}_{\text{pairs}}$, either s_i is or s_j is true), this were not the case.

As Differential Privacy, Pufferfish requires the probability of an output given a certain predicate in the input to be close to the probability the same output given another predicate.

Pufferfish, however, allows domain experts to define their own pairs of predicates. It is e. g. possible to choose the predicates *value x is in the database* and *value x is not in the database*. With appropriate predicates, Differential Privacy can be modelled as a special case of this framework. Based on the Pufferfish framework, Kifer et al. extract a set of axioms for privacy notions [KL10; KL12] which, as they argue, should be fulfilled by any privacy notion. They define a privacy definition as a set of randomised algorithms. The first axiom states that any privacy definition must be invariant to transformations, i. e. the application of an arbitrary algorithm to the result does not affect the privacy of the result. The second axiom states that any two algorithms that satisfy a privacy definition can be applied interchangeably while still satisfying the privacy definition. Based on these axioms a mathematical analysis is possible, such that semantic guarantees of a privacy definition can be made explicit [LK12]. In Chapter 4, Section 3, we will discuss these axioms in the context of our Bayes Privacy framework.

3. Attacks on Structural Privacy Notions

The motivation to define privacy originated from the field of data mining. The basic idea to hide the association of individuals with sensitive values seems to be reasonable. As described in Section 2.2, the first attempts to formally capture privacy followed this idea by hiding individuals in a crowd. Formal notions like k-anonymity, l-diversity, t-closeness, and variations thereof, however, describe structural properties of the output. They do not restrict the anonymisation mechanism beyond these structural properties of the output. Furthermore, they do not consider explicitly what an adversary learns from a release.

This allows, for example, a malicious anonymiser to encode information into the result by simply adding additional rows to the database in such a way that the structural properties needed by the privacy notion are not violated. There are, however, also subliminal channels, that do not need a malicious anonymiser.

In the remainder of this section, we will show weaknesses in structural notions that are not already covered in the related work (cf. Sections 2.2 and 3.1). The first example requires a malicious anonymiser that wants to circumvent the privacy notion. The second example is a side channel that occurs because of the structural requirements to the release and does not require a malicious anonymiser.

3.1. Attacks on Structural Notions in Literature

In literature, there are two classes of attacks on structural privacy notions, namely the *homogeneity attack* [Mac+07] and the *background knowledge* or *composition attack* [GKS08; HMN14]. Summarising, both attacks use additional knowledge to break the anonymity notion.

The *homogeneity attack* exploits that all sensitive values in a bucket of a k-anonymous database can be identical or similar. Consider for example the table in Figure 3.4.2. The rows {9,10,11,12} form a 4-bucket. Since the only value of the sensitive attribute in this bucket is cancer, however, an adversary that knows an individual in the database that falls into this bucket learns that this individual has cancer.

The *composition attack* exploits the general fact that k-anonymity and many notions based thereon do not compose with background knowledge. Consider for example the

bucket {1,2,3,4} from the table in Figure 3.4.2. If an adversary knows an individual falling into this bucket and if she can rule out, for example from another 4-anonymous database, that this individual has a heart disease, the adversary learns that this individual has a viral infection. For another example consider the two 2-anonymous tables in Figure 3.6. These tables are derived from the same original table (cf. Figure 3.5). If these two tables, however, are joined, the adversary learns for each row the exact values of the attributes of the quasi-identifiers of the original database. This breaks the intention of the anonymity notion.

The homogeneity attack and the composition attack are symptoms of the *no free lunch theorem* [KM11] which states that as long as there is utility in the anonymisation result, there is an adversary that can exploit this utility in order to break the anonymity notion. In Chapter 4, Section 2, we will provide an intuitive argumentation for this theorem. Therefore, anonymity notions have not only to be designed with adversaries in mind, but they also have to restrict adversaries. Notions need to formalise for which adversaries they hold.

In the following, we will present new attacks on structural notions, that are not already broadly discussed in literature.

3.2. Subliminal Channel in k-anonymity

The notion k-anonymity allows a malicious adversary to encode information in the result. Consider for example a table with more than one possible k-anonymous anonymisations. For $k = 2$ such a table is depicted in Figure 3.5. For this table, an adversary can choose

name	age	area code	disease
Alice	31	3524	flu
Bob	31	3456	flu
Carol	35	3524	gastritis
X	35	3456	stomach pain

Figure 3.5.: An example for a table that has more than one 2-anonymous tables. The attribute *name* is an identifier and suppressed when generating a k-anonymous database. The attributes *age* and *area code* are the quasi-identifier. These attributes are generalised until the k-anonymity property is fulfilled. The attribute *disease* is the sensitive information and is neither suppressed nor generalised. Figure 3.6 depicts two 2-anonymous transformations of this table.

either to generalise the attribute *age* or the attribute *area code* in order to generate a 2-anonymous result. These results are depicted in Figure 3.6. A malicious anonymiser can, for example choose to disclose the table in Figure 3.6.1, if the X in the original table is *Eve* or to disclose the table in Figure 3.6.2 else. Now, although the result of the anonymisation process is 2-anonymous, an attacker learns if Eve is in the database just by knowing the anonymisation process and looking at which attribute was generalised.

This is because the notion k-anonymity was not defined with respect to adversaries. Cryptographic notions are less prone to this kind of attack because they do restrict what adversaries learn from the information given to them.

age	area code	disease
3*	3524	flu
3*	3456	flu
3*	3524	gastritis
3*	3456	stomach pain

3.6.1: A 2-anonymous version of the table depicted in Figure 3.5 generated by generalising the attribute *age*.

age	area code	disease
31	3*	flu
31	3*	flu
35	3*	gastritis
35	3*	stomach pain

3.6.2: A 2-anonymous version of the table depicted in Figure 3.5 generated by generalising the attribute *area code*.

Figure 3.6.: Tables adhering 2-anonymity derived from the table in Figure 3.5.

3.3. Subliminal Channel in *l*-diversity

The example in Section 3.2 needed the cooperation of the anonymiser and the adversary. The following example, however, will show that structural notions even have subliminal channels if the anonymiser is trustworthy. Consider a database about a city with three city parts A, B, and C that have 7, 9 and 20 inhabitants, respectively (cf. Figure 3.7). The original database consists of a quasi-identifier the city part, and a binary sensitive

city part	# inhabitants
A	7
B	9
C	20

Figure 3.7.: City parts and number of inhabitants of the example for a subliminal channel in *l*-diversity.

information attribute p. Consider for example the table depicted in Figure 3.8. Let the anonymisation process merge city parts (generalise the attribute *city part*) as long as the database does not adhere 2-diversity and try to minimise the number of merges. Now, an adversary can infer information about individuals in the city parts that should be protected by *l*-diversity.

If, for example, all city parts in the disclosed database are merged, an attacker can infer information based on the number of people with p according to Figure 3.9. Consider for example 8 people with p living in the city. If anybody in city part A had p and without loss of generality nobody in C had p the anonymisation process would have stopped after merging A and C. In the other case (people in A, B and C have p) the anonymisation process would have stopped immediately.

This example shows, that knowing an anonymisation process, an attacker can infer information that should be protected by *l*-diversity. As with the example in Section 3.2, this subliminal channel exists because the enforced privacy notion does not explicitly consider adversaries. In contrast, semantic privacy notions like Differential Privacy (Definition 20) or the security notion for database outsourcing Ind-ICP (Chapter 6, Definition 53), are designed with adversaries in mind, and therefore, cover many attacks structural notions are susceptible to. The Bayes Privacy framework we present in Chapter 4 allows for an intuitive definition of such notions.

city part	p?
A	1
A	0
⋮	⋮
B	0
B	1
⋮	⋮
C	1
C	1
⋮	⋮

Figure 3.8.: An example for an original table that shows the distribution of people with the property p among the city parts.

# people with p	information adversary can infer
1	(*nothing to infer*)
2 - 7	all inhabitants with p live in the same city part
8,9	nobody living in part A has p
10 - 15	all inhabitants with p live in part C
16 - 20	all inhabitants with p live in part C or all inhabitants of A and B have p
21	(*impossible*)

Figure 3.9.: Information an adversary can infer depending on the number of people with attribute p in the disclosed database adhering 2-diversity.

4. The Bayes Privacy Framework

This chapter is partially based on work already published in [Hub+11; HM11; Hub10; HMN13] and [HMN14].

1. Introduction

In this chapter, we present the Bayes Privacy framework. It allows for an intuitive definition of semantic privacy notions. These notions are based on what sensitive information an adversary can learn from a release. It is based on the framework presented in our previous work [HMN13]. The ideas behind the Bayes Privacy framework are related to the ideas of other frameworks such as the frameworks presented in [Blu+05] and [KM12]. Privacy is defined with respect to an adversary. For statistical privacy notions, this adversary is an optimal Bayes estimator: The a priori knowledge of the adversary, the knowledge before disclosing the anonymised data, is compared to the a posteriori knowledge of the adversary, the knowledge after disclosing the anonymised data.

A formalisation of a privacy notion requires the definition of the information we want to protect from being disclosed. This can be done by defining sensitive predicates. Predicates are Boolean functions over data sets. For example a sensitive predicate can be the information if an individual in the data set has a specific rare disease or even if a specific individual is present in the database at all.

In [Blu+05] and [KM12], a release must not change the adversaries belief about sensitive information more than a certain threshold. The a posteriori knowledge is bound to the a priori knowledge by a security parameter. The Bayes Privacy framework takes this principle a step further. It introduces privacy breaches based on single sensitive predicates and, additionally, it considers breaches of different magnitudes separately. This allows for the definition of more precise notions, for example notions that allow for small breaches in most cases and breaches of a high magnitude in rare cases. Disclosing the average of an attribute value can be seen as such a mechanism [Dua09]. This mechanism intuitively preserves privacy. It is, however, not considered as privacy preserving by notions and frameworks that only consider the worst case, such as classical Differential Privacy [Dwo08a] or the Pufferfish framework [KM12].

A downside of privacy notions is that, in general, they do not compose. The combination of two releases of mechanisms with the same privacy guarantee can break this guarantee. Therefore results regarding the composability of privacy notions have impact on data privacy. A distinguished example for a privacy notion that provides some form of composability is Differential Privacy. It provides a *graceful degradation* under composition. The combination of two mechanisms that individually provide ϵ-Differential Privacy provides 2ϵ-Differential Privacy [Dwo08a]. In this chapter, we will provide general composability results for notions defined in the Bayes Privacy framework.

In cryptography, computational notions are of particular interest, since they allow for methods that base their security on complexity assumptions. Therefore, we provide a

computational variant of the Bayes Privacy framework and show that our game-based computational privacy notion IND-ICP also can be modelled as a computational Bayes Privacy notion.

Structure of this Chapter In Section 2, we introduce the Bayes Privacy framework. The relation of the Bayes Privacy framework to the Pufferfish framework is discussed in Section 3. This Section also discusses the two privacy axioms introduced by Kiefer et. al. in [KL10], and shows that they do not hold for notions that allow methods to fail to provide privacy in rare cases. In Section 4 we explore compositional properties of Bayes Privacy notions. In Section 5, we will provide examples for notions defined in the Bayes Privacy framework. We will show, how Differential Privacy, the current gold standard in privacy notions, can be defined in the Bayes Privacy framework. Additionally, we show on the example of the average mechanism that the Bayes Privacy framework allows for the definition of privacy notions for mechanisms that fail to provide privacy in rare cases. Finally, Section 6 provides a definition of the Bayes privacy framework with respect to computationally bounded adversaries and prove that IND-ICP can also be defined in the Bayes Privacy framework.

2. Formalisations

The Bayes Privacy framework defines privacy by limiting the knowledge an adversary gains about sensitive information. Given the anonymised release, the adversary tries to guess sensitive predicates. If she can determine the value of a sensitive predicate with the release better than without it, there is a *privacy breach*. In contrast to existing frameworks, the Bayes Privacy framework allows to limit breaches individually.

Figure 4.1 provides an overview over the notation used by the framework and Figure 4.2 provides an overview of the knowledge of the adversary.

notation	description
U	universe of world data sets
$\mathcal{X}$	Set of probability distributions X of U
D	set of all actual data sets or databases
E	mechanism $E : U \to D$, generates a database $d \in D$ from a data set $u \in U$
L	set of sensitive predicates, $l \in L$, $l : U \to \{true, false\}$
F	anonymisation mechanism, $F : D \to D$
s	the release, $s := F \circ E(u)$ for a $u \in U$
$\mathcal{A}$	the adversary

Figure 4.1.: Notation used in the Bayes Privacy framework

The *universe* U is the set of all data sets. We call the data sets in U *world data sets*. Additionally, every data set in U has a probability. The *set* $\mathcal{X}$ *of distributions* of U allows to model possible background knowledge of the adversary. For example, the adversary may know that a data set containing the patients and their diseases of a general hospital in which all patients have the same disease is less likely than a data set with different diseases. This can be modelled as a probability distribution of world data sets.

notation	known to adversary?
U	known
$\mathcal{X}$	known, adversary chooses $X \in \mathcal{X}$
u	unknown
E	known, potential random coins unknown
$E(u)$	unknown
L	known
F	known, potential random coins unknown
$F(d)$	known

Figure 4.2.: Knowledge of the adversary $\mathcal{A}$ about the elements of the framework. The adversary does not know the world data set drawn from the universe. The distribution of the universe, however, is known to the adversary.

Actual data sets or databases are abstractions from the real world. They do not contain all information of the real world. They may even contain inaccuracies owing to the process that generated the database. For example humans may make mistakes while entering data. Some survey techniques like randomized response [War65] themselves even introduce noise to the data set. In the Bayes Privacy framework, this is modelled with a *database generation mechanism* E. It generates the actual data set d from the world data set $u \in U$ and is known to the adversary. If E involves probabilistic processes, however, the randomness is unknown to the adversary. This allows for modelling entropy in the actual database. If the mechanism E itself prevents the adversary from learning a sensitive predicate at all, there is no need to consider this predicate in the anonymisation mechanism F.

The set L contains the *sensitive predicates* that the adversary should be prevented from learning. A sensitive predicate $l \in L$ is defined on all world data sets. For a given world data set a sensitive predicate either evaluates to *true* or to *false*. In order for a privacy notion to be meaningful, a sensitive predicate must not be a constant. Otherwise, the adversary already knows its value and there is no point trying to prevent her from learning it.

The mechanism F is the *anonymisation mechanism*. The idea of Bayes Privacy is that given a release anonymised with F, the adversary $\mathcal{A}$ should not be able to learn the value of any sensitive predicate in L. This can also be interpreted as an experiment with the adversary: The adversary $\mathcal{A}$ chooses a distribution $X \in \mathcal{X}$. According to this distribution, a world data set u is drawn from U and the actual database $d = E(U)$ is generated. Then, the adversary gets the anonymisation $F(d)$ and has to output a sensitive predicate l. Figure 4.3 provides an overview of this process and the interactions of the adversary with it.

Privacy Breach: The Difference of the Adversaries a priori and a posteriori Beliefs If given the universe of data sets U and a distribution X of U, the probability of a sensitive predicate l being true differs from the probability of l being true given U, X, and the release s, the adversary $\mathcal{A}$ potentially learns something about the value of l from the release s. We say, there is a privacy breach. In the following, instead of $\Pr[l = \mathit{true}]$ we write $\Pr[l]$ and instead of $\Pr[F(d) = s]$ we write $\Pr[s]$. Formally, a breach for a sensitive

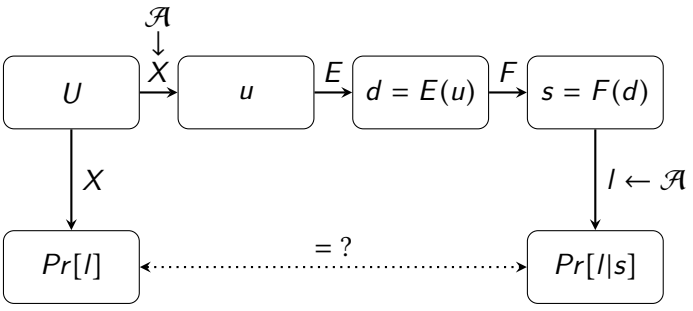


Figure 4.3.: The anonymisation process and the interactions of the adversary. A world data set u is drawn from the universe U of all possible data sets according to a distribution X. The mechanism E generates the actual database from the world data set. The mechanism F anonymises the resulting database. If, for a sensitive predicate I, the probabilities $Pr[I = true]$ and $Pr[I = true|F(d) = s]$ differ, there is a privacy breach.

predicate I and a release s is defined as follows:

Definition 22 (Privacy Breach for a Predicate and a Release)**.** *Let U be a universe of data sets with distribution X, $E : U \to D$ and $F : D \to S$ mechanisms , and L a set of sensitive predicates. A* breach *for the predicate $I \in L$ and the release $s \in S$ is defined as:*

$$b(I, s) := \begin{cases} 1 & Pr[I] \in \{0, 1\} \\ \frac{Pr[I|s]}{Pr[I]} & otherwise \end{cases}$$

The probability is taken over the distribution X and the random coins used in E and F.

Please note that for meaningful notions, there have to be both, data sets for that the sensitive predicate is true as well as data sets for that the sensitive predicate is false (i.e. $0 \leq Pr[I] \leq 1$). Otherwise, there is no point in hiding this predicate. The probability $\Pr[I]$ depends on the distribution of the universe X. If $\Pr[I] \notin \{0, 1\}$ then there are $u, u' \in U$ with $Pr[u] \notin \{0, 1\}$ and $Pr[u'] \notin \{0, 1\}$ and $I(u) \neq I(u')$.

Also note that a breach is only defined on the probability of a sensitive predicate being true. This allows for modelling predicates, that are sensitive only if they are true. For example the information that an individual has AIDS could be considered as sensitive while the opposite could be considered as not sensitive. If a predicate I is sensitive irregardless if it is true or false one can also add $\neg I$ to the set of sensitive predicates L.

In particular, a breach for a sensitive predicate I depends on the release s, the distribution of the universe X, and the mechanisms F and E. For two different distributions of the universe, the same release can cause different breaches for the same sensitive predicate. If the mechanisms E and F are probabilistic, a given world data set can result in different releases. Therefore, different breaches, even for a single sensitive predicate and a single data set, are possible. Consequently, the probability of a breach depends on the distribution X as well as on the random coins used in the data generation and anonymisation mechanisms. Additionally, different breaches can vary in their probability. Consequently, a privacy notion that considers a number of sensitive predicates and allows for probabilistic release mechanisms may limit breaches with different probabilities differently.

It is reasonable to consider a limited set of possible distributions of the universe, since it has been proven to be impossible to provide privacy for arbitrary distributions of the universe without losing all utility of the anonymised data [KM11]. Consider the following argument:

Let an anonymisation mechanism leak 1 bit of information about the original data set. Then, the adversary can classify all possible data sets according to this bit. For every data set in the first class, this bit is 0 while it is 1 for every data set in the second class. Since the adversary may choose the distribution of the universe, she can choose a distribution where all data sets except a data set u_0 in the first class and a data set u_1 in the second class have probability 0 and where u_0 and u_1, each, have probability 0.5. Since the anonymised database now leaks the bit of the class of the original data set, the adversary can determine the original database exactly. Intuitively, this breaks privacy and leads to a high breach if the value of a sensitive predicate for u_0 differs from the value for u_1. Therefore, we have to restrict the adversaries possible background knowledge.

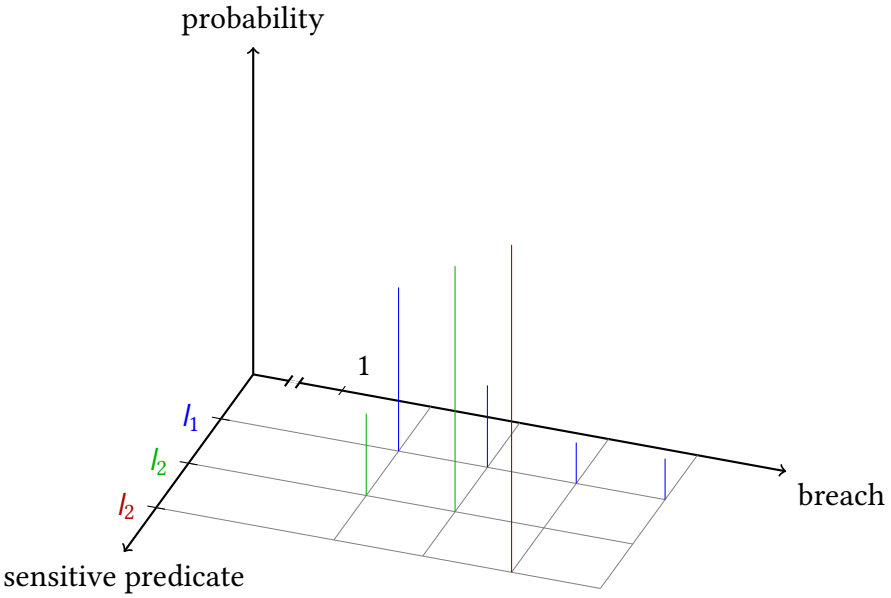


Figure 4.4.: An example for breach distributions of three sensitive predicates $\{l_1, l_2, l_3\}$. The sensitive predicate axis shows for which predicate the breach occurs, the breach axis the magnitude of the breach and the probability axis shows the probability of the breach. Not depicted: breaches of magnitude 1 and lower.

Privacy Notion: An Upper Bound for Breaches Different probabilities of different breaches induce a probability distribution of breaches. Figure 4.4 depicts an example for breach distributions of three sensitive predicates. Since breaches of 1 and lower do not compromise privacy (cf. Definition 22), only breaches of magnitude greater than 1 are depicted. Please note, that the probabilities of the breaches depicted for a single predicate do not necessarily need to add up to 1. The overall probability for breaches greater than 1 can be lower than 1.

In order to reduce the complexity of the resulting privacy notion, for a given release, only the largest breach can be considered. This is reasonable if all sensitive predicates are considered as equally critical. Additionally, if there are sensitive predicates of different

criticality, they can be considered independently by defining separate privacy notions. Therefore, in the following, we only consider the largest breach caused by a given release. Formally, such a breach can be defined as follows:

Definition 23 (Privacy Breach for a Set of Predicates and a Release)**.** *Let U be a universe of data sets with distribution X and let $E : U \to D$ and $F : D \to S$ mechanisms. A* breach *b for a set of sensitive predicates L and a release $s \in S$ is defined as:*

$$b(s) := \max_{l \in L} b(l, s)$$

A release can cause multiple breaches. As mentioned above, this definition only considers the largest breach a release causes. With this definition, we can define the probability of a certain breach. Later, we will use this definition to limit the probability of breaches.

Definition 24 (Breach Probability)**.** *Let U be a universe of data sets with distribution X, $E : U \to D$ and $F : D \to S$ mechanisms, and L a set of sensitive predicates. The probability of a breach b' is called* breach probability of *b' and is defined as:*

$$\Pr[b'] := \Pr_{s \in S}[b' = b(s)]$$

The probability is taken over the distribution X and the random coins used in E and F.

Please note that the breach probability is not the absolute probability of a breach but the probability that this breach is the largest one.

This definition of breach probabilities induces a probability distribution for breaches. Consider the example in Figure 4.5. Here, there are three breaches with a probability greater than 0. Two breaches have a probability of 0.25 and one breach has a probability of 0.5. Note that different sensitive predicates may be responsible for different breaches.

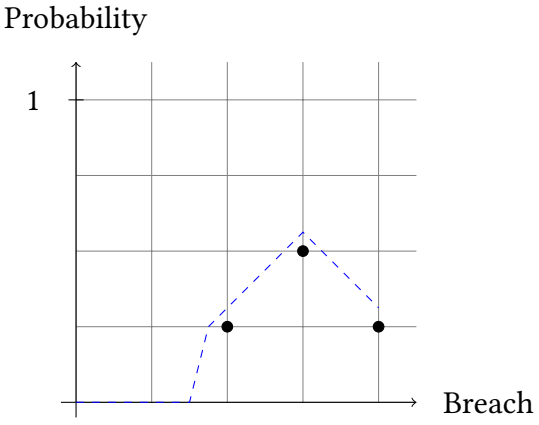


Figure 4.5.: Example for a probability distribution of breaches B (dots) and a dominating function p (dashed line).

Furthermore, an anonymisation mechanism may be well suited for a specific predicate, but not very well for different predicates. Consider for example an anonymisation process that replaces names in a medical disease database with pseudonyms. This process does not hide the number of patients with a specific disease, but it may hide the disease of

a specific patient very well. Thus, a privacy notion should describe how well specific sensitive predicates are hidden from the adversary. Consequently, a privacy notion can be described by the sensitive predicates, the background knowledge of the adversary, and an upper bound of breaches. Formally, this can be defined as follows:

Definition 25 (Bayes Privacy Notion)**.** *For a given universe of possible data sets U, a* privacy notion *is a tuple $(L, p, \mathcal{X})$, where L is a set of sensitive predicates, $p : (1, \infty) \to [0, 1]$ a function, and $\mathcal{X}$ is a set of distributions X. An anonymisation mechanism F fulfils a privacy notion $(L, p, \mathcal{X})$ with respect to a mechanism E if for all distributions $X \in \mathcal{X}$ of the universe U the following holds:*

The function p dominates the breach probability distribution in the interval $(1, \infty)$ for the universe U, the distribution X, and the mechanisms E and F.

If the mechanism E is the identity function, we say that F fulfils a privacy notion $(L, p, \mathcal{X})$ instead of F fulfils a privacy notion $(L, p, \mathcal{X})$ with respect to the mechanism E.

A privacy notion limits the breaches a mechanism may cause. For example, if the dominating function is set constant to 0, any mechanism that fulfils this notion may not output releases that cause breaches greater than 1. Or, more intuitively, such a privacy notion does not allow the adversary to learn anything about a sensitive predicate from a release.

Privacy notions form a half partial order and therefore allow for the comparison of notions. For two privacy notions $A = (L, p, \mathcal{X})$ and $B = (L', p', \mathcal{X}')$ we say that notion B is stronger than A ($A \subseteq B$) iff $L \subseteq L'$, $p' \leq p$, and $\mathcal{X} \subseteq \mathcal{X}'$.

Simplifying Privacy Notions with Cumulative Breach Distributions Considering the cumulative distribution of a breach probability allows for the definition of privacy notions that guarantee that with a specific probability all breaches are equal to or below a certain threshold. This enables the definition of classical privacy notions such as for example (ϵ, δ)-Differential Privacy [Dwo+06c] in the Bayes Privacy framework. Figure 4.6 shows the cumulative distribution of the breach distribution of Figure 4.5. Here, for

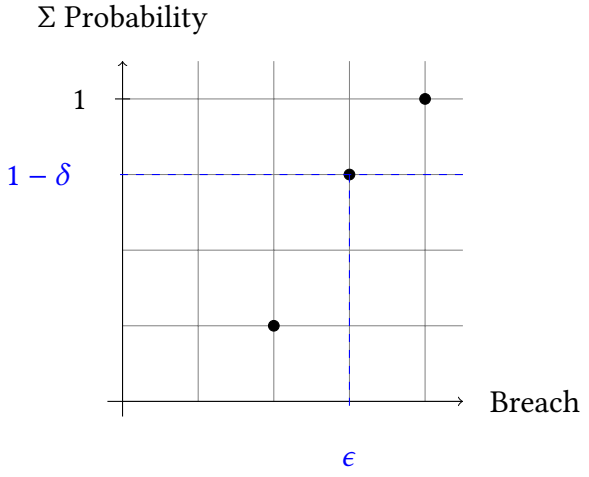


Figure 4.6.: Cumulative distribution of the distribution in Figure 4.5. The probability of a breach equal or less than ϵ is $1 - \delta$.

example, as indicated by the dashed lines the probability of a breach equal or less than

ϵ is $1 - \delta$. Depending on the sensitive predicates considered by the breach distribution, this translates to notions such as (ϵ', δ)-Differential Privacy, while ϵ' is a function of ϵ. Section 5 provides an example of the definition of Differential Privacy in our framework.

Notions that guarantee that with a specific probability all breaches are equal to or below a certain threshold can also be based on privacy notions defined in Definition 25. Then, the guarantee is not based on the cumulative distribution of breaches, but on the integral of the function p that dominates all breaches. This allows for the definition of (ϵ, δ)-privacy notions in the Bayes Privacy framework:

Definition 26 ((ϵ, δ)-Bayes Privacy Notion)**.** *For a universe of possible data sets U, a privacy notion $A = (L, p, \mathcal{X})$ according to Definition 25, a probability δ, and a threshold $\epsilon \geq 1$ a* (ϵ, δ)-privacy notion *is a tuple $(\epsilon, \delta, L, \mathcal{X})$.*

An anonymisation mechanism F fulfils an (ϵ, δ)-privacy notion $(\epsilon, \delta, L, \mathcal{X})$ with respect to a mechanism E if for all distributions $X \in \mathcal{X}$ of the universe U the following holds:

In the interval $[1, \epsilon]$, the threshold $1 - \delta$ dominates the cumulative breach probability distribution for E, F, and L.

This means that the probability of breaches greater than ϵ is equal to or smaller than δ. The advantages of an (ϵ, δ)-privacy notion are twofold. First, the dominating function, which is not necessarily a constant, is replaced with a threshold ϵ. This is more in line with other privacy frameworks. Second, we introduce the threshold δ that specifies the probability for which mechanisms that fulfil the notion are allowed to produce breaches greater than ϵ. Now, in order to check if a mechanism fulfils a (ϵ, δ)-privacy notion, we only need to consider the largest breach that has a probability greater than δ. If the largest breach with a probability $> \delta$ is smaller than ϵ, the mechanism fulfils the (ϵ, δ)-privacy notion.

This definition also allows for the comparison of (ϵ, δ)-privacy notions. The privacy notion $(\epsilon_A, \delta_A, L, \mathcal{X})$ is stronger than the notion $(\epsilon_B, \delta_B, L, \mathcal{X})$ iff $\epsilon_A \leq \epsilon_B$ and $\delta_A \leq \delta_B$. Figure 4.7 shows an example (ϵ_A, δ_A) and (ϵ_B, δ_B). A mechanism that fulfils the notion $(\epsilon_A, \delta_A, L, \mathcal{X})$ also fulfils the notion $(\epsilon_B, \delta_B, L, \mathcal{X})$. A mechanism that fulfils notion $(\epsilon_B, \delta_B, L, \mathcal{X})$ does not necessarily fulfil the notion $(\epsilon_A, \delta_A, L, \mathcal{X})$, since it can have breaches of ϵ_B with any probability and breaches bigger than ϵ_B with probability δ_B.

3. The Bayes Privacy Framework and other Privacy Frameworks

In the last section, we introduced privacy notions that are defined by limiting the probabilities of breaches. Furthermore, we introduced notions that allow for methods that fail providing privacy with a small probability, so-called (ϵ, δ)-notions, into the Bayes Privacy framework. In [KL10; KL12] Kifer and Lin postulated privacy axioms that all privacy notions should fulfil. At first sight, these axioms seem reasonable as they allow for a mathematical examination of privacy notions. Unfortunately, these axioms do not hold for Bayes Privacy notions, as we will show in this section.

Since the Pufferfish framework [KM12] follows these axioms [KM14], (ϵ, δ)-notions can not be modelled in the Pufferfish framework. In this section, we will provide meaningfully examples of Bayes Privacy notions that do not fulfil the privacy axioms. Furthermore, we show that for every Pufferfish notion there is Bayes Privacy notion that is fulfilled

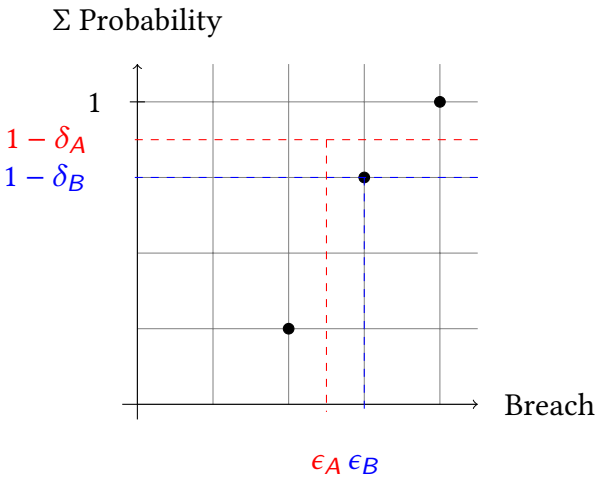


Figure 4.7.: Cumulative distribution of the distribution in Figure 4.5 and thresholds of two (ϵ, δ)-privacy notions. The probability of a breach less than or equal ϵ_B is $1 - \delta_B$. However, the probability of a breach less than or equal to ϵ_A is greater than $1 - \delta_A$.

by exactly the same mechanisms, suggesting that the Bayes Privacy framework is more general than the Pufferfish framework and similar frameworks.

3.1. Bayes Privacy and the Privacy Axoims of Kifer and Lin

In order to achieve a consistent basis that allows for a more precise mathematical examination of privacy notions, Kifer and Lin [KL10; KL12] presented two privacy axioms. These axioms are deemed to be essential for every privacy notion and are known to hold for Differential Privacy [Dwo08a] and notions of similar structure, for example notions defined in the Pufferfish framework.

The first axiom states that any computation (without additional input other than the release and randomness) on the anonymised data should not compromise the privacy. The privacy definition must still hold after the computation, i. e. the combined mechanisms satisfy the same definition.

Axiom 1. *(Transformation Invariance) [KL10]*
Let $\mathcal{M}$ be a privacy mechanism for a particular privacy definition and let $\mathcal{A}$ be a randomised algorithm whose input space contains the output space of $\mathcal{M}$ and whose randomness is independent of both the data and the randomness in $\mathcal{M}$. Then $\mathcal{M}' \equiv \mathcal{A} \circ \mathcal{M}$ must also be a privacy mechanism satisfying the privacy definition.

The idea of this axiom is that given anonymised data, an adversary simply can execute computations on it and change the result, which should not compromise the privacy. This idea seems reasonable and supports the design of privacy preserving mechanisms on top of an existing one and enables simple reduction proofs for mechanisms. The first example in this section, however, shows, that Bayes Privacy notions do not necessarily fulfil this axiom. Consequently, one can argue, that this axiom is too strict.

The second axiom is the convexity axiom. It states that given two mechanisms fulfilling the same privacy notion, every mechanism that just randomly selects one of these two mechanisms also fulfils this privacy notion.

Axiom 2. *(Convexity) [KL10]*
Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be privacy mechanisms that satisfy a particular privacy definition (and such that the randomness in $\mathcal{M}_1$ is independent of the randomness in $\mathcal{M}_2$). For any $p \in \{0, 1\}$, let $\mathcal{M}_p$ be a randomised algorithm that on input i outputs $\mathcal{M}_1(i)$ with probability p (independent of the data and the randomness in $\mathcal{M}_1$ and $\mathcal{M}_2$) and outputs $\mathcal{M}_2(i)$ with probability $1 - p$. Then $\mathcal{M}_p$ is a privacy mechanism that satisfies the privacy definition.

The intuition behind the convexity axiom is that if two mechanisms fulfil the same privacy notion it should not matter which one is used to anonymise data. Therefore they can be used unchangeably. More precisely, a mechanism that selects randomly one of the two mechanisms that individually already fulfil a privacy notion should also fulfil this privacy notion. The second example will show that this also does not hold for (ϵ, δ)-Bayes Privacy notions.

Example 1 In the first example, the post processing mechanism changes the probabilities and magnitude of breaches in such a way, that the resulting mechanism does not fulfil the original notion. The original notion can either be modelled as a (ϵ, δ)-notion that allows mechanisms to fail to provide privacy with probability $\delta = 0.5$, or it can be modelled as a Bayes Privacy notion with a dominating function $p = 0.5$. In both cases, after post processing, the original privacy notion will be violated.

Suppose an intelligence service has a release mechanism that answers a binary question, for example about a secret operation, to a politician truthfully with probability p while answering "no comment" the rest of the time. The intelligence service might find this mechanism acceptable, because there is only a breach with probability p. With probability $1 - p$, the politician does not learn anything about their secret operations.

Further assume the politician has to relay the information to the press, but does not want to give fishy answers. So instead of the "no comment" the politician answers the question randomly. In our framework, this example can be modelled with a universe that contains two data sets. In the first one, the answer to the binary question is 0; in the second one it is 1:

$$U = \{u_0 = \{t = 0\}, u_1 = \{t = 1\}\}$$

For the sake of simplicity, we set the probability of each data set to 0.5:

$$\mathcal{X} = \{X = \{\Pr[u_0] = 0.5, \Pr[u_1] = 0.5\}\}$$

The sensitive predicates are the answers to the binary question t. The first predicate defines the answer 0 as sensitive, the second predicate defines the answer 1 as sensitive:

$$L = \{t = 0, t = 1\}$$

The mechanism E simply extracts the answer to b from the data sets. Therefore, we define $E(u) = t$, or more precisely $E(u_0) = 0$, $E(u_1) = 1$. The mechanism F is the mechanism of the intelligence service. It tells the truth with probability 0.5 and answers $\perp$ with the remaining probability of 0.5. Since the politician does not want to relay fishy answers the the press, he randomly chooses a possible answer if the answer he gets from the intelligence service is $\perp$. Therefore, the post processing mechanism G returns randomly 0 or 1 if the input is $\perp$. Formally, the mechanisms F and G are defined as follows:

$$F(d) = \begin{cases} d & \text{probability } p \\ \perp & \text{probability } 1-p \end{cases} \qquad G(d') = \begin{cases} d' & d' \in \{0,1\} \\ v, \text{ with } v \leftarrow \{0,1\} & d' = \perp \end{cases}$$

If the politician gets an answer other than $\perp$ it is the truth. Therefore, the probability that the sensitive predicate has the same value than the release is 1:

$$\Pr[t = 0 \mid F(d) = 0] = \Pr[t = 1 \mid F(d) = 1] = 1$$

In the following, for our calculations and figures, we set $p = 0.5$. If the mechanism F produces a breach (cf. Definition 22) of a sensitive predicate (either $t = 0$ or $t = 1$), the magnitude of the breach is 2:

$$b(I, F(d)) = \frac{\Pr[t = 0 \mid F(d) = 0]}{\Pr[t = 0]} = \frac{\Pr[t = 1 \mid F(d) = 1]}{\Pr[t = 1]} = \frac{1}{0.5} = 2$$

Since the probability for $F(d) = d$ is $p = 0.5$, the probability of a breach of 2 is 0.5. Figure 4.8 shows the breach distribution for the mechanism F, as well as an example for a dominating function.

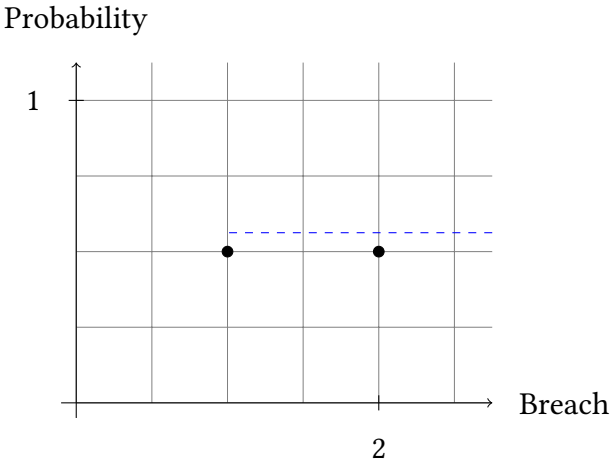


Figure 4.8.: Breach distribution for the mechanism F. Note, that since breaches equal or less than 1 are not considered to compromise privacy, the domain of the dominating function is $(1, \infty)$ (cf. Definitions 22 and 25). The probability for no privacy breach at all is $1 - p = 0.5$.

The second mechanism G changes these probabilities. Since the politician guesses the answer if he gets no answer ($\perp$) from the intelligence service, it holds true that the probability that the release after post processing is equal to the value of t is 0.75:

$$\Pr[t = 0 \mid G(d') = 0] = \Pr[t = 1 \mid G(d') = 1] = 0.75$$

Now, however, the probability for this breach is 1 and the magnitude of the breach is smaller with $b(I, G(F(d))) = 1.5$:

$$b(I, G(F(d))) = \frac{\Pr[t = 0 \mid, G(d') = 0]}{\Pr[t = 0]} = \frac{0.75}{0.5} = 1.5$$

Obviously, after application of G, the security definition of the first algorithm is not met anymore (cf. Figure 4.9). While our intelligence service might have accepted the mechanism F, they might reject the mechanism $G \circ F$, since with $G \circ F$ the public always learns something about their secret operations.

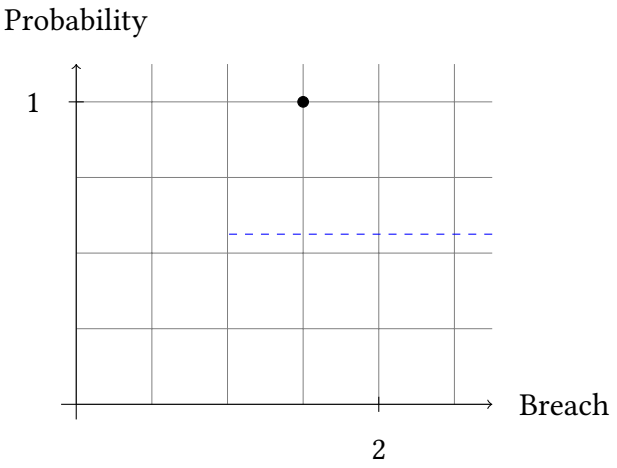


Figure 4.9.: Breach distribution for the Mechanism $G \circ F$ and the dominating function for the mechanism F from Figure 4.8. This function does not dominate the mechanism $G \circ F$. Furthermore, there is always a privacy breach.

Example 2 For the second example consider an examination about corruption in politics. The goal is to find the truth about whether a certain politician was bribed or not. A proof that the politician was bribed would mean the end of his career. Consider the same universe U, distribution $\mathcal{X}$, mechanism E as in the first example. Now, $t = 1$ means that the politician was bribed. Therefore, the sensitive predicate (from the view of the politician) is $L = \{t = 1\}$.

Consider the following two mechanisms: Mechanism F_1 hides the predicate l_1 with a high probability v, for example say $v = 0.99$:

$$F_1(d) = \begin{cases} d & \text{probability } 0.01 \\ \perp & \text{probability } 0.99 \end{cases}$$

Here, $\perp$ means, that the examination could not find any evidence. The second mechanism F_2 hides the predicate (perfectly) is a mechanism F_2. It simply returns a random bit:

$$F_2(d) = b, \text{ with } b \leftarrow \{0, 1\}$$

Separately, both mechanisms hide the sensitive predicate perfectly ($\epsilon = 0$) with at least probability 0.99. Formally, this means, that the mechanisms F_1 and F_2 fulfil the notion $A = (L, 0.01, \mathcal{X})$. This notion can also be defined as an (ϵ, δ)-notion that allows mechanisms to fail to provide privacy with probability 0.01 for L, $\mathcal{X}$, U, and E as defined above.

Both mechanisms individually might be acceptable by the politician, since the probability that he can continue his career unharmed is very high.

Now, consider a meta mechanism M that combines F_1 and F_2. The mechanism M randomly chooses which of the mechanisms F_1 and F_2 to use:

$$M(d) = \begin{cases} F_1(d) & \text{probability p} \\ F_2(d) & \text{probability 1-p} \end{cases}$$

Since the random coins used in anonymisation mechanisms are secret, if the output of M is either 0 or 1, one of both mechanisms of F_1 and F_2 could have been selected. Therefore,

in this combination, suddenly, the random output of F_2 gets correlated to the real value of d.

This decreases the probability δ' that the predicate is perfectly hidden to $p \cdot (1 - v) + (1 - p) \cdot 0.5$, which is smaller than 0.99 for any non trivial probability p. This means that the combination of the two mechanisms according to Axiom 2 does not fulfil the notion fulfilled by the individual mechanisms irregardless of the probability (except for $p = 0$ and $p = 1$) used in the combination. The breach generated by $M(d) = 1$ is significantly lower than the breach generated by $F_1(d) = 1$: However, the probability that $M(d) = 1$ is higher than the probability that $F_1(d) = 1$ (for $0 < p < 1$):

$$\Pr[M(d) = 1] = p \cdot (0.01) + (1 - p) \cdot 0.5 \geq 0.01 = \Pr[F_1(d) = 1]$$

For our politician, this means, that the probability that he can redeem himself without a doubt is lower for the mechanism M. This might not be acceptable for him, even if he deems the individual mechanisms to be acceptable.

To summarise the results, there are meaningful privacy guarantees that can be modelled in the Bayes Privacy framework but do not fulfil the privacy axioms mentioned above.

3.2. Bayes Privacy and Pufferfish

The last sections showed that the Bayes Privacy framework allows for modelling notions that can not be modelled in the Pufferfish framework or any framework for which Axioms 1 and 2 hold. In this section, we show that for every notion in the Pufferfish framework, there is a corresponding notion in the Bayes Privacy framework that is fulfilled by exactly the mechanisms that also fulfil the Pufferfish notion.

For all pairs of sensitive predicates, the Pufferfish framework requires that the probability of the output is within a certain range (determined by the security parameter ϵ) given one of the sensitive predicates is true. This translates to the requirement that all breaches are below a certain threshold. Consequently, the dominating function p (the security parameter in our privacy framework, cf. Definition 25) is a constant. The Bayes Privacy framework does not restrict p to be constant. Here, p can be a arbitrary function. This allows for potentially more precise privacy guarantees.

Lemma 2. *Let $P = (\mathbb{S}, \mathbb{S}_{pairs}, \mathbb{D})$ be a Pufferfish notion. Let the set $\mathbb{M}$ be the set of all mechanisms that fulfil P. Then there is a Bayes Privacy notion P' that is fulfilled by exactly the mechanisms in $\mathbb{M}$.*

The following proof shows how to define P' given P. The basic idea of the proof is that given a set of mechanisms, there is a dominating function for their breaches. The proof shows the relation of a notion in Pufferfish and the privacy breaches that depend on the possible output of each mechanism and the on the set of sensitive predicates $\mathbb{S}$. This allows for defining a Bayes Privacy notion that is fulfilled by all mechanisms in $\mathbb{M}$.

Proof. Let $P = (\mathbb{S}, \mathbb{S}_{\text{pairs}}, \mathbb{D})$ be a privacy notion in the Pufferfish framework. Then, there is an upper bound $b(I, s)$ for each release s and each sensitive predicate $I \in \mathbb{S}$ that depends on P:

For a given distribution $\theta \in \mathbb{S}$ of databases, Pufferfish requires for all pairs of sensitive predicates $(I_1, I_2) \in \mathbb{S}_{\text{pairs}}$ and for all possible outputs s of the anonymisation function

(cf. Definition 21) that the probability of s given I_1 is bound to the probability of s given I_2:

$$\Pr[s|I_1] \leq e^{\epsilon} \Pr[s|I_2]$$

and

$$\Pr[s|I_2] \leq e^{\epsilon} \Pr[s|I_1]$$

The predicates of a pair $(I_1, I_2) \in \mathbb{S}_{\text{pairs}}$ are mutually exclusive, but not necessarily exhaustive. This means that both predicates can not be true. They can, however, both be false:

$$\Pr[I_1, I_2] = 0$$

$$\Pr[\neg I_1, \neg I_2] \geq 0$$

Therefore, and with the law of total probability (Lemma 1), we get for $(I_1, I_2) \in \mathbb{S}_{\text{pairs}}$:

$$\begin{aligned} & \Pr[s] = \Pr[s|I_1]\Pr[I_1] + \Pr[s|I_2]\Pr[I_2] \\ & \qquad + \Pr[s|\neg I_1, \neg I_2]\Pr[\neg I_1, \neg I_2] \\ \Leftrightarrow \quad & \Pr[s|I_2] = \frac{1}{\Pr[I_2]}(\Pr[s] - \Pr[s|I_1]\Pr[I_1] \\ & \qquad - \Pr[s|\neg I_1, \neg I_2]\Pr[\neg I_1, \neg I_2]) \end{aligned}$$

With this formula, we get:

$$\begin{aligned} & \Pr[s|I_1] \leq e^{\epsilon} \Pr[s|I_2] \\ \Leftrightarrow \quad & \Pr[s|I_1] \leq e^{\epsilon} \frac{1}{\Pr[I_2]}(\Pr[s] - \Pr[s|I_1]\Pr[I_1] \\ & \qquad - \Pr[s|\neg I_1, \neg I_2]\Pr[\neg I_1, \neg I_2]) \\ \Leftrightarrow \quad & \Pr[s|I_1](1 + e^{\epsilon}\Pr[I_1]) \leq e^{\epsilon} \frac{1}{\Pr[I_2]}(\Pr[s] - \Pr[s|\neg I_1, \neg I_2]\Pr[\neg I_1, \neg I_2]) \end{aligned}$$

According to Bayes' Theorem (Lemma 1), the equation above is equal to:

$$\begin{aligned} \Leftrightarrow \quad & \frac{\Pr[I_1|s]\Pr[s]}{\Pr[I_1]}(1 + e^{\epsilon}\Pr[I_1]) \leq e^{\epsilon} \frac{1}{\Pr[I_2]}(\Pr[s] - \Pr[\neg I_1, \neg I_2|s]\Pr[s]) \\ \Leftrightarrow \quad & \frac{\Pr[I_1|s]}{\Pr[I_1]}(1 + e^{\epsilon}\Pr[I_1]) \leq e^{\epsilon} \frac{1}{\Pr[I_2]}(1 - \Pr[\neg I_1, \neg I_2|s]) \\ \Leftrightarrow \quad & \frac{\Pr[I_1|s]}{\Pr[I_1]} \leq \frac{e^{\epsilon}}{1 + e^{\epsilon}\Pr[I_1]} \frac{1}{\Pr[I_2]}(1 - \Pr[\neg I_1, \neg I_2|s]) \\ \Leftrightarrow \quad & b(I_1, s) \leq \frac{e^{\epsilon}}{1 + e^{\epsilon}\Pr[I_1]} \frac{1}{\Pr[I_2]}(1 - \Pr[\neg I_1, \neg I_2|s]) \end{aligned}$$

Since it is possible that I_1 is a sensitive predicate in more than one pairs in $\mathbb{S}_{\text{pairs}}$, the upper bound for a breach is determined by the pair for which $b(I_1, s)$ has the largest value. With the existence of an upper bound for each breach $b(s, I)$ that depends on the Pufferfish notion P, there also exists a dominating function p that is implied by P:

$$p = \max_{I_1, I_2 \in \mathbb{S}_{\text{pairs}}, s} \frac{e^{\epsilon}}{1 + e^{\epsilon}\Pr[I_1]} \frac{1}{\Pr[I_2]}(1 - \Pr[\neg I_1, \neg I_2|s]) \tag{4.1}$$

With this domination function, all mechanisms that fulfil the Pufferfish notion $P = (\mathbb{S}, \mathbb{S}_{\text{pairs}}, \mathbb{D})$ also fulfil the Bayes Privacy notion $P' = (\mathbb{S}, p, \mathbb{D})$.

Moreover, let $M' \notin \mathbb{M}$ be a mechanism that does not fulfil the Pufferfish notion P. Then, there is an $(l_1, l_2) \in \mathbb{S}_{\text{pairs}}$ for which:

$$\Pr[s|l_1] > e^{\epsilon} \Pr[s|l_2]$$

Consequently, analogous to the equations above follows, that:

$$b(l_1, s) > \frac{e^{\epsilon}}{1 + e^{\epsilon} \Pr[l_1]} \frac{1}{\Pr[l_2]} (1 - \Pr[\neg l_1, \neg l_2|s])$$

Then, the mechanism M' can cause a breach that is not dominated by the function in equation 4.1. Thus, M' does not fulfil the Bayes Privacy notion P'. □

This result seems to be in contradiction to the examples in this section that showed that Bayes Privacy notions do not necessarily fulfil the privacy axioms that are fulfilled by Pufferfish notions. However, the Bayes Privacy framework allows to express notions that can not be expressed in the Pufferfish framework. For example, the Pufferfish framework demands for every release s and all all pairs of sensitive predicates $(l_1, l_2) \in \mathbb{S}_{\text{pairs}}$ that $\Pr[s|l_1] \leq e^{\epsilon} \Pr[s|l_2]$. This means that if a certain release is possible given l_1, it has also be possible given l_2. The Bayes Privacy framework is not that restrictive and therefore allows to express notions that can not be expressed in the Pufferfish framework.

4. Composition and Decomposition of Bayes Privacy Notions

An important property of security notions in general is composition. Security mechanisms are always used in a context that can influence the security properties of the mechanism. If a security notion composes universally, mechanisms retain this property independently of the context they are used in. Therefore, universal composability is desirable for security notions.

There are cryptographic frameworks with a focus on composability. The most established framework is the *Universal Composability (UC) Framework* [Can01]. This framework follows the ideal-world-real-world paradigm in oder to prove the security properties of a mechanism and its compositional properties. The UC-framework, however, can not be used directly to study the composition properties of privacy notions since there can always be constructed a context in which a mechanism that may fulfil a privacy notion in different contexts, does fails to provide privacy [KM11] (cf. Section 2).

In the Bayes Privacy framework, the possible background knowledge of the adversary can also be interpreted as allowed contexts a mechanism can be used in. Therefore, the Bayes Privacy framework itself allows to study the compositional properties of privacy notions. Intuitively, one would assume that Bayes Privacy notions self-compose linearly. A breach changes the adversaries belief about a sensitive predicate. A second breach for the same sensitive predicate should change the adversaries belief a second time. Consequently, the overall breach for that predicate should not be larger than the sum of the individual breaches. In general, however, this is not the case. Consider the following example:

Let our universe U be comprised of four databases $d_0, \dots d_3$. Let each database be equally probable (i. e. the set of background knowledge $\mathcal{X}$ of adversaries only contains the uniform distribution.). Let the set of sensitive predicates L contain only one sensitive predicate I with:

$$I(d_0) = I(d_1) = \mathit{false}$$
$$I(d_2) = I(d_3) = \mathit{true}$$

Let F_0 and F_1 be anonymisation mechanisms according to Figure 4.10. Individually, the release of the result of an anonymisation mechanism F_0 or F_1 does not change the a posteriori probability of I. Therefore, F_0 and F_1, individually fulfil the Bayes Privacy notion $(L, 0, \mathcal{X})$.

	Original Database			
Mechanism	d_0	d_1	d_2	d_3
F_0	1	0	1	0
F_1	a	b	b	a

Figure 4.10.: Output of the Mechanisms F_1 and F_2 as a function of the input.

If an adversary, however, learns both, a release of F_0 and a release of F_1, she learns the original database and, therefore, the exact value of I. For example, if the release generated by F_0 is 1 and the release generated by F_1 is b, the original database was d_2 an the value of I is *true* (cf. Figure 4.11).

	$F_1(d_x)$	
$F_2(d_x)$	0	1
a	d_3	d_0
b	d_1	d_2

Figure 4.11.: With the releases generated by F_1 and F_2, the adversary learns the original database and therefore the exact value of the sensitive predicate I.

This simple example shows that, in general, Bayes Privacy notions do not compose. Two releases of mechanisms, that individually fulfil the same Bayes Privacy notion can completely break privacy. In the following, we will study when and how Bayes Privacy notions compose. Therefore, we introduce the notion of a posteriori beliefs an adversary can have after a release from a mechanism that fulfils a privacy notion P. Consider Figure 4.12. Prior to the release, the adversary has the knowledge X. The release s changes the assumptions of the adversary about the original data set. There may be data sets that can now be excluded, because they can not lead to the release s and there may be data sets that are now more probable, because they lead to the release s with a high probability. Therefore, after the release, the adversary has the belief X'.

If the mechanism that produced the release fulfils a privacy notion and the adversary with background knowledge X is considered by this notion, we call the belief X' an a posteriori belief of P:

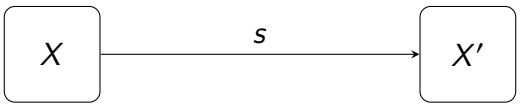


Figure 4.12.: The a priori knowledge X of an adversary is transformed to the a posteriori knowledge X' after the release of s. The distributions X and X' are assumptions about the probabilities of the data sets in the universe.

Definition 27 (A Posteriori Belief of a Bayes Privacy Notion)**.** *Let $P = (L, p, \mathcal{X})$ be a Bayes Privacy notion. An* a posteriori belief *of P is a belief of an adversary $\mathcal{A}$ with an a priori background knowledge $X \in \mathcal{X}$ after learning a release from a mechanism that fulfils P.*

Consequently, a Bayes Privacy notion has a set of a priori beliefs and a set of a posteriori beliefs. Consider Figure 4.13. The set of a posteriori beliefs for a notion is the result of conditioning each a priori belief with each release that can be produced by a mechanism that fulfils this notion. Note, since a mechanism that does not change any belief, for

Figure 4.13.: All possible a priori knowledge $\mathcal{X}$, and all possible a posteriori knowledge $\mathcal{X}'$ allowed by a notion after a release by a mechanisms that fulfils this notion. Note, that $\mathcal{X} \subseteq \mathcal{X}'$

example a mechanism that releases just random numbers, does fulfil any notion, the a priori beliefs of a Bayes Privacy notion are a subset of the a posteriori beliefs. Also note, that for a given a priori belief, not all a posteriori beliefs are equally possible. If and with which probability an a posteriori belief can be reached from a given a priori belief is determined by the dominating function of the Bayes Privacy notion that restricts the breach probabilities.

With this view of Bayes Privacy notions, we can (de-)compose notions. Consider for example the notion from Figure 4.13 with the a priori beliefs $\mathcal{X}$ and the a posteriori beliefs $\mathcal{X}'$. Let us assume that there are sets of distributions, that lie *in between* $\mathcal{X}$ and $\mathcal{X}'$. This allows us to define notions that compose to the original notion. Consider Figure 4.14. The Bayes Privacy notion has been decomposed into 2 notions. In the following, we

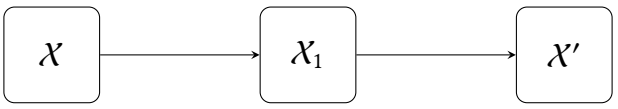


Figure 4.14.: Decomposition of the notion from Figure 4.13 into two notions.

will examine under which circumstances the composition of two releases fulfils a Bayes Privacy notion, or more precisely, how two notions can be combined into a composite notion. Therefore, we need to define, when we can combine notions:

Definition 28 (Compatibility of Bayes Privacy Notions)**.** *For a Universe U, let $P_1 = (L, p_1, \mathcal{X}_1)$ and $P_2 = (L, p_2, \mathcal{X}_2)$ be Bayes Privacy-notions. Let $\mathcal{X}_1'$ be the set of all a posteriori beliefs of P_1. We say that P_2* is compatible to *P_1 iff $\mathcal{X}_1' = \mathcal{X}_2$.*

If two Bayes Privacy notions P_1 and P_2 are compatible, the dominating function of the notion composed of P_1 and P_2 depends on the dominating functions of the individual notions. A dominating function limits the probability of a breach and a breach changes the adversaries belief. Consequently, the probability of a given a posteriori belief is determined by the dominating function.

We can, for two compatible Bayes Privacy notions with special dominating functions, specify the compositional notion easily. In the following, we will do this for notions with constant dominating functions and for notions with dominating functions, that allow breaches up to a certain threshold and disallow breaches bigger than this threshold.

Proposition 1. *For a universe U, let $P_1 = (L, p_1, \mathcal{X}_1)$ and $P_2 = (L, p_2, \mathcal{X}_2)$ be Bayes Privacy notions with $p_1(b) = pr_1$ and $p_2(b) = pr_2$ for all $b \in (1, \infty)$ with $pr_1, pr_2 \in [0, 1]$ Furthermore, let P_2 be compatible to P_1 and let $M = (M_1, M_2)$ with $M(d) = (M_1(d), M_2(d))$ be a mechanism with M_1 fulfilling P_1 and M_2 fulfilling P_2. Then, the following holds:*

The mechanism M fulfils the Bayes Privacy notion $P = (L, p, \mathcal{X})$ with $p(b) = \max\{pr_1, pr_2\}$.

In the following proof, we calculate the breach for a sensitive predicate and a release of the composite mechanism M.

Proof. Let P, P_1, P_2, and M be defined as in Proposition 1. Then, for all breaches $b_1, b_2 > 1$, data sets $d \in U$, sensitive predicates $l \in L$ and releases $M(d) = (s_1, s_2)$ holds:

$$\Pr[b(l, s_1) = b_1] \leq pr_1$$

with respect to an adversary with background knowledge $X \in \mathcal{X}$, and

$$\Pr[b(l, s_2) = b_2] \leq pr_2$$

with respect to an adversary with background knowledge $X' \in \mathcal{X}'$. For an adversary with background knowledge $X \in \mathcal{X}$, a breach b for a sensitive predicate $l \in L$ and a release $s = M(d)$ is defined as:

$$b(l, s) = \frac{\Pr[l|s]}{\Pr[l]}$$

In the following, we explicitly state the background knowledge of the adversary in the formulas:

$$b(l, s) = \frac{\Pr[l|s]}{\Pr[l]} = \frac{\Pr[l|s, X]}{\Pr[l|X]} = \frac{\Pr[l|s_1, s_2, X]}{\Pr[l|X]}$$

Let X' be the a posteriori belief of the adversary after the release of s_1. Then we get:

$$(l, s) = \frac{\Pr[l|s_1, s_2, X]}{\Pr[l|X]} = \frac{\Pr[l|s_2, X']}{\Pr[l|X]}$$
$$b(l, s_1) = \frac{\Pr[l|s_1, X]}{\Pr[l|X]} = \frac{\Pr[l|X']}{\Pr[l|X]}$$

Combining these two formulas, we get:

$$b(I,s) = \frac{\Pr[I|s_2, X']}{\Pr[I|X']} \cdot b(I,s_1)$$
$$b(I,s) \leq b(I,s_2) \cdot b(I,s_1) \tag{4.2}$$

Since in P_1 all breaches have probability pr_1 and in P_2, all breaches have probability pr_2, the probability for a breach $b > 1$ in P is:

$$\Pr[b(I,s) > 1] \leq \Pr[b(I,s_2) \cdot b(I,s_1) > 1] \leq \max\{pr_1, pr_2\}$$

Which is the behaviour as defined by the dominating function p in Proposition 1. □

Intuitively, this is the expected result. Since we did not restrict the magnitude of breaches but only their probabilities, the probability that a breach occurs depends on the probabilities of breaches occurring in the individual notions. If for example M_1 causes a breach, it does not matter if M_2 causes a breach and vice versa. Therefore, the probability for a breach only depends on the maximal probability for a breach in the individual notions.

In the following, we study how notions compose, if we restrict the magnitude of breaches. Therefore, we define two compatible notions, that allow breaches up to certain thresholds:

Proposition 2. *For a universe U, let $P_1 = (L, p_1, \mathcal{X}_1)$ and $P_2 = (L, p_2, \mathcal{X}_2)$ be Bayes Privacy notions with*

$$p_1(b) = \begin{cases} 1, & b \in (1, b_1] \\ 0, & b \in (b_1, \infty) \end{cases}$$

and

$$p_2(b) = \begin{cases} 1, & b \in (1, b_2] \\ 0, & b \in (b_2, \infty) \end{cases}$$

Furthermore, let P_2 be compatible to P_1 and let $M = (M_1, M_2)$ with $M(d) = (M_1(d), M_2(d))$ be a mechanism with M_1 fulfilling P_1 and M_2 fulfilling P_2. The mechanism M fulfils the Bayes Privacy-notion $P = (L, p, \mathcal{X})$ with

$$p(b) = \begin{cases} 1, & b \in (1, b_1 \cdot b_2] \\ 0, & b \in (b_1 \cdot b_2, \infty) \end{cases}$$

for all breaches $b \geq 1$.

Proof. In the proof for Proposition 1, we got the following equation (Equation 4.2):

$$b(I,s) \leq b(I,s_2) \cdot b(I,s_1)$$

From this equation follows that the biggest breach of P is equal to or smaller than the product of the individually biggest breaches in P_1 and P_2. The biggest breach allowed in P_1 is b_1; the biggest breach allowed in P_2 is b_2. Therefore, the biggest breach that can be caused by a release of M is $b_1 \cdot b_2$. Breaches greater than $b_1 \cdot b_2$ can not be caused by M, which proves Proposition 2. □

The proofs for Propositions 1 and 2 suggest, that two compatible notions that individually restrict the breach probability of possible breaches and only allow breaches up to a certain threshold compose according to Proposition 1 and Proposition 2. This holds true:

Proposition 3. *For a universe U, let $P_1 = (L, p_1, \mathcal{X}_1)$ and $P_2 = (L, p_2, \mathcal{X}_2)$ be Bayes Privacy notions with*

$$p_1(b) = \begin{cases} pr_1, & b \in (1, b_1] \\ 0, & b \in (b_1, \infty) \end{cases}$$

and

$$p_2(b) = \begin{cases} pr_2, & b \in (1, b_2] \\ 0, & b \in (b_2, \infty) \end{cases}$$

with $z_1, z_2 \in (1, \infty)$ and $pr_1, pr_2 \in [0, 1]$. Furthermore, let P_2 be compatible to P_1 and let $M = (M_1, M_2)$ with $M(d) = (M_1(d), M_2(d))$ be a mechanism with M_1 fulfilling P_1 and M_2 fulfilling P_2. The mechanism M fulfils the Bayes Privacy-notion $P = (L, p, \mathcal{X})$ with

$$p(b) = \begin{cases} \max\{pr_1, pr_2\}, & b \in (1, b_1 \cdot b_2] \\ 0, & b \in (b_1 \cdot b_2, \infty) \end{cases}$$

for all breaches $b \geq 1$.

Proof. From Proposition 2 follows that the largest breach possible is $b_1 \cdot b_2$. From Proposition 1 follows that the probability for a breach is equal to or lower than $\max\{pr_1, pr_2\}$. □

Propositions 1, 2, and 3 show that the Bayes Privacy framework allows to study the composition properties of privacy notions. Privacy notions with more complex dominating functions require an individual, more profound analysis of their compositional properties. In practice, one can substitute such complex notions with notions with simple dominating functions such as in the Propositions 1, 2, and 3 in order to get an upper bound for the dominating function of the composed notion. In Section 5.1 we will study the composition properties of Differential Privacy and we will show that Differential Privacy composes according to Proposition 2.

5. Examples

In this section we show how the Bayes Privacy framework can be applied. Since Differential Privacy is the gold standard in privacy notions, in Section 5.1, we show how to express Differential Privacy as notion in the Bayes Privacy framework. Furthermore, we show the compositional properties of Differential Privacy in the Bayes Privacy framework. In Section 5.2 we provide an example that shows how Bayes Privacy allows for modelling a notion for an averaging mechanism. This is particularly interesting because intuitively disclosing the average value of an attribute provides privacy. Furthermore, the averaging mechanism is extensively used in practice, for example during election periods.

5.1. Differential Privacy

As the Pufferfish framework, the Bayes Privacy framework allows to model Differential Privacy, even (ϵ, δ)-Differential Privacy. In order to model Differential Privacy in the

Bayes Privacy framework, we need to explicitly model the background knowledge of the adversary. Since the intention of Differential Privacy is to hide for any individual if it is represented in the database, we assume the worst adversary that, for a given individual, knows if all other individuals are in the database. This can be done with the following universe and distributions:

Definition 29 (Differential Privacy Background Knowledge)**.** *For a universe U containing all possible data sets of a given domain, we define the set $\mathcal{X}_{U,dp}$ as the set of all distributions of U with the following property: There are data sets u_0 and u_1 with probability $p_0, p_1 > 0$, with $d_0 \Delta d_1 = 1$. All other data sets have probability 0.*

Differential Privacy hides for any individual whether it is in the database. This can be formalised in the form of predicates as follows:

Definition 30 (Differential Privacy Predicates)**.** *For a given universe U with data about individuals $I_1, \dots I_n$, we define the set of predicates $L_{U,dp}$ as the set that contains all predicates of the form:*

$$\text{individual } I_i \text{ is in the database } d$$

for all individuals I_i.

With these definitions, we can state and prove the following lemma:

Proposition 4. *Let U be a universe of databases. Then, a mechanism F that fulfils $(\frac{e^\epsilon}{1+\Pr[I](e^\epsilon-1)}, \delta, L_{U,dp}, \mathcal{X}_{U,dp})$-privacy also fulfils (ϵ, δ)-Differential Privacy.*

Proof. For the sake of readability, in this proof, we write L instead of $L_{U,dp}$ and $\mathcal{X}$ instead of $\mathcal{X}_{U,dp}$. In order to prove Proposition 4, we use Bayes' Theorem the law of total probability for binary events (cf. Lemma 1). According to Definition 26 fulfilling $(\frac{e^\epsilon}{1+\Pr[I](e^\epsilon-1)}, \delta, L, \mathcal{X})$-privacy means, that the probability of all breaches with a maximum magnitude of $\frac{e^\epsilon}{1+\Pr[I](e^\epsilon-1)}$ is equal or less than $1-\delta$. Therefore, for all $I \in L$ and all $d \in U$ holds:

$$\Pr[b(I,s) \leq \frac{e^\epsilon}{1+\Pr[I](e^\epsilon-1)}] \leq 1-\delta$$

For $\Pr[I] = 0$, we get:

$$1 = b(I,s) \leq \frac{e^\epsilon}{1+\Pr[I](e^\epsilon-1)} = e^\epsilon$$

This is true for all $\epsilon \geq 0$. For $\Pr[I] = 1$, we get:

$$1 = b(I,s) \leq \frac{e^\epsilon}{1+\Pr[I](e^\epsilon-1)} = \frac{e^\epsilon}{e^\epsilon}$$

This also is true for all ϵ. Therefore, we only have to examine the case where $\Pr[l] > 0$ and $\Pr[\neg l] > 0$. This is the case when l distinguishes the two databases d_0 and d_1. Let w.l.o.g. $l(d_0) = \mathit{true}$ and $l(d_1) = \mathit{false}$. Then, the following holds:

$$\begin{aligned}
& b(l,s) \leq \frac{e^\epsilon}{1+\Pr[l](e^\epsilon-1)} \\
\Leftrightarrow \quad & \frac{\Pr[l|s]}{\Pr[l]} \leq \frac{e^\epsilon}{1+\Pr[l](e^\epsilon-1)} \\
\Leftrightarrow \quad & \frac{\Pr[l|s]}{\Pr[l]}\Pr[s] \leq \frac{e^\epsilon}{1+\Pr[l](e^\epsilon-1)}\Pr[s] \\
\Leftrightarrow \quad & \frac{\Pr[l|s]}{\Pr[l]}\Pr[s] \leq \frac{e^\epsilon}{1+e^\epsilon\Pr[l]-\Pr[l]}\Pr[s]
\end{aligned}$$

With Bayes' Theorem, we get:

$$\begin{aligned}
\Leftrightarrow \quad & \Pr[s|l] \leq \frac{e^\epsilon}{1+e^\epsilon\Pr[l]-\Pr[l])}\Pr[s] \\
\Leftrightarrow \quad & \Pr[s|l](1+e^\epsilon\Pr[l]-\Pr[l]) \leq e^\epsilon\Pr[s] \\
\Leftrightarrow \quad & \Pr[s|l](1-\Pr[l]) \leq e^\epsilon(\Pr[s]-\Pr[s|l]\Pr[l]) \\
\Leftrightarrow \quad & \Pr[s|l] \leq e^\epsilon\frac{\Pr[s]-\Pr[s|l]\Pr[l]}{1-\Pr[l]}
\end{aligned}$$

Since $\Pr[l] = 1 - \Pr[\neg l]$, this is equal to:

$$\Leftrightarrow \quad \Pr[s|l] \leq e^\epsilon\frac{\Pr[s]-\Pr[s|l]\Pr[l]}{\Pr[\neg l]}$$

With the law of total probability, we get:

$$\Leftrightarrow \quad \Pr[s|l] \leq e^\epsilon\Pr[s|\neg l]$$

Since l distinguishes the two databases d_0 and d_1, this es equal to:

$$\Leftrightarrow \quad \Pr[F(d_0) = s] \leq e^\epsilon\Pr[F(d_1) = s]$$

This only holds, when the mechanism F fulfils ϵ-Differential Privacy.

□

Proposition 4 implies that the largest possible breach for a sensitive predicate gets smaller when the probability of this predicate gets larger. Since, in order to satisfy Differential Privacy, for any sensitive predicate $l \in L$ there is also its negation $\neg l \in L$, the largest possible breach for all $l \in L$ is smallest for $\Pr[l] = \frac{1}{2}$ for all sensitive predicates l. For $\Pr[l] = \frac{1}{2}$, we get:

$$b(l,s) \leq \frac{e^\epsilon}{1+\Pr[l](e^\epsilon-1)} = \frac{e^\epsilon}{\frac{1}{2}e^\epsilon+\frac{1}{2}} = 2\frac{e^\epsilon}{e^\epsilon+1} < 2$$

This means, that if for an individual, the probability to be represented in the database is equal to the probability to not be represented in the original database, Differential Privacy implies an upper bound for the worst privacy breach for this individual. This upper bound is independent of the security parameter ϵ of Differential Privacy since with a breach of 2 for an individual, the adversary learns if this individual is in the database, which is prevented by Differential Privacy.

In Section 4 we showed that the Bayes Privacy framework allows to study the composition properties of privacy notions. We know that Differential Privacy composes linearly. This means that the composition of two mechanisms that are ϵ_1 and ϵ_2 differentially private, respectively, is $\epsilon_1 + \epsilon_2$ differentially private [McS09]. This can also be shown in the Bayes Privacy framework.

Proposition 5. *Let U be a universe of databases and E be the identity function. Let M_1 and M_2 be mechanisms that fulfil ($\frac{e_1^\epsilon}{1+\Pr[I](e_1^\epsilon-1)}$, δ, $L_{U,dp}$, $\mathcal{X}_{U,dp}$)-privacy and ($\frac{e_2^\epsilon}{1+\Pr[I](e_2^\epsilon-1)}$, δ, $L_{U,dp}$, $\mathcal{X}_{U,dp}$)-privacy with respect to E, respectively.*

Then, the mechanism $M = (M_1, M_2)$ fulfils ($\frac{e^{\epsilon_1+\epsilon_2}}{1+\Pr[I](e^{\epsilon_1+\epsilon_2}-1)}$, δ, $L_{U,dp}$, $\mathcal{X}_{U,dp}$)-privacy.

Proof. In the following, we show with Proposition 2, that the biggest breach b of a mechanism, that is a composition of two mechanisms fulfilling ϵ_1-Differential Privacy and ϵ_2-Differential Privacy, respectively, is bound by $\frac{e^{\epsilon_1+\epsilon_2}}{1+\Pr[I](e^{\epsilon_1+\epsilon_2}-1)}$.

In order to apply Proposition 2, we need to show, that the two privacy notions in Proposition 5 are compatible. The possible background knowledge of the adversary $\mathcal{X}_{U,dp}$ contains all distributions according to Definition 29. After a release with a differentially private mechanism, the a posteriori knowledge of the adversary is also in $\mathcal{X}_{U,dp}$. Otherwise she would have learned for an individual if it is in the database. Therefore the two privacy notions in Proposition 5 are compatible. Now we can apply Proposition 2. From equation 4.2 follows for distributions $X \in \mathcal{X}$ and $X' \in \mathcal{X}'$, where X' is the a posteriori distribution of X after the release of s_1:

$$b(I, s) \leq \frac{e^{\epsilon_1}}{1 + \Pr[I|X](e^{\epsilon_1} - 1)} \cdot \frac{e^{\epsilon_2}}{1 + \Pr[I|X'](e^{\epsilon_2} - 1)}$$

We need to show the following equation:

$$b(I, s) \leq \frac{e^{\epsilon_1+\epsilon_2}}{1 + \Pr[I|X](e^{\epsilon_1+\epsilon_2} - 1)} \tag{4.3}$$

Therefore, we show that:

$$\frac{e^{\epsilon_1}}{1 + \Pr[I|X](e^{\epsilon_1} - 1)} \cdot \frac{e^{\epsilon_2}}{1 + \Pr[I|X'](e^{\epsilon_2} - 1)} \leq \frac{e^{\epsilon_1+\epsilon_2}}{1 + \Pr[I](e^{\epsilon_1+\epsilon_2} - 1)}$$

Since each denominator and numerator in this equation is greater than 0, we get:

$$\begin{aligned}
&e^{\epsilon_1+\epsilon_2}(1 + \Pr[I|X](e^{\epsilon_1+\epsilon_2} - 1)) \leq e^{\epsilon_1+\epsilon_2}((1 + \Pr[I|X](e^{\epsilon_1} - 1)) \cdot (1 + \Pr[I|X'](e^{\epsilon_2} - 1)))\\
&\Leftrightarrow\\
&1 + \Pr[I|X](e^{\epsilon_1+\epsilon_2} - 1) \leq (1 + \Pr[I|X](e^{\epsilon_1} - 1)) \cdot (1 + \Pr[I|X'](e^{\epsilon_2} - 1))\\
&\Leftrightarrow\\
&1 + \Pr[I|X]e^{\epsilon_1+\epsilon_2} - \Pr[I|X] \leq 1 + \Pr[I|X]e^{\epsilon_1} - \Pr[I|X]\\
&\qquad + \Pr[I|X']e^{\epsilon_2} + \Pr[I|X]\Pr[I|X']e^{\epsilon_1+\epsilon_2} - \Pr[I|X]\Pr[I|X']e^{\epsilon_2}\\
&\qquad - \Pr[I|X'] - \Pr[I|X]\Pr[I|X']e^{\epsilon_1} + \Pr[I|X]\Pr[I|X']
\end{aligned}$$

We subtract 1 on both sides, divide through $\Pr[I|X]$ and, for the sake of readability, we set $c := \frac{\Pr[I|X']}{\Pr[I|X]}$ (Please note, that $c > 0$.):

$$e^{\epsilon_1+\epsilon_2} - 1 \leq e^{\epsilon_1} - 1 + ce^{\epsilon_2} + \Pr[I|X']e^{\epsilon_1+\epsilon_2} - \Pr[I|X']e^{\epsilon_2} - c - \Pr[I|X']e^{\epsilon_1} + \Pr[I|X']$$

$$\Leftrightarrow$$

$$0 \leq c(-1 + e^{\epsilon_2}) + \Pr[I|X'](e^{\epsilon_1+\epsilon_2} - e^{\epsilon_1} - e^{\epsilon_2} + 1) + e^{\epsilon_1} \tag{4.4}$$

Since $\epsilon_1, \epsilon_2 > 0$, the following holds:

- $-1 + e^{\epsilon_2} > 0$
- $e^{\epsilon_1+\epsilon_2} + 1 > e^{\epsilon_1} + e^{\epsilon_2}$

As a result, Equation 4.4 is true. Therefore, Equation 4.3 also is true.

□

5.2. Averages

Averages are commonly used without even questioning the privacy properties of the averaging mechanism. A common example are election periods, where projections of the result and fine grained results of every district are published frequently. Without strong assumptions about the distribution of the universe, there are cases when the averaging mechanism fails to provide privacy. A meaningful notion for this mechanism can not be modelled in frameworks that only consider the worst case. For example, if all individuals voted for the same party, the average over all votes allows to determine the vote of each individual.

In this section, we will show how the privacy properties of the averaging mechanism can be captured in the Bayes Privacy framework. A similar approach can be found in [Bha+11], where the entropy in the original database is used to privacy preservingly answer linear queries over reals in a database without the addition of noise to the result. Their notion ϵ-Noiseless Privacy is similar to Differential Privacy and only considers the worst case. It is achieved through strong assumptions about the adversaries background knowledge and the distribution of attribute values and possible databases.

Since the Bayes Privacy framework allows to model privacy notions that fail to provide privacy in some rare cases, it allows for modelling a notion for a mechanism that calculates the average value of an attribute for a given data set without such strong assumptions. For example publishing the average of all salaries of the employees of a company intuitively preserves the privacy of individuals if the uncertainty of the adversary is high enough. However, there are possible but very improbable average salaries that cause high breaches. If a privacy notion does not allow for high breaches with a small probability, averaging can not be considered to be a privacy preserving mechanism. Consider the following example, where each salary is either 1, 2, or 3 and each salary has an identical independent probability of $p = \frac{1}{3}$. If the company employs 3 individuals, every database has a probability of $(\frac{1}{3})^3$. W.l.o.g., the salary of employee A is considered sensitive and the goal is to publish the average salary of all employees. Figure 4.15 shows an example for an original database where the average salary is $\frac{7}{3}$.

There are 3 different salaries and 3 employees, therefore there there are 27 different databases and 7 different possible average salaries. For a given salary $x \in \{1, 2, 3\}$ the

employee	salary
A	1
B	3
C	3

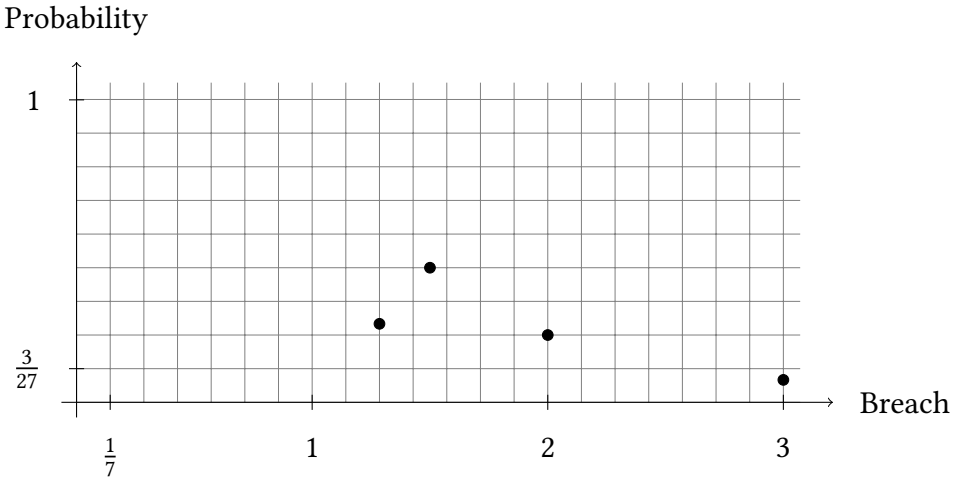
Figure 4.15.: An example database that shows the salaries of three employees. Here, the average salary is $\frac{7}{3}$. In our example, there are 6 possible databases with this average salary.

average salary s	probability of release $\Pr[s]$	conditional probability of I $\Pr[\text{salary}(A) = 1, 2, 3 \mid s]$	breach $b(s)$
1	$\frac{1}{27}$	1, 0, 0	3
$\frac{4}{3}$	$\frac{3}{27}$	$\frac{2}{3}, \frac{1}{3}, 0$	2
$\frac{5}{3}$	$\frac{6}{27}$	$\frac{3}{6}, \frac{2}{6}, \frac{1}{6}$	$\frac{3}{2}$
2	$\frac{7}{27}$	$\frac{2}{7}, \frac{3}{7}, \frac{2}{7}$	$\frac{9}{7}$
$\frac{7}{3}$	$\frac{6}{27}$	$\frac{1}{6}, \frac{2}{6}, \frac{3}{6}$	$\frac{3}{2}$
$\frac{8}{3}$	$\frac{3}{27}$	$0, \frac{1}{3}, \frac{2}{3}$	2
3	$\frac{1}{27}$	0, 0, 1	3

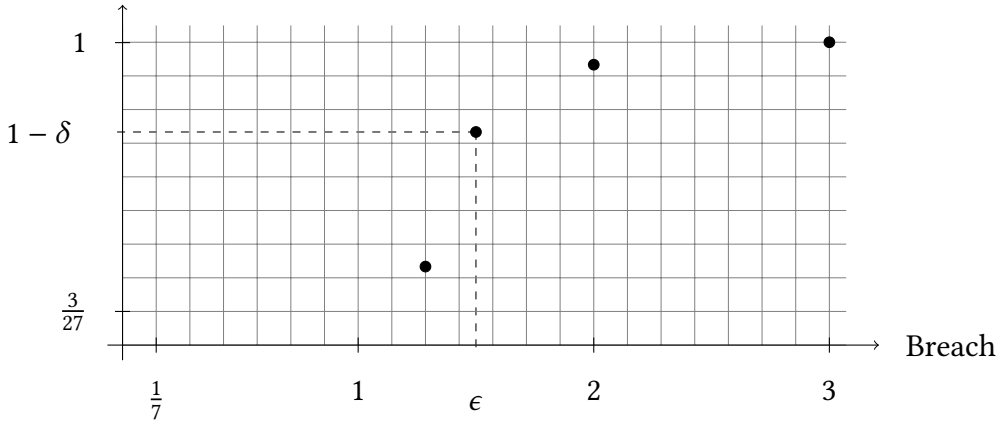
Figure 4.16.: Average salaries and their probabilities in the averaging mechanism example. Figure 4.17 depicts the breach distribution and the cumulative breach distribution.

a priori probability that employee A has salary x is $\frac{1}{3}$. Note that we chose this simple distribution as an example. Different distributions of databases, that do not assume independence of attribute values can also be used in the Bayes Privacy framework. The a priori probability that an employee has a certain salary changes after publication of the average salary. The table in Figure 4.16 shows the probabilities of each average salary, and the breach caused by its publication.

Publishing an average salary of 2, for example, results in a breach of $\frac{9}{7}$. Publishing an average salary of 1 results in fully disclosing the salary of A and a breach of 3. This event, however, has a probability of $\frac{1}{27}$ and, therefore, is rather unlikely. In this example, publishing the average is a mechanism that fulfils privacy with $\epsilon = \frac{3}{2}$ and $\delta = \frac{8}{27}$. Figure 4.17 depicts the breach distribution and the cumulative breach distribution as well as ϵ and δ for this example. This is a very simple example and larger databases with a bias to an average salary different to the expected value are even less probable. This will result in a smaller ϵ and δ. Consequently, high breaches are even less probable in larger examples.



4.17.1: Breach distribution. The maximal breaches of 2 and 3 have a low probability.



4.17.2: Cumulative breach distribution.

Figure 4.17.: Breach distribution 4.17.1 and cumulative breach distribution 4.17.2 for the averaging mechanism example. Even in this small example, the thresholds δ and ϵ are relatively small.

6. Privacy with Respect to Bounded Adversaries

Notions defined in the Bayes Privacy framework do not allow for methods based on computational complexity assumptions. For example simply encrypting a database with RSA prior to release will always lead to a high breach since an optimal bayes estimator is able to guess the plaintext to an RSA ciphertext. There are, however, many mechanisms that rely on complexity assumptions which are deemed to be secure, since a real world adversary is always bounded in its computational power. Therefore, privacy notions that consider complexity assumptions allow for much more methods than statistical privacy notions without losing security in the real world.

Computational privacy notions enables the use of, for example, encryption and cryptographic hash functions in mechanisms for database anonymisation. Another example is the average mechanism. The example in Section 5 showed, that there is an intuitive

(ϵ, δ)-privacy notion that the averaging mechanism fulfils. Since the problem of inverting a mechanism that computes the average value of an attribute can be seen as a knapsack problem, it is computationally hard to invert the averaging mechanism. Therefore, mechanisms such as the averaging mechanism may provide an even stronger privacy guarantee in a model that considers computational complexity than in a purely statistical model.

In this section, we will present a variant of the Bayes Privacy framework that allows to model privacy notions with respect to bounded adversaries. Furthermore, in order to show that game-based security notions also can be modelled as Bayes Privacy notions, we model Ind-ICP, a computational security notion for secure database outsourcing (cf. Definition 53 in Chapter 6), as a Computational Bayes Privacy notion.

6.1. Computational Bayes Privacy

In the following, we will adapt the Bayes Privacy framework presented in Section 2 with respect to bounded adversaries. In order to do so, the definition of a breach needs to be adapted. In Section 2, a breach is defined without considering a computationally bounded adversary. For a bounded adversary, a breach can be defined by the relation of the probability that the adversary determines the value of a sensitive predicate given a release and the probability that the sensitive predicate has this value.

Now, a breach is not caused by the release but rather by the adversary interpreting the release. Therefore, we define the following experiment:

Security Game 3 ($Priv^{\mathcal{A}}(X, I, s, k)$)**.**

1. *The adversary $\mathcal{A}$ is given the inputs 1^k, X, I, and s.*
2. *$\mathcal{A}$ outputs a guess b.*

The result of the experiment is b.

In this experiment, the adversary gets the background knowledge X and the sensitive predicate I. Additionally, she can learn the release s. If the adversary can determine the value of the sensitive predicate better than without it, the adversary causes a privacy breach. If the adversary gains no information about the value of the sensitive predicate I from the release, there is no privacy breach for I because of the release.

Note that if X can not be sampled efficiently by the adversary and the adversary might not learn something about the value of I from the release s, the probability, that she guesses the value of I right ($Pr[Priv^{\mathcal{A}}(X, I, s, k) = I(u)]$) may be lower than the probability that an optimal Bayes estimator guesses the value of I right without the release ($Pr[I = I(u)]$).

As mentioned above, we define a breach by the relation of the probabilities that the adversary wins this experiment to the probability that an maximum likelihood estimator without access to the release determines the value of the sensitive predicate.

Definition 31 (Privacy Breach for a Predicate and a Release Caused by an Adversary)**.** *Let U be a universe of data sets with distribution X, $E : U \to D$ and $F : D \to S$ mechanisms, $I \in L$ a sensitive predicate, $\mathcal{A}$ an adversary, and k a security parameter.*

A breach b for the predicate I caused by an adversary $\mathcal{A}$ given a release $s \in S$ is defined as:

$$b(l, s, \mathcal{A}) := \begin{cases} 1 & Pr[l] \in \{0, 1\} \\ \frac{\Pr[Priv^{\mathcal{A}}(X,l,s,k)=l(u)]}{Pr[l=l(u)]} & otherwise \end{cases}$$

while $l(u)$ is the value of the sensitive predicate l given the world data set u.

The adversary $\mathcal{A}$ causes a maximal breach if she maximises the probability of guessing $l(u)$ right. The probability $\Pr[Priv_{\mathcal{A}}(X, l, s, k) = l(u)]$ is maximal, if the adversary determines the value of the sensitive predicate in Security Game 3 and returns it.

Since we do not only want to be able to consider breaches a single sensitive predicate, but for sets of sensitive predicates, we define the breach $b(s, \mathcal{A})$ with respect to a set of sensitive predicates analogously to Definition 23:

Definition 32 (Privacy Breach for a Set of Predicates an a Release Caused by an Adversary)**.** *Let U be a universe of data sets with distribution X, $E : U \to D$ and $F : D \to S$ mechanisms, L a set of sensitive predicates, $\mathcal{A}$ an adversary, and k a security parameter. A breach b caused by the adversary $\mathcal{A}$ for a set of sensitive predicates L given a release $s \in S$ is defined as:*

$$b(s, \mathcal{A}) := \max_{l \in L} b(s, l, \mathcal{A})$$

As in Definition 24, we define the probability of an specific breach, but now with respect to an adversary:

Definition 33 (Breach Probability with Respect to an Adversary)**.** *Let U be a universe of data sets with distribution X, $E : U \to D$ and $F : D \to S$ mechanisms, and L a set of sensitive predicates. The probability of a specific breach b' with respect to an adversary $\mathcal{A}$ is called* breach probability of b' with respect to adversary $\mathcal{A}$ *and is defined as:*

$$\Pr(b', \mathcal{A}) := \Pr_{s \in S}[b' = b(s, \mathcal{A})]$$

The probability is taken over the distribution X, the random coins of the adversary, and the random coins used in F and E.

With this definition of breach probabilities, we can define a computational privacy notion by restricting adversaries that can generate breaches to probabilistic polynomial time algorithms. As in the statistical setting, a dominating function limits the distribution of allowed breaches.

Definition 34 (Computational Bayes Privacy Notion)**.** *For a given universe of possible data sets U, a* computational privacy notion *is a tuple $(L, p, \mathcal{X})$, where L is a set of sensitive predicates, $p : (1, \infty) \to [0, 1]$ a function, and $\mathcal{X}$ is a set of distributions X. An anonymisation mechanism F with a security parameter k fulfils a privacy notion $(L, p, \mathcal{X})$ with respect to a mechanism E if for all distributions $X \in \mathcal{X}$ of the universe U and all PPT (probabilistic polynomial time) adversaries $\mathcal{A}$ the following holds:*

There is a function q that is negligible in k such that the function $p + q(k)$ dominates the breach probability distribution with respect to the adversary $\mathcal{A}$ in the interval $(1, \infty)$ for the universe U, the distribution X, and the mechanisms E and F.

If the mechanism E is the identity function, we say that F fulfils the computational privacy notion $(L, p, \mathcal{X})$ instead of F fulfils the computational privacy notion $(L, p, \mathcal{X})$ with respect to the mechanism E.

Each distribution $X \in \mathcal{X}$ models possible background knowledge given to the adversary. A mechanism fulfils a privacy notion, if it only produces releases, that enable an adversary only to cause breaches limited by the notion. These breaches are limited by the dominating function p and a function that is negligible in the security parameter of the anonymisation mechanism.

In Section 6.2, we will show that IND-ICP, a practical security notion for secure database outsourcing, can be modelled as a Computational Bayes Privacy notion.

6.2. IND-ICP as a Computational Bayes Privacy Notion

Computational privacy notions can be useful for searchable encryption and secure database outsourcing, since they allow to use methods like encryption, signatures, or cryptographic hash functions in the anonymisation mechanism.

In Chapter 6, Section 2, we introduce the game-based security notion IND-ICP. The notion IND-ICP allows for practical mechanisms for secure database outsourcing and searchable encryption. Although this notion is weaker than classical notions for encryption (e. g. IND-CPA), it has a game-based definition that is similar to the experiment used in the definition of IND-CPA (cf. Definition 1 in Chapter 2 and Definition 14 in Chapter 6).

In this section, we use the Computational Bayes Privacy framework to model Computational Bayes IND-ICP. We introduce the notion Computational Bayes IND-ICP and show that this notion is the same notion as IND-ICP. This proof serves as an example that game-based security notions can also be modelled in the Bayes Privacy framework.

For the proof in this section, we reference definitions from Chapter 6. In the following, we define computational Bayes IND-ICP. For this definition, we use *independent column permutations* (independent column permutation (icp)), a special form of database transformations (cf. Definition 52, Chapter 6).

Definition 35 (Computational Bayes IND-ICP)**.** *For*

- *a universe U that contains for each database $d \in U$ all databases d' with $\exists$ icp π with $\pi(d) = d'$.*
- *a set of distributions $\mathcal{X}$, with: $\forall\, d\, \exists\, X \in \mathcal{X}$ with $\forall$ icp π: $\Pr[d] = \Pr[\pi(d)] > 0$ and $\Pr[d'] = 0$ iff $\nexists$ icp π with $d = \pi(d')$.*
- *for all attributes $i \neq i'$ a set of all predicates L of the form:* The value $v_i[j]$ of attribute i stands in relation to the value $v_{i'}[j']$ of attribute i', *where v_i is a list of all values of attribute i*
- *a function $p : (1, \infty) \to 0$.*

We call the computational privacy notion $(L, p, \mathcal{X})$ Computational Bayes IND-ICP.

In order to define a Computational Bayes Privacy notion, we defined the universe and the background knowledge of the adversary explicitly. Informally, the idea of IND-ICP is to hide relations between attribute values. Therefore, we assume that an adversary knows everything about the database except the relations of the attribute values. This is modelled with distributions where all databases with identical attribute values but different relations between attribute values are equally probable. This is an implicit assumption of IND-ICP. In the Bayes Privacy-framework, we have to model this assumption explicitly.

An adversary with background knowledge $X \in \mathcal{X}$ knows the size of the original database and how often each attribute value occurs in the original database. Furthermore, we need to define the set of predicates that the notion considers as sensitive. We do this directly by defining each relation of two attribute values as a sensitive predicate.

The notion IND-ICP does not allow the adversary to learn anything about the relations of the attribute values. Therefore, we set the dominating function p to 0.

With this definition of computational Bayes IND-ICP, the following proposition holds:

Proposition 6. *A mechanism that fulfils IND-ICP for all databases in U, also fulfils Computational Bayes IND-ICP. A mechanism that fulfils Computational Bayes IND-ICP for all databases in U, also fulfils IND-ICP.*

In order to prove Proposition 6, we assume in the first step, that there is an adversary that can break IND-ICP and use her to break Computational Bayes IND-ICP. In the second step, we assume there is an adversary that can break computational Bayes IND-ICP and use her to break IND-ICP. In the following, we will prove this proposition by reducing the two experiments $Priv_{\mathcal{A}}(X, I, s, k)$ and $\text{IND-ICP}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}(k)$ to each other.

Proof. Let $U, \mathcal{X}, L$, and p be as defined in Definition 35.

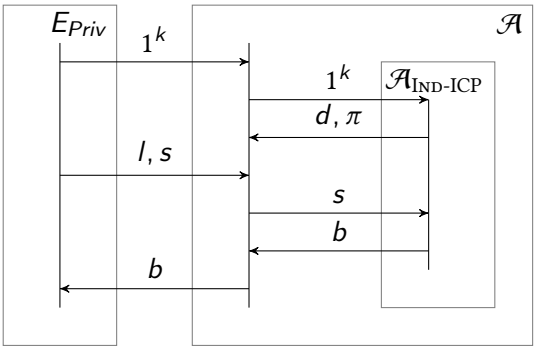


Figure 4.18.: The adversary $\mathcal{A}_{\text{IND-ICP}}$ that wins the experiment IND-ICP is used to win the Privacy experiment for computational Bayes IND-ICP. The Adversary $\mathcal{A}$ wins the epxeriment if $f(s) \in d, \pi(d)$ and $I(\pi(d)) \neq I(d)$.

i)
Let $\mathcal{A}_{\text{IND-ICP}}$ be a PPT that can break IND-ICP. This means, that the advantage of $\mathcal{A}$ according to Definition 53 in Chapter 6 is not negligible.

We will show, how a non negligible advantage in the IND-ICP experiment can be used to produce a breach of $(L, p, \mathcal{X})$-privacy. If the adversary $\mathcal{A}_{\text{IND-ICP}}$ can break IND-ICP, she wins the game $\text{IND-ICP}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}(k)$ (cf. Definition 14, Chapter 6) with a probability of $\frac{1}{2} + q$, while q is non negligible in k. We use this adversary to generate a breach of $(L, p, \mathcal{X})$ privacy with non negligible probability. Figure 4.18 shows the reduction of the privacy experiment of $(L, p, \mathcal{X})$-privacy to the adversary $\mathcal{A}_{\text{IND-ICP}}$ of the IND-ICP experiment $\text{IND-ICP}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}(k)$.

The adversary $\mathcal{A}_{\text{IND-ICP}}$ returns a database d and an independent column permutation π for which she has a non negligible advantage. This means, that the adversary can distinguish $f(d)$, the anonymisation of d, from $f(\pi(d))$, the anonymisation of the independent column permutation of d, non negligibly better than randomly guessing.

Since the independent column permutation chosen by the adversary cannot be id, there is at least one pair of attribute values $(v_i[j], v_{i'}[j'])$ with $i \neq i'$ that are in relation in d but are not in relation to each other in $\pi(d)$. The adversary learns the relation of these attribute values from $f(d)$ or $f(\pi(d))$, respectively, with probability $\frac{1}{2} + q$.

Since we modelled these relations as sensitive predicates, our adversary $\mathcal{A}$ that uses the adversary $\mathcal{A}_{\text{IND-ICP}}$ learns this sensitive predicate with probability $\frac{1}{2} + q$. This results in a breach probability > 0 for a breach for this sensitive predicate. Since this predicate is w.l.o.g. true for the database d and false for database $\pi(d)$, this breach is greater than 1.

This means, that our adversary $\mathcal{A}$ wins the experiment $Priv_{\mathcal{A}}(X, I, f(d'), k)$ iff:

- the database d' given to $\mathcal{A}$ by the experiment matches the database d or the database $\pi(d)$ returned by $\mathcal{A}_{\text{IND-ICP}}$.
- the sensitive predicate I can be used to distinguish d from $\pi(d)$.

What is left, is to show that this happens with a non negligible probability:

The size of the databases in U is limited (and independent of the security parameter). For a database d, there is a limited (and independent of the security parameter) number of independent column permutations. Therefore, the probability of d is non negligible. Consequently, the database d' given to $\mathcal{A}$ by the experiment matches the database d or the database $\pi(d)$ returned by $\mathcal{A}_{\text{IND-ICP}}$ with a non negligible probability.

Furthermore, also there is a limited number of independent column permutations, the adversary $\mathcal{A}_{\text{IND-ICP}}$ can return, the probability, that I can be used to distinguish d and $\pi(d)$ is also non negligible. Since our adversary $\mathcal{A}$ knows I (input from the experiment) and π and d (input from $\mathcal{A}_{\text{IND-ICP}}$), she can determine, when I can be used to distinguish d from $\pi(d)$.

However restarting the adversary $\mathcal{A}_{\text{IND-ICP}}$ after our adversary $\mathcal{A}$ got the sensitive predicate I from the experiment may not yield such a database d and an independent column permutation I. This is because d and pi may not be in the set of databases or independent column permutations, respectively, the adversary $\mathcal{A}_{\text{IND-ICP}}$ can return. Then however there is sensitive predicate I' that can be used to distinguish $f(d)$ and $f(\pi(d))$, and the adversary $\mathcal{A}_{\text{IND-ICP}}$ can be used in the experiment $Priv_{\mathcal{A}}(X, I', f(d'), k)$ in order to cause a breach.

ii)

Now, let the adversary $\mathcal{A}_{Priv}$ be a PPT that can break Computational Bayes IND-ICP. This means, that the adversary can cause a breach non negligibly greater than 1 with a non negligible probability. Consequently, there is at least one sensitive predicate $I' \in L$ and one database d' for which $\mathcal{A}_{Priv}$ can determine the value $I(d)$ non negligibly better than random guessing.

We simply use this adversary in the experiment $\text{IND-ICP}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}(k)$ as follows (cf. Figure 4.19):

In the first step, we get an security parameter which we forward to the adversary $\mathcal{A}_{Priv}$. Now, $\mathcal{A}_{Priv}$ is expecting a distribution X and I and the experiment $\text{IND-ICP}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}(k)$ is expecting an independent column permutation π and a database d. We choose the sensitive predicate I, the database d, and the independent column permutation π randomly but with $I(d) \neq I(\pi(d))$. This means, that the predicate I can be used to distinguish the databases $I(d)$ and $I(\pi(d))$. For X we chose the distribution from $\mathcal{X}$ where $\forall$ icp $\pi : \Pr[d] = \Pr[\pi(d)] > 0$.

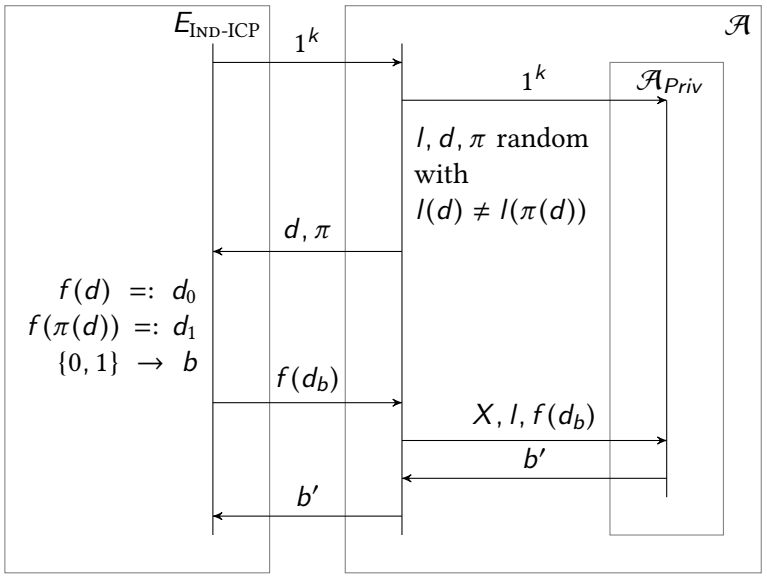


Figure 4.19.: The adversary $\mathcal{A}_{Priv}$ that wins the computational Bayes privacy experiment is used to win the experiment IND-ICP experiment. The adversary $\mathcal{A}$ chooses I and d randomly. Therefore, $\mathcal{A}$ eventually chooses I and d so that the adversary $\mathcal{A}_{Priv}$ has a non negligible advantage in determining the value of I from $f(d)$. Then, and if $f(d_b) = f(d)$ the adversary $\mathcal{A}$ has an non negligible advantage in the computational Bayes privacy experiment.

We get an anonymisation $f(d_b)$ from the experiment and forward it to the adversary $\mathcal{A}_{Priv}$. Then, we get a bit b' from $\mathcal{A}_{Priv}$ and forward it to the experiment.

Since we choose I and d random, we eventually choose I and d so that the adversary can determine the value of I from $f(d)$ non negligibly better than guessing. This is the case, when $I = I'$ and $d = d'$

In these cases, we get with a probability of $\frac{1}{2}$ $f(d') = f(d_b)$ from the experiment. Then, IND-ICP will not hold, since the bit b' we get from the adversary will be equal to the bit b chosen at random in the experiment with a probability non negligibly greater than $\frac{1}{2}$. Therefore, Proposition 6 holds. □

The advantage of modelling notions such as IND-ICP in the Bayes Privacy framework is that it forces already in the definition of the notion to explicitly formalise the background knowledge of the adversary. In the game-based definition provided in Chapter 6, Section 2, this is only made implicitly and made explicitly not until used in a proof for a method that realises IND-ICP. Implicit assumptions for example about the background knowledge of the adversary may lead to a false sense of security and to uses of a mechanism that enable attacks that break the security notion. We will show in Chapter 7, Section 8 that additional background knowledge of an adversary about the distribution of the original databases enables her to break a method that provably fulfils IND-ICP.

5. Privacy for Data Outsourcing

This chapter is partially based on work already published in [Hub+11; Hei+10] and in [Ach+16].

1. Introduction

In the last two decades, the IT industry has seen numerous trends such as service orientation, mobile IT, and cloud computing. A majority of these trends is connected with a common paradigm shift concerning data processing and storage: Data is processed and stored less and less on local devices or in a self maintained computing centre, but is outsourced to external data processing centres and accessed over the Internet *as a service*. There are examples for applications such as office suites [Mic; Goo], image manipulation software [Ado; Aut], application development systems [IBM; Cod], and even desktop environments [Ama; Cit; VMw] that can be accessed as a service.

Outsourcing data and computation has many advantages, for example more efficient use of computing and storage resources. Furthermore, pay-per-use models can reduce costs for IT infrastructures. Also, the risk of fatal data loss or incidents caused by external hackers is minimised due to specialisation of the individual service providers.

There are, however, still security concerns that pose a huge impediment for data outsourcing. A client using a service loses control over his data. For example, the client can not control whether his data is copied or misused. An adversary, like a malicious system administrator, may copy sensitive data and sell it to e. g. competitors. Providers may even be required by law to disclose the data of their clients to government agencies. Current security measures of cloud computing providers focus on external adversaries. For example, authentication and authorisation mechanisms prohibit access from unauthorised clients and encryption and signature schemes protect the client's data during sending over the Internet. Measures against internal adversaries mostly try to restrict physical access to servers to authorised staff. Examples such as Wikileaks, however, show that legal insiders also have to be taken into account.

In this chapter, we provide a framework for security notions for data outsourcing, that takes internal adversaries into account. We distinguish three fundamental security goals, namely *Data Privacy*, *Query Privacy*, and *Result Privacy*, and show their relations. In contrast to plain encryption, outsourced data is accessed, for example in order to find the occurrences of a keyword or in order to retrieve a tuple with certain conditions. An adversary observing such accesses potentially gains additional information that can be used to break the confidentiality of the data, the queries, or the result.

The goal of Query Privacy is related to *private information retrieval (PIR)*, where the goal is to achieve query indistinguishability for queries to an outsourced database. For the sake of efficiency, the time to access a single field in a database should be sublinear in the size of the database. For single server solutions, this is in conflict with the requirement of query indistinguishability. Query indistinguishability for single server solutions requires

that each field is part of the result. Otherwise, the adversary can use the fields that are not part of the result sets to distinguish queries. In real world scenarios, this security requirement might be to strong. Sometimes it may be sufficient, that the adversary can not distinguish two queries, if they are in a certain subset of all possible queries. For example, for certain applications it may be enough that an adversary observing queries is not able to break the security notion of the encryption scheme used to encrypt the outsourced data. Therefore, we generalise the framework in order to support the definition of application specific security notions. We introduce leakage relations for data and queries that allow the adversary to learn certain properties about the database or the query.

Structure of this Chapter Section 2 provides an overview over security notions for data outsourcing in literature and motivates the three distinct privacy goals of data outsourcing. We define basic privacy notions for each privacy goal of data outsourcing in Section 3. In Section 4, we show and prove fundamental relations of these basic privacy notions. In Section 5, we show that private information retrieval is an instance of one of these fundamental notions. Section 6 provides generalisations of the basic notions defined in Section 3 that can be used to define application specific privacy notions with a controlled leakage of information (cf. Chapter 6).

2. Security Notions for Data Outsourcing in Literature

Outsourcing data involves one or multiple clients that provide the data and one or multiple servers that store the data. To retrieve parts of the outsourced data, a client submits queries to the servers which then execute them and return the result. Applications that fit into this paradigm are not restricted to relational databases, but to any information that can be structured meaningfully, for example searchable documents, Structured Query Language (SQL) databases, and image or email archives.

As already mentioned in the introduction, entrusting private data to a service provider introduces a security risk. While an IT service provider can be trusted to provide the contractually agreed-on services faithfully, one can easily imagine that a curious employee might try to learn confidential information from the outsourced data. Cryptographic methods promise to secure confidentiality in such a scenario.

Theoretically, well-known cryptographic schemes, namely fully homomorphic encryption and secure multi party computations can be used to build schemes for secure database outsourcing and searchable encryption with strong, provable security properties.

The field of secure multi-party computation (MPC) [GMW87; Yao82] deals with the following problem: How can a set of parties, each with an individual secret input, securely perform a joint function evaluation without any of the parties learning the input of the others? A prominent MPC example is Yaos millionaire problem [Yao82]: How can two millionaires—without openly discussing their wealth—determine who is the richer of the two? Theoretically, secure multi-party computation protocols can be used in outsourcing scenarios. However, most results in MPC don't yield practical techniques applicable to outsourcing scenarios.

The field of fully homomorphic encryption (FHE) [Gen09] deals with encryption schemes with a privacy homomorphism. This homomorphism allows to perform calculations on two ciphretexts, with the result being an encryption of the sum or the product of the two plaintexts, respectively. Consequently, an FHE scheme allows to perform

arbitraty calculations on encrypted data. In theory, FHE can be applied to many scenarios that involve data outsourcing. Clients can encrypt their data prior to outsourcing, service providers perform their calculations on encrypted data and send the encrypted results back to the clients.

The huge overhead, however, would cancel out the benefits of outsourcing. For example, in order to transform an existing service into one, that operates fully homomorphic on encrypted data, the service itself has to be expressed and executed as a circuit. Moreover, there are outsourcing scenarios, where MPC or FHE can not even theoretically be applied to [VJ10]. Therefore, in the following, we will consider schemes with a trade-off between security and efficiency. For example, an efficient and practical database outsourcing scheme should provide support for sublinear query execution time in the size of the database [KC05]. Practical approaches that can be found in literature attempt to find a trade-off between security and efficient support a large number of queries.

Since we distinguish between data privacy, query privacy, and result privacy and since the latter implies the former two, which we will formally prove in Section 4, we structure the related work into three categories:

- privacy of outsourced data
- privacy of queries
- privacy of outsourced data and queries

This work focuses on game-based definitions for security notions, therefore, we also focus on presenting game-based security notions from literature.

2.1. Security Notions for Data Privacy

Security notions for the privacy of outsourced data extensively have been studied in the context of searchable encryption. The idea of searchable encryption is to encrypt data in such a way that it can efficiently be searched for the positions or presence of keywords. Consequently, searchable encryption is strongly related to database outsourcing.

There are game-based notions [Goh03; EFG10; ABO07] as well as simulation-based notions [Cas+13; Cas+14; KP13]. There are also notions modelled in the UC Framework [Can01; SPS13; KO15], a well-known framework for the definition of simulation-based security notions.

Consequently, there is a huge variance between notions, even between notions that only consider data privacy: There are notions which consider different kinds of adaptive adversaries and notions which only consider static security.

An example for an adaptive game-based security notion is *semantic security against adaptive chosen keyword attacks* (IND-CKA) established by Goh et al. [Goh03] and improved by Curtomola et al. [Cur+06]. The intuition of IND-CKA is that an adversary should not be able to distinguish two sets of data even if she can issue and observe queries on encrypted sets of data of her choosing. The queries the adversary is allowed to choose are restricted to keyword queries with keywords that occur in both data sets.

Another example for a game-based security notion for Data Privacy is *pp-security* [ABO07]. Here the adversary is not allowed to repeat queries to oracles. Moreover, the adversary has to query its oracles with different classes of queries based on the state of the experiment. These constraints regarding the oracles are well-suited for the approach

presented in [ABO07] but they might be to restrictive for a wide range of real world scenarios, for example scenarios where adversaries are allowed to issue the same query twice.

2.2. Security Notions for Query Privacy

The privacy of queries on data outsourced to a single server has been studied in the context of single server PIR [Cho+98]. The idea of PIR is to retrieve information from a database without the database learning which information has been retrieved. This lead to the notion of *query indistinguishability*.

There are various PIR schemes with sublinear *communication complexity* ([KO97; CMS99; GR05]). All single server PIR schemes inherently have a *computational complexity* which is linear in the size of the data [SC07]. In order to achieve query indistinguishability, for every query, each part of the database has to be processed in order to calculate the result. Otherwise, the server learns which parts of the database are not part of the query result. Consequently, the server can distinguish two queries. Please note that PIR does not guarantee that the adversary does not learn the data itself [Cho+98]. The adversary may learn the complete data, as long as the queries are hidden from her.

Because of their communication complexity, PIR schemes can not be considered practical in our sense. Even schemes that are *communication efficient* seem less practical than the trivial solution of transferring the complete data: According to an argument by Sion et al. [SC07] single server solutions for PIR will always remain less efficient than the trivial solution as long as Moore's Law on computation power and Nielsen's Law on bandwidth increase hold. Therefore, practical PIR schemes involve multiple servers. In [DG15], for example, Di Crescenzo and Gosh show how approaches for a two party model can be adapted in a three party model resulting in schemes with a logarithmic run time for each party. As mentioned above, the notion of PIR requires that an adversary observing access patterns cannot distinguish any two queries. This notion is adaptive. This means, that the adversary can query the oracle multiple times (even for the challenge).

In the context of Oblivious RAM (ORAM), the privacy of queries has also been studied. The concept of ORAM has first been first introduced by Goldreich and Ostrovsky [GO96] and then further explored and improved upon [PR10; Shi+11; DMN11]. An *oblivious* RAM can not distinguish access patterns. Also, similar to PIR, the data itself is not required to be private. Similar to PIR, ORAM constructions can not be considered efficient in our sense: Either they have poly-logarithmic computation cost while requiring the client to store a constant amount of data [Shi+11] or they have logarithmic computation cost but require the client to store at least a sublinear amount of data dependent on the size of the RAM [Goo+11].

2.3. Security Notions for Data Privacy as well as Query Privacy

Security notions that consider both, data privacy as well as query privacy, can also be found in literature. For example Chase et al. [CS14] model the privacy of queries and that of the data in their simulation-based notion *chosen query attacks*. In this notion, the privacy of queries and the privacy of data cannot be considered isolated. Here, both concepts are intertwined which makes it difficult to compare this notion to notions that only consider one concept and, consequently, to model schemes that only posses one of the two properties.

Haynberg et al. [Hay+13] try to separate both properties with the introduction of the notions *data privacy*, which has a somewhat-adaptive adversary, and the complement notion *pattern privacy*, which is similar to PIR. Their notion for data privacy, however, only allows the adversary to observe the execution of one query. While the notion works for their scheme, this limitation is too strict for other schemes.

2.4. Modelling Information Leakage

A reoccurring pattern in security notions for practical schemes is the usage of one ore multiple *leakage functions* [KPR12; HK14]. A leakage function describes the information the scheme leaks to the adversary during execution. In order to allow for efficient schemes, a certain amount of of leakage seems necessary. In fact, there are no efficient single server schemes without leakage of information about the queries [KC05].

In their work in [Cas+13; Cas+14], Cash et al. investigate the construction of efficient and practical schemes that also have a formal security analysis. Their analyses follow a simulation-based approach. Simulation-based notions are outside of our scope, however. Their constructions leak information about the plaintext and the query to the adversary which they explicitly model by a *leakage function* $\mathcal{L}$. The simulator is given access to this leakage function.

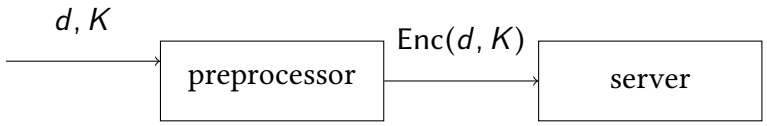
This is similar to the work of Chase et al. [CS14]. Their notion allows to describe the information that leaks through the encryption itself and the information about the ciphertext *and* the queries combined that is leaked by evaluating queries. The same technique is employed by Stefanov et al. [SPS13] in the UC Framework.

In game-based notions, leakage can be modelled by restricting the challenges the adversary can choose. In our framework, we define *leakage relations* that model information leakage and optionally restrict adversaries to choose data sets or queries fulfilling such a leakage relation.

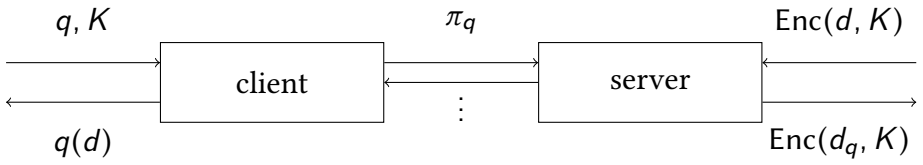
3. Formalisations

Classical encryption schemes such as, for example, the Elgamal encryption scheme [Elg85], are defined by three algorithms: *Key Generation*, *Encryption*, and *Decryption* and are classically used to send confidential messages from a sender to a receiver. Messages are de- and encrypted as a whole. In contrast, data outsourcing schemes have to provide means to search or even change the encrypted data. In this section, we provide a formal definition of a data outsourcing scheme.

We model the interactions in a data outsourcing scheme in two phases. In the first phase, the *initialisation phase*, the data set to be outsourced is encrypted and outsourced. In the second phase, the *query phase*, the outsourced data set is queried. These two phases are depicted in Figure 5.1. In the initialisation phase, the data set d is encrypted by a preprocessor using an encryption key K. Then, the encryption $\mathsf{Enc}(d, K)$ is sent to the server. In the query phase, the client queries the encrypted data. In order to do so, the client runs an interactive protocol π_q with the server. The preprocessor and the client are not necessarily the same entities. The inputs of the client are the query q and the encryption key K. The input of the server is the encrypted data set $\mathsf{Enc}(d, K)$. At the end of the protocol, the client outputs the result of the query $q(d)$ and the server outputs the



5.1.1: initialisation phase



5.1.2: query phase

Figure 5.1.: The two phases in an outsourcing scheme. In the initialisation phase (5.1.1) a preprocessor receives and encrypts the data set d and uploads the encrypted data set $\mathsf{Enc}(d, K)$ to the server. In the query phase (5.1.2) the client executes queries q by running an interactive protocol π_q with the server. After the interaction, the client outputs the query result $q(d)$ and the server outputs the updated encrypted data set $\mathsf{Enc}(d_q, K)$. In the protocol for the next query, the input of the server is the updated encrypted data set $\mathsf{Enc}(d_q, K)$.

updated encrypted data set $\mathsf{Enc}(d_q, K)$. For subsequent queries, the updated encrypted data set is the input of the server.

Consequently and in order to be useful, a data outsourcing scheme allows the client to use the outsourced data set like a local data set. The interactive protocol π_q for a query q retrieves the result set $q(d)$. Additionally, the interactive protocol π_q updates the outsourced data set to $\mathsf{Enc}(d_q, K)$, while d_q is the database d after execution of query q. Note that, depending on the outsourcing scheme, some queries may only retrieve information from the outsourced data set but not update it ($d_q = d$). For example, queries of classical searchable encryption schemes only retrieve information from the encrypted data. In the following, we provide formal definitions of data outsourcing schemes.

Definition 36 (Data Outsourcing Scheme)**.** *An* outsourcing scheme *for data sets is a tuple* $(\mathsf{Gen}, \mathsf{Enc})$ *such that*

$$\mathsf{Gen} : 1^k \to \{0,1\}^n$$
$$\mathsf{Enc} : D \times \{0,1\}^n \to \Gamma$$

For an outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc})$ *and a data set* d*, we call* $\mathsf{Enc}(d, K)$ *the outsourced data set. We call an outsourcing scheme for a data set* retrievable *if there is a function* $\mathsf{Dec} : \Gamma \times \{0,1\}^n \to \Delta$ *such that* $\forall K \in \{0,1\}^n, d \in D : \mathsf{Dec}(\mathsf{Enc}(d, K), K) = d$.

This definition does not require encrypted data sets to be decryptable. The idea of outsourcing schemes is not to upload and download data sets as a whole, but to execute queries on outsourced data sets. Note that outsourcing schemes may use *internal* encryption schemes to realise their algorithms Gen, Enc, and Dec. In fact, in our database outsourcing scheme in Chapter 7, Section 4, we use an internal Ind-CPA secure encryption scheme. Also note that this definition uses the same key K for de- and encryption. The extension into an asymmetric scheme is straightforward.

Queries are used in order to insert information into or extract information from outsourced data sets. We define queries as follows:

Definition 37 (Query). *A* query $q : D \to P \times D$ *is a PPT algorithm that, on input of a data set* $d \in D$ *returns a* result set $q(d) \in P$ *and an updated data set* $d_q \in D$. *We denote the set of all queries with* $\mathbf{Q}$.

Based on this definition, we can distinguish between two types of queries. Queries that do not chance the data and queries that potentially do. In this work, we focus on relational databases and relational algebra or SQL as query language. Here, the domain of the result sets P is the domain of databases (DB). Furthermore, a query that potentially changes the database can add, remove or change tuples of d. In SQL, these queries are called *INSERT*, *UPDATE*, and *DELETE queries*. We will discuss queries in more detail in Chapter 7, Section 2.3. Without loss of generality we assume that queries are functions of the data, i.e. $\forall\, q\, \exists\, d_1, d_2 \in D : q(d_1) \neq q(d_2)$. Furthermore, the query result is the same if evaluated twice.

Given the definition of queries, we can define protocols that execute queries. Our idea of the *correctness* of a protocol is relative to a given query q. If running the two-party protocol on outsourced data has the same effect as if query q had been executed *before* outsourcing the data, we say the protocol executes query q:

Definition 38 (Protocol that Executes a Query on an Outsourced Data Set). *A two-party protocol* $\pi_q \in \Pi$ *between a server* $\mathfrak{S}$ *and client* $\mathfrak{C}$ executes a query q *for an outsourced data set* $\mathsf{Enc}(d, K)$ *if*

- *The client, on input of a key* K, *outputs the result set* $q(d) = \pi_q^{\mathfrak{C}}(K, \mathsf{Enc}(d, K))$.
- *The server, on input the outsourced data set* $\mathsf{Enc}(d, K)$, *outputs an updated outsourced data set* $\mathsf{Enc}(d_q, K) = \pi_q^{\mathfrak{S}}(\mathsf{Enc}(d, K))$.

The first property demands that the client gets the correct result: the result set to the query q for the database d. The second property demands that the server gets the correct result: the updated and encrypted data set. Please note that, since we focus on single server outsourcing, Definition 38 involves only two parties, namely a server and a client. The extension of this definition to multiple servers or even multiple clients is straightforward. Now, we can define data outsourcing schemes, that can execute queries.

Definition 39 (Queryable Outsourcing Scheme). *A* queryable outsourcing scheme *for data sets is a tuple* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ *such that*

- $(\mathsf{Gen}, \mathsf{Enc})$ *is an outsourcing scheme for data sets, and*
- $\mathsf{Q} \subseteq \Pi$ *is a non-empty set of efficient two-party protocols that execute a query for outsourced data sets.*

Note that the client has no direct access to the data when interacting with the server in order to execute a query (cf. Definition 38 and Figure 5.1).

Since the focus of this work is on database outsourcing, we need a definition of a *database outsourcing scheme*. A database outsourcing scheme is simply a specialisation of a queryable outsourcing scheme: a queryable outsourcing scheme that operates on databases.

Definition 40 (Database Outsourcing Scheme)**.** *A* database outsourcing scheme *is a queryable outsourcing scheme* (Gen, Enc, Q) *such that*

- *The input domain of* Enc *is* DB*, and*
- *each protocol in* Q *executes a query for databases.*

In oder to security privacy notions for data outsourcing schemes, we need a definition for what a party *sees* during an execution of a protocol. We call this the view of a party.

Definition 41 (View of a Protocol Party)**.** *A* view *of a protocol party is the totality of its inputs, received messages, sent messages and outputs. To denote the view of protocol party* $\mathfrak{P} \in \{\mathfrak{C}, \mathfrak{S}\}$ *in protocol* π *with inputs* c*, we write*

$$\mathsf{view}^{\pi}_{\mathfrak{P}}(c).$$

In particular, the encrypted data is part of the server's view.

In our game-based notions, we will give the adversary oracle access to certain views.

3.1. Basic Privacy Notions for Outsourced Data Sets

In this section, we define basic privacy notions for outsourced data sets. We differentiate between Static Security, where the adversary does not observe queries, and security where the adversary observes queries. In the latter case, we define notions for the three privacy goals *Data Privacy*, *Query Privacy*, and *Result Privacy*. Later in Section 4, we establish fundamental relations between these notions. In Section 6, based on the notions defined in this section, we introduce a framework for generalised notions for data outsourcing. This framework allows for defining application specific, relaxed security notions.

Please note that we only consider honest but curious adversaries. We do not consider active adversaries, that manipulate the outsourced data or the protocol. For a brief discussion of actively malicious adversaries, please refer to Chapter 7, Section 8.4.

As stated earlier, we define the notions for Static Security and security where the adversary observes queries as game-based notions. Therefore, we define a privacy game for each notion. The corresponding privacy notions holds, if the advantage of the adversary winning this game compared to mere guessing is negligible.

3.1.1. Static Security

In the game for Static Security, the adversary may chooses two data sets and gets the encryption of one of these data sets. Then, the adversary guesses which data set has been encrypted. the game is won, if the adversary guesses right. In this case, the result of the experiment is 1.

The static notion of privacy for outsourced data captures the intuition that no adversary may deduce any information about the data from its ciphertext alone (*indistinguishability under chosen-data attacks*).

Security Game 4 ($\mathsf{IND\text{-}CDA}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \mathsf{Gen}(1^k)$.
2. *The adversary* $\mathcal{A}$ *is given input* 1^k *and oracle access to* $\mathsf{Enc}(\cdot, K)$.

3. *$\mathcal{A}$ outputs two data sets d_0 and d_1 of equal length to the experiment.*
4. *The experiment chooses a random bit $b \leftarrow \{0, 1\}$.*
5. *$\mathcal{A}$ is given $\mathsf{Enc}(d_b, K)$.*
6. *$\mathcal{A}$ outputs a guess b' for b.*

The result of the experiment is 1 *if* $b' = b$ *and* 0 *else.*

We say an outsourcing scheme has Static Security if the adversary can not win the game defined above with a probability that is non negligibly greater than gussing (i. e. $\frac{1}{2}$).

Definition 42 (Static Security)**.** *An outsouring scheme* (Gen, Enc) *has* indistinguishable encryptions under chosen-data attacks IND-CDA *or* Static Security*, if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that:*

$$\Pr[\text{IND-CDA}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}(k) = 1] \leq \frac{1}{2} + negl(k)$$

Definition 47 is similar to the security notion IND-CPA for encryption schemes, but is defined on data sets rather than plaintexts (cf. Defintion 15).

3.1.2. Privacy in the Presence of Queries

In contrast to encrypted messages, which, in most cases, are en- and decrypted as a whole, outsourced data is intended to be queried. Therefore, we also have to consider security, when the adversary can observe the execution of queries. Then, there can be distinguished three conceptually different privacy goals for data outsourcing (cf. Section 2):

- keeping the data private
- keeping the queries private
- keeping the results private

We model these privacy goals with the means of three security games. The adversary is supplied with an oracle for views (cf. Definition 41) on the interaction between client and server and tries to learn a challenge bit.

In all three security games, the adversary is supplied with an *open* view oracle in addition to the challenge oracle. This open oracle provides views of the server for arbitrary queries executed on an encryption of arbitrary data sets *using the challenge key* without giving the adversary direct access to the challenge key. This oracle has two purposes for the adversary. On the one hand it serves as an encryption oracle since the server view contains the encrypted data. Consequently, the dynamic notions directly imply Static Security. Further, the open view oracle facilitates replay attacks for the adversary. She easily can check the challenge candidates against the open view oracle. This implies that the *probabilistic property* of Static Security expands to schemes with security in the presence of queries in the sense that an access pattern of a protocol executing a query does not reveal information about the data (Data Privacy), the query (Query Privacy), or the query result (Result Privacy).

For example, this probabilistic property can be achieved by accessing all of the outsourced data for each query. Nontrivial solutions can involve randomising the structure of the encrypted ciphertext (as in the work of Haynberg et al. [Hay+13]), randomising the protocol which realises the query, or even secure hardware.

Data Privacy In the Data Privacy experiment, the adversary tries to discern information about the data from the queries.

Security Game 5 ($\text{D-IND}^{\mathcal{A}}_{(\text{Gen},\text{Enc},\text{Q})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \text{Gen}(1^k)$.
2. *The adversary* $\mathcal{A}$ *is given input* 1^k*, receives access to an oracle for* $\text{view}^{\pi_\cdot}_{\Sigma}(\text{Enc}(\cdot, K))$*, and continues to have access to it. The oracle takes a query* q *and a data set* d *as input and returns* $\text{view}^{\pi_q}_{\Sigma}(\text{Enc}(d, K))$.
3. $\mathcal{A}$ *outputs two data sets* d_0 *and* d_1 *of equal length to the experiment.*
4. *The experiment draws a random bit* $b \leftarrow \{0, 1\}$.
5. *Challenge oracle:* $\mathcal{A}$ *is given access to an oracle for* $\text{view}^{\pi_\cdot}_{\Sigma}(\text{Enc}(d_b, K))$*. That is, the oracle takes any query* q *such that* $\pi_q \in Q$ *as input, internally runs the protocol* π_q *on* $\text{Enc}(d_b, K)$*, and outputs* $\text{view}^{\pi_q}_{\Sigma}(\text{Enc}(d_b, K))$ *to the adversary.*
6. $\mathcal{A}$ *outputs a guess* b' *for* b.

The result of the experiment is 1 *if* $b' = b$ *and* 0 *else.*

Definition 43 (Data Privacy)**.** *An outsourcing scheme* (Gen, Enc, Q) *has* Data Privacy*, if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $negl$ *such that:*

$$\Pr[\text{D-IND}^{\mathcal{A}}_{(\text{Gen},\text{Enc},\text{Q})}(k) = 1] \leq \frac{1}{2} + negl(k)$$

Query Privacy The notion of Query Privacy captures the goal of hiding the queries themselves from the adversary. This notion is equivalent to Private Information Retrieval (see Section 5 for a discussion and proof).

Security Game 6 ($\text{Q-IND}^{\mathcal{A}}_{(\text{Gen},\text{Enc},\text{Q})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \text{Gen}(1^k)$.
2. *The adversary* $\mathcal{A}$ *is given input* 1^k*, receives access to an oracle for* $\text{view}^{\pi_\cdot}_{\Sigma}(\text{Enc}(\cdot, K))$*, and continues to have access to it. The oracle takes a query* q *and a data set* d *as input and returns* $\text{view}^{\pi_q}_{\Sigma}(\text{Enc}(d, K))$.
3. $\mathcal{A}$ *outputs two queries* q_0 *and* q_1 *to the experiment.* q_0 *and* q_1 *must yield protocols* π_{q_0} *and* π_{q_1} *with the same number of protocol messages.*
4. *The experiment draws a random bit* $b \leftarrow \{0, 1\}$.
5. *Challenge oracle:* $\mathcal{A}$ *is given access to an oracle for* $\text{view}^{\pi_{q_b}}_{\Sigma}(\text{Enc}(\cdot, K))$*. That is, the oracle takes any data set* $d \in D$ *as input, internally runs the protocol* π_{q_b} *on* $\text{Enc}(d, K)$*, and outputs* $\text{view}^{\pi_{q_b}}_{\Sigma}(\text{Enc}(d, K))$ *to the adversary.*
6. $\mathcal{A}$ *outputs a guess* b' *for* b.

The result of the experiment is 1 *if* $b' = b$ *and* 0 *else.*

Definition 44 (Query Privacy)**.** *An outsouring scheme* (Gen, Enc, Q) *has* Query Privacy*, if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $negl$ *such that:*

$$\Pr[\text{Q-IND}^{\mathcal{A}}_{(\text{Gen},\text{Enc},\text{Q})}(k) = 1] \leq \frac{1}{2} + negl(k)$$

Result Privacy The third privacy goal, Result Privacy, captures the idea that the adversary must not learn the result of any query executed on any data. In order to formalise this idea, we allow the adversary to output two data-set-query-pairs (d_0, q_0) and (d_1, q_1). A query result is always determined by a query and a data set on which it is evaluated. Then, the adversary is challenged on the view of query q_b executed on data set d_b. If she cannot deduce b from the view, we say the scheme has result privacy.

Security Game 7 ($\mathsf{R\text{-}IND}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \mathsf{Gen}(1^k)$.
2. *The adversary* $\mathcal{A}$ *is given input* 1^k*, receives access to an oracle for* $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$*, and continues to have access to it. The oracle takes a query* q *and a data set* d *as input and returns* $\mathsf{view}^{\pi_q}_{\mathfrak{S}}(\mathsf{Enc}(d, K))$.
3. $\mathcal{A}$ *outputs two data-set-query pairs* (d_0, q_0) *and* (d_1, q_1) *to the experiment.* ($|d_0| = |d_1|$ *and* q_0 *and* q_1 *must yield protocols* π_{q_0} *and* π_{q_1} *with the same number of protocol messages.)*
4. *The experiment draws a random bit* $b \leftarrow \{0, 1\}$.
5. *Challenge:* $\mathcal{A}$ *is given access to the oracles for* $\mathsf{view}^{\pi_{q_b}}_{\mathfrak{S}}(\mathsf{Enc}(d_b, K))$, $\mathsf{view}^{\pi_{q_b}}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$, *and* $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(d_b, K))$ *and continues having access to them.*
6. $\mathcal{A}$ *outputs a guess* b' *for* b.

The result of the experiment is 1 *if* $b' = b$ *and* 0 *else.*

Definition 45 (Result Privacy)**.** *An outsourcing scheme* (Gen, Enc, Q) *has* Result Privacy*, if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function negl such that:*

$$\Pr[\mathsf{R\text{-}IND}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k) = 1] \leq \frac{1}{2} + \mathit{negl}(k)$$

Result Privacy implies Data Privacy and Query Privacy and vice versa. See Section 4 for a proof.

4. Fundamental Relations Among the Basic Privacy Notions

In this Section, we examine and establish fundamental relations among the three concepts of Data Privacy, Query Privacy, and Result Privacy. Concretely, our findings are:

- Data Privacy and Query Privacy are independent concepts.
- Result Privacy is equivalent to having Data Privacy and Query Privacy at the same time.

In order to show that Data Privacy and Query Privacy are independent concepts, we take a scheme that has Data Privacy and modify it in such a way that the resulting scheme still has Data Privacy but does not have Query Privacy and, in reverse, we modify a scheme that has Query Privacy which results in a scheme with Query Privacy but no Data Privacy.

Theorem 2 (D-IND $\not\Longrightarrow$ Q-IND)**.** *If a data outsourcing scheme that has Data Privacy exists, there is a data outsourcing scheme that has Data Privacy but no Query Privacy.*

Proof. Let (Gen, Enc, Q) be a data outsourcing scheme that has Data Privacy. We modify it in a way to violate Query Privacy, but keep Data Privacy intact. To this end, we amend the protocols that execute queries to have the client transmit the executed query in the clear after the actual protocol is complete. We have to show that the modification violates Query Privacy but does not violate Data Privacy.

With the modification, the adversary in experiment Q-IND can easily extract the executed query from any view and thus determine the challenge bit with certainty. Thus the modification violates Query Privacy. To see that this modification does not violate Data Privacy, first note that the modified scheme retains its Data Privacy up until the point of the modification. We argue that the transmission of the query in the clear does not break Data Privacy. Consider experiment D-IND. Because the experiment draws the key K after the scheme is fixed, the scheme is independent of the actual key used to encrypt the data set d. Further, because the query is supplied by the adversary in the experiment and the adversary has learned neither d_b nor K up to this point, the query is also independent of d_b and K. This concludes the argument. □

Theorem 3 (Q-IND $\not\Longrightarrow$ D-IND)**.** *If there is a retrievable data outsourcing scheme that has Static Security, there is a data outsourcing scheme that has Query Privacy and Static Security, but no Data Privacy.*

For this proof, we change each protocol of the given outsourcing scheme. Each changed protocol transfers the encryption key K to the server, retrieves the outsourced data set, and executes the query locally. Please note that Theorem 3 also separates Static Security from Data Privacy.

Proof. Let (Gen, Enc, Q) be a retrievable data outsourcing scheme that has Static Security. We construct a modified scheme (Gen, Enc, Q′) that suits our purposes. By adopting Gen and Enc, we retain static security. We design Q' such that it has query privacy, but trivially loses Data Privacy. Q′ is constructed iteratively, starting with an empty set. For each protocol $\pi_q \in \mathsf{Q}$ that realises a query q, we define a protocol $\pi'_q \in \mathsf{Q}'$ as follows:

(Recall that the client's input is the encryption key K and a query q; the server's input is an encrypted data set $\mathsf{Enc}(d, K)$.)

1. Client: Transfer K to the Server.
2. Server: Decrypt $\mathsf{Enc}(d, K)$ and send $d = \mathsf{Dec}(\mathsf{Enc}(d, K), K)$ back to the Client.
3. Client: Execute query q locally on d and output $q(d)$.

Protocol π' transmits the data set d in the open, violating Data Privacy. Because the client executes q locally and never transmits any information that depends on q, the scheme (Gen, Enc, Q′) does have Query Privacy. □

The following theorems show that Result Privacy is equivalent to both Data Privacy and Query Privacy (at the same time).

Theorem 4 (R-IND $\Longrightarrow$ D-IND)**.** *There is no data outsourcing scheme that has Result Privacy but no Data Privacy.*

Proof. Assume a data outsourcing scheme (Gen, Enc, Q) for which there is an efficient adversary $\mathcal{A}_D$ against experiment D-IND. We give an efficient reduction for $\mathcal{A}_D$ that breaks the Result Privacy (experiment R-IND) of the scheme, contradicting the assumption. The reduction is straightforward. It has to provide a challenge oracle $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_b, K))$.

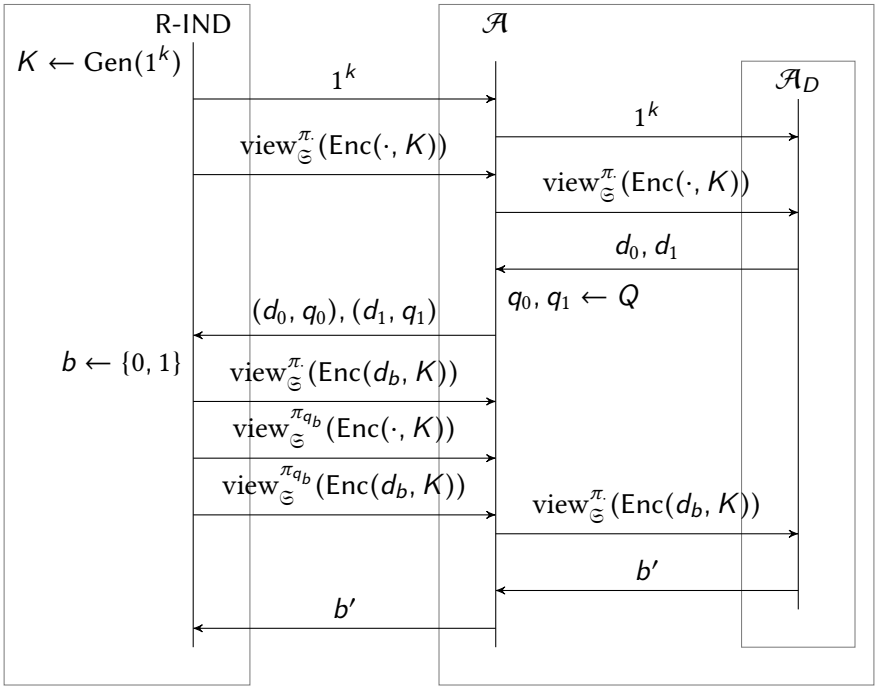


Figure 5.2.: Sketch for the proof of Theorem 4: An efficient reduction of an adversary $\mathcal{A}_D$ that breaks Data Privacy to an adversary $\mathcal{A}$, that breaks Result Privacy.

Such an oracle is provided by experiment R-IND and only has to be *passed through* (see Figure 5.2). □

Theorem 5 (R-IND $\implies$ Q-IND)**.** *There is no data outsourcing scheme that has Result Privacy but not Query Privacy.*

Proof. The proof of Theorem 5 is analogous to the proof of Theorem 4. Instead of passing through the oracle for $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_b, K))$, now, the reduction $\mathcal{A}$ passes the oracle $\text{view}_{\mathfrak{S}}^{\pi_{q_b}}(\text{Enc}(\cdot, K))$ to the adversary $\mathcal{A}_Q$ (see Figure 5.3). □

Theorem 6 (D-IND $\wedge$ Q-IND $\implies$ R-IND)**.** *Data Privacy and Query Privacy together imply Result Privacy: There is no data outsourcing scheme that has Data Privacy and Query Privacy but no Result Privacy.*

We prove the statement using a game-hopping technique. Assume any adversary against R-IND. We replace both view oracles for d_b and q_b with an oracle for fixed challenges d_0 and q_0, respectively. We argue the indistinguishability of these steps with Data Privacy and Query Privacy. Finally, in the now-transformed experiment, the adversary has no advantage since her input is independent of b. Concluding, given a scheme with Data Privacy and Query Privacy, no adversary against result privacy has a non-negligible advantage in the game R-IND.

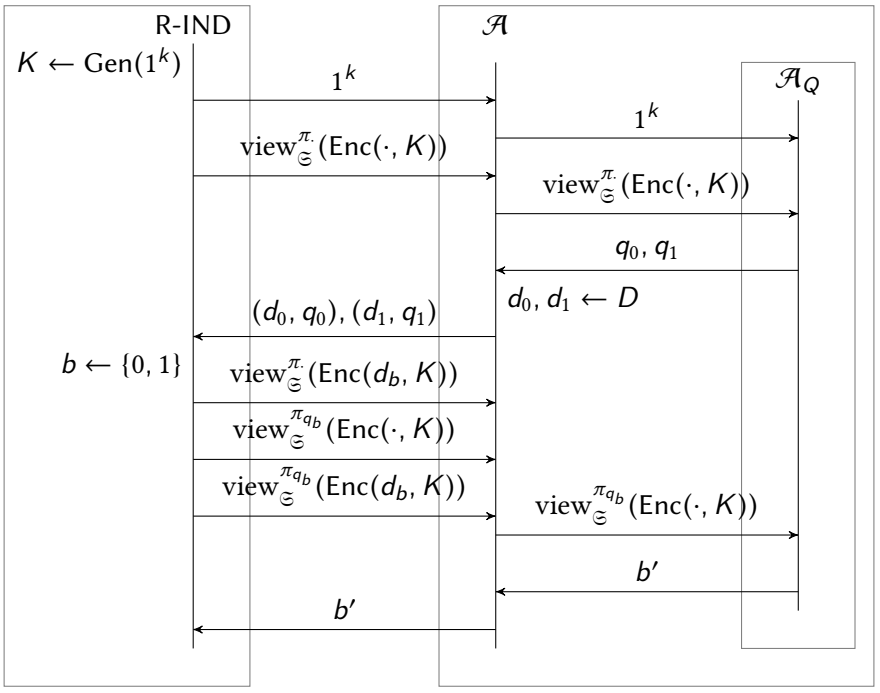


Figure 5.3.: Sketch of the proof for Theorem 5: An efficient reduction of an adversary $\mathcal{A}_Q$ that breaks Query Privacy to an adversary $\mathcal{A}$, that breaks Result Privacy.

Proof. Starting from the result privacy experiment R-IND, we define two game transformations: R-IND′ and R-IND″. In the unmodified experiment R-IND, the adversary is supplied with the following view oracles:

- $\mathrm{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathrm{Enc}(d_b, K))$
- $\mathrm{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathrm{Enc}(\cdot, K))$
- $\mathrm{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathrm{Enc}(d_b, K))$

In the following two steps, we set the bit b in the two view oracles to 0.

In the modification R-IND′ (cf. Figure 5.4), we replace the oracle $\mathrm{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathrm{Enc}(d_b, K))$ with an oracle for $\mathrm{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathrm{Enc}(d_0, K))$ and the oracle $\mathrm{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathrm{Enc}(d_b, K))$ with $\mathrm{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathrm{Enc}(d_0, K))$. This modification is indistinguishable for the adversary. If she could distinguish R-IND from R-IND′, she could also break Data Privacy which is impossible by our assumptions. We prove the indistinguishability of R-IND from R-IND′ in Lemma 3.

In the game R-IND″ (cf. Figure 5.5), we replace $\mathrm{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathrm{Enc}(\cdot, K))$ by $\mathrm{view}_{\mathfrak{S}}^{\pi_{q_0}}(\mathrm{Enc}(\cdot, K))$ and further $\mathrm{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathrm{Enc}(d_0, K))$ by $\mathrm{view}_{\mathfrak{S}}^{\pi_{q_0}}(\mathrm{Enc}(d_0, K))$. In R-IND″ the adversary receives no input that is dependent on the challenge bit b. Consequently, the adversary in R-IND″ can not have an advantage over guessing b. This modification is also indistinguishable for the adversary. If she could distinguish R-IND′ from R-IND″, she could also break Query Privacy, which is impossible by assumption. She thus has no advantage over guessing b. We have to argue that R-IND″ is indistinguishable from R-IND′ for the adversary. We prove the indistinguishability of R-IND′ from R-IND″ in Lemma 4. □

Lemma 3. *An adversary who can distinguish between running in experiment* R-IND *and experiment* R-IND′ *yields a successful adversary against Data Privacy.*

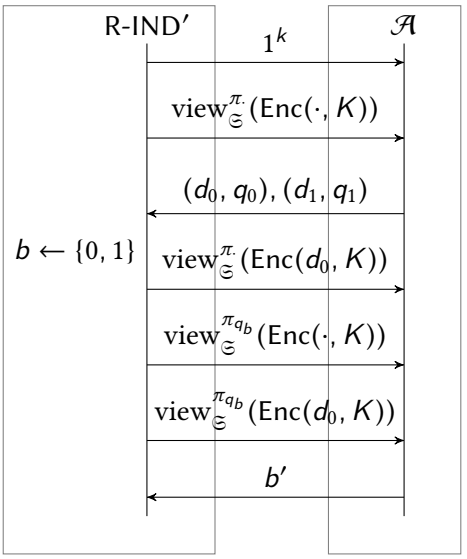


Figure 5.4.: The experiment R-IND′. In contrast to the experiment R-IND, the adversary gets the oracles for $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_0, K))$ and $\text{view}_{\mathfrak{S}}^{\pi_{q_b}}(\text{Enc}(d_0, K))$.

We model the distinction between the two R-IND and R-IND′ with an experiment D-Oracle-IND in which the adversary must decide whether she is running in R-IND or in R-IND′ (see Figure 5.6). The difference between the experiments R-IND and R-IND′ is whether the first challenge oracle is $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_b, K))$ or $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_0, K))$.

Thus, in D-Oracle-IND the adversary is challenged on deciding whether she has access to an oracle $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_b, K))$ or whether she is accessing oracle $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_0, K))$, where the bit b is set to 0. This is decided by a random challenge bit c, which is drawn by the experiment D-Oracle-IND and has to be guessed by the adversary. (The query view oracle $\text{view}_{\mathfrak{S}}^{\pi_{q_b}}(\text{Enc}(\cdot, K))$ is provided to the adversary with no change.) To clearly separate the different challenge bits, we name the challenge bit in the distinction experiment D-Oracle-IND c.

Security Game 8 ($\text{D-Oracle-IND}^{\mathcal{A}}_{(\text{Gen},\text{Enc},\text{Q})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \text{Gen}(1^k)$.
2. $\mathcal{A}$ *receives oracle access to* $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(\cdot, K))$.
3. $\mathcal{A}$ *outputs two data-set-query pairs* (d_0, q_0) *and* (d_1, q_1) *to the experiment (under the restrictions that* $|d_0| = |d_1|$ *and that* π_{q_0} *and* π_{q_1} *have the same number of messages).*
4. *The experiment draws two random bits* $b \leftarrow \{0,1\}$ *and* $c \leftarrow \{0,1\}$.
5. *Challenge:*
   If $c = 0$*:* $\mathcal{A}$ *oracle receives access to* $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_0, K))$.
   If $c = 1$*:* $\mathcal{A}$ *oracle receives access to* $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_b, K))$.
6. $\mathcal{A}$ *receives access to* $\text{view}_{\mathfrak{S}}^{\pi_{q_b}}(\text{Enc}(\cdot, K))$.
7. *If* $c = 0$*:* $\mathcal{A}$ *oracle receives access to* $\text{view}_{\mathfrak{S}}^{\pi_{q_b}}(\text{Enc}(d_0, K))$.
   If $c = 0$*:* $\mathcal{A}$ *oracle receives access to* $\text{view}_{\mathfrak{S}}^{\pi_{q_b}}(\text{Enc}(d_b, K))$.

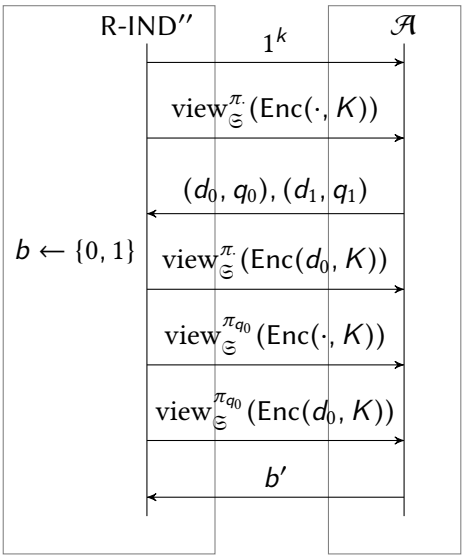


Figure 5.5.: The experiment R-IND''. In contrast to the experiment R-IND', the adversary gets the oracles for $\text{view}_{\mathfrak{S}}^{\pi_{q_0}}(\text{Enc}(\cdot, K))$ and $\text{view}_{\mathfrak{S}}^{\pi_{q_0}}(\text{Enc}(d_0, K))$.

8. *$\mathcal{A}$ outputs a guess c' for c.*

In order to prove Lemma 3, we give an efficient reduction $\mathcal{A}$ which transforms an adversary $\mathcal{A}_{\text{D-Oracle-IND}}$ that has an advantage over guessing c in this experiment into an adversary on Data Privacy.

Proof. Assume an adversary $\mathcal{A}_{\text{D-Oracle-IND}}$ with a non-negligible advantage in experiment D-Oracle-IND. We construct a reduction that has a non-negligible advantage in experiment D-IND (also see Figure 5.6). We need to simulate the following oracles:

- Step 2: $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(\cdot, K))$.
  This oracle is provided by the experiment D-IND and can be relayed.
- Step 5: $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_b, K))$.
  This oracle is provided by the experiment D-IND and can be relayed.
- Step 6: $\text{view}_{\mathfrak{S}}^{\pi_{q_b}}(\text{Enc}(\cdot, K))$.
  From the experiment D-IND, we get the oracle $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(\cdot, K))$. We fix q_1 and output $\text{view}_{\mathfrak{S}}^{\pi_{q_1}}(\text{Enc}(\cdot, K))$.
- Step 7: $\text{view}_{\mathfrak{S}}^{\pi_{q_b}}(\text{Enc}(d_b, K))$.
  From the experiment D-IND, we get the oracle $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_b, K))$. We fix q_1 and output $\text{view}_{\mathfrak{S}}^{\pi_{q_1}}(\text{Enc}(d_b, K))$.

Now, we can distinguish two cases for the challenge b in the experiment D-IND:

1. $b = 0$:
   We forwarded the oracles
   - $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(\cdot, K))$,
   - $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\text{Enc}(d_0, K))$,

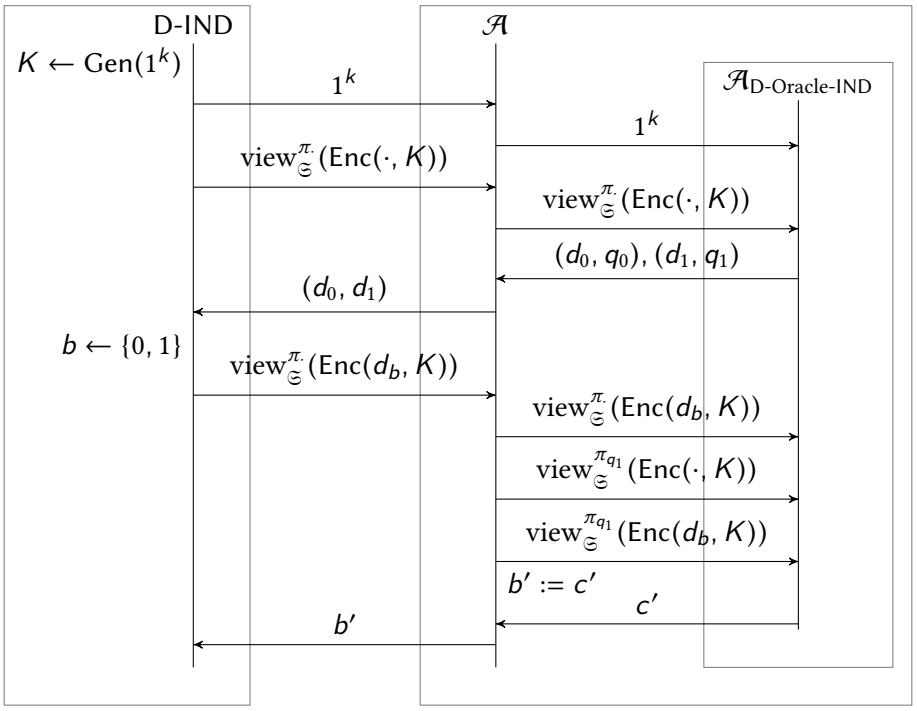


Figure 5.6.: Sketch of the proof for Lemma 3. An efficient adversary that can decide whether she is running in experiment R-IND or R-IND' yields an efficient adversary against D-IND.

- $\text{view}_{\mathfrak{S}}^{\pi_{q_1}}(\mathsf{Enc}(\cdot, K))$, and
- $\text{view}_{\mathfrak{S}}^{\pi_{q_1}}(\mathsf{Enc}(d_0, K))$.

The views are inconsistent. We simulated D-Oracle-IND with $b = 1$ and $c = 0$.

2. $b = 1$:
   We forwarded the oracles
   - $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$,
   - $\text{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_1, K))$,
   - $\text{view}_{\mathfrak{S}}^{\pi_{q_1}}(\mathsf{Enc}(\cdot, K))$,
   - and $\text{view}_{\mathfrak{S}}^{\pi_{q_1}}(\mathsf{Enc}(d_1, K))$.

   We simulated D-Oracle-IND with $b = 1$ and $c = 1$.

Please note that if in the experiment D-Oracle-IND the random bit b is 0, the challenge is independent of c. Therefore, the advantage from the adversary over guessing c has to be from the cases above, where $b = 1$. Thus, we return $\mathcal{A}_{\text{D-Oracle-IND}}$'s guess c' as our guess b' to inherit $\mathcal{A}_{\text{D-Oracle-IND}}$'s success probability. □

Lemma 4. *An adversary who can distinguish between* R-IND′ *and* R-IND″ *yields a successful adversary on Query Privacy.*

The proof of Lemma 4 is analogous to that of Lemma 3: We define a distinction experiment $\mathcal{A}_{\text{Q-Oracle-IND}}$ in which the adversary has to decide whether she is running in the experiment R-IND′ or in the experiment R-IND″.

Security Game 9 ($\text{Q-Oracle-IND}^{\mathcal{A}}_{(\text{Gen},\text{Enc},\text{Q})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \text{Gen}(1^k)$.
2. $\mathcal{A}$ *receives oracle access to for* $\text{view}^{\pi_\cdot}_{\mathfrak{S}}(\text{Enc}(\cdot, K))$.
3. $\mathcal{A}$ *outputs two data-set-query pairs* (d_0, q_0) *and* (d_1, q_1) *to the experiment (under the restrictions that* $|d_0| = |d_1|$ *and that* π_{q_0} *and* π_{q_1} *have the same number of messages).*
4. *The experiment draws two random bits* $b \leftarrow \{0,1\}$ *and* $c \leftarrow \{0,1\}$.
5. $\mathcal{A}$ *receives oracle access to* $\text{view}^{\pi_\cdot}_{\mathfrak{S}}(\text{Enc}(d_0, K))$.
6. *Challenge:*
   If $c = 0$*:* $\mathcal{A}$ *receives oracle access to* $\text{view}^{\pi_{q_0}}_{\mathfrak{S}}(\text{Enc}(\cdot, K))$.
   If $c = 1$*:* $\mathcal{A}$ *receives oracle access to* $\text{view}^{\pi_{q_b}}_{\mathfrak{S}}(\text{Enc}(\cdot, K))$
7. *If* $c = 0$*:* $\mathcal{A}$ *receives oracle access to* $\text{view}^{\pi_{q_0}}_{\mathfrak{S}}(\text{Enc}(d_0, K))$.
   If $c = 0$*:* $\mathcal{A}$ *receives oracle access to* $\text{view}^{\pi_{q_b}}_{\mathfrak{S}}(\text{Enc}(d_0, K))$.
8. $\mathcal{A}$ *outputs a guess* c' *for* c.

Proof. Assume an adversary $\mathcal{A}_{\text{Q-Oracle-IND}}$ with a non-negligible advantage in experiment Q-Oracle-IND. We construct a reduction that has a non-negligible advantage in experiment D-IND (also see Figure 5.7). We need to simulate the following oracles:

- Step 2: $\text{view}^{\pi_\cdot}_{\mathfrak{S}}(\text{Enc}(\cdot, K))$.
  This oracle is provided by the experiment Q-IND and can be relayed.
- Step 5: $\text{view}^{\pi_\cdot}_{\mathfrak{S}}(\text{Enc}(d_0, K))$.
  To simulate this oracle we use the $\text{view}^{\pi_\cdot}_{\mathfrak{S}}(\text{Enc}(\cdot, K))$ oracle provided by the experiment Q-IND and fix d_0.
- Step 6: $\text{view}^{\pi_{q_b}}_{\mathfrak{S}}(\text{Enc}(\cdot, K))$.
  This oracle is provided by the experiment Q-IND and can be relayed.
- Step 7: $\text{view}^{\pi_{q_b}}_{\mathfrak{S}}(\text{Enc}(d_0, K))$.
  From the experiment Q-IND, we get the oracle $\text{view}^{\pi_{q_b}}_{\mathfrak{S}}(\text{Enc}(\cdot, K))$. We fix d_0 and output $\text{view}^{\pi_{q_b}}_{\mathfrak{S}}(\text{Enc}(d_0, K))$.

Now, we can distinguish two cases for the challenge b in the experiment D-IND:

1. $b = 0$:
   We forwarded the oracles
   - $\text{view}^{\pi_\cdot}_{\mathfrak{S}}(\text{Enc}(\cdot, K))$,
   - $\text{view}^{\pi_\cdot}_{\mathfrak{S}}(\text{Enc}(d_0, K))$,
   - $\text{view}^{\pi_{q_0}}_{\mathfrak{S}}(\text{Enc}(\cdot, K))$, and
   - $\text{view}^{\pi_{q_0}}_{\mathfrak{S}}(\text{Enc}(d_0, K))$.

   The challenge is consistent. We simulated Q-Oracle-IND with $c = 0$.

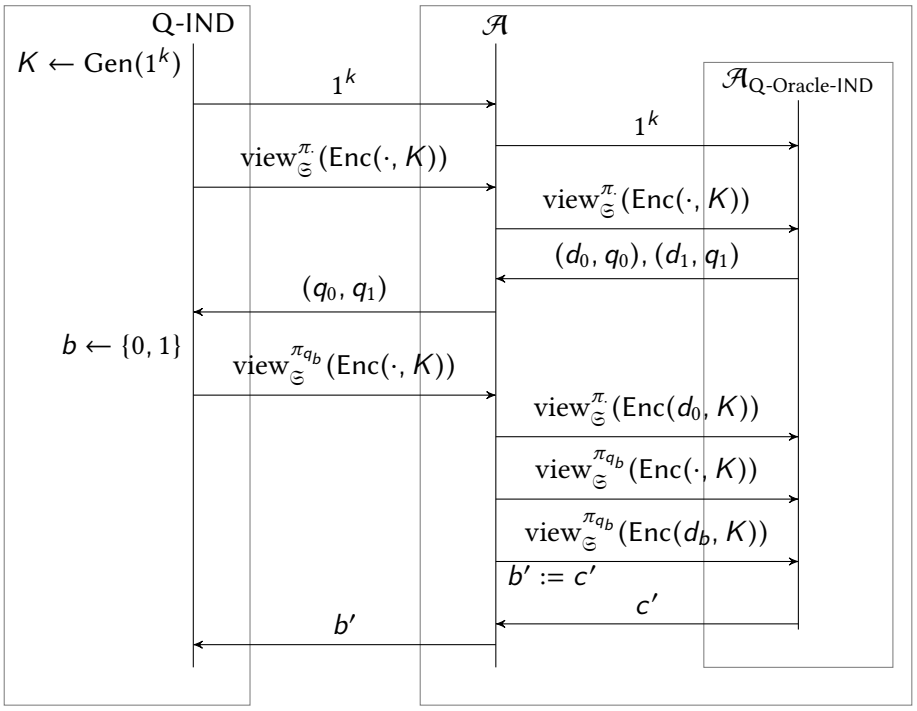


Figure 5.7.: Sketch of the proof for Lemma 4. An efficient adversary that can decide whether she is running in experiment R-IND' or R-IND" yields an efficient adversary against Q-IND.

2. $b = 1$:
   We forwarded the oracles
   - $\mathrm{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathrm{Enc}(\cdot, K))$,
   - $\mathrm{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathrm{Enc}(d_0, K))$,
   - $\mathrm{view}_{\mathfrak{S}}^{\pi_{q_1}}(\mathrm{Enc}(\cdot, K))$, and
   - $\mathrm{view}_{\mathfrak{S}}^{\pi_{q_1}}(\mathrm{Enc}(d_0, K))$.

   The challenge is inconsistent. We simulated Q-Oracle-IND with $c = 1$ and $b = 1$. Please note that for $b = 1$ and for $c = 0$, the adversary would receive the same oracles as for $b = 0$. Therefore, we only have to consider the case $c = 1$, here.

Thus, we return $\mathcal{A}_{\text{Q-Oracle-IND}}$'s guess as the reduction's own guess $b' = c'$ to inherit $\mathcal{A}_{\text{Q-Oracle-IND}}$'s success probability. □

The Theorems 4, 5 and 6 lead to the following corollary:

Corollary 1 (R-IND $\iff$ D-IND $\wedge$ Q-IND)**.** *Result Privacy is equivalent to both, Data Privacy and Query Privacy (at the same time).*

5. Query Privacy and Private Information Retrieval

In the original definition of (single server) Computational Private Information Retrieval (cPIR) [KO97], the notion is defined for a pair of polynomial time Turing machines, a *database*, and a *user*. The user queries the database once. The database machine has a

n-bit input string on its input tape, reads the query from its read only communication tape, and writes a reply to its write only communication tape. The user finally outputs a bit. We give a definition of the cPIR notion using our notation and conventions.

Definition 46 (Private Information Retrieval)**.** *A queryable outsourcing scheme* (Gen, Enc, Q) *exhibits* Computational Single Server Private Information Retrieval *when for any $n \in \mathbb{N}$, any security parameter $k \in \mathbb{N}$, and any data set d over $\Sigma = \{0,1\}^n$ the following two conditions hold:*

1. *Correctness:* $\forall\, i \in \{0, \dots, n-1\}\, \exists\, \pi_i \in Q : \pi_i^{\mathfrak{C}}(d) = d[i]$.
2. *Privacy:* $\forall c \in \mathbb{N}, i, j \in \{0, \dots, n-1\}, \forall\, \mathcal{A}\, \exists\, K \in \mathbb{N}$ *such that* $\forall k > K$:

$$|\Pr[\mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_i}(\mathsf{Enc}(d,K))) = 1] - \Pr[\mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_j}(\mathsf{Enc}(d,K))) = 1]| < \frac{1}{\max(k,n)^c}$$

Theorem 7. *Private Information Retrieval is equivalent to Query Privacy.*

In our proof, we implicitly exclude schemes that store unretrievable information in the data set. We assume that each queryable outsourcing scheme has, for each bit in its data set, a query that retrieves it. This is not a restriction, as one can easily construct a *non-redundant* scheme from one that stores unretrievable information.

Proof. For this proof, let the domain of all data sets be $\Delta = \{0,1\}^*$. We fix a security parameter $k \in \mathbb{N}$. W.l.o.g., we assume any queryable outsourcing scheme (Gen, Enc, Q) with a protocol π_i that outputs the $i+1$th bit of the data set for all $i \in \{0, \dots, n-1\}$, where n is the length of the data set $d \in \Delta$. We prove the theorem in two steps.

PIR $\implies$ Q-IND:
Assume any efficient adversary $\mathcal{A}$ who is successful in the experiment Q-IND with a non-negligible advantage over guessing. We show that there are $i, j \in \{0, \dots, n-1\}$, and an efficient algorithm $\mathcal{A}'$ such that they violate the privacy condition of Definition 46.

Construct $\mathcal{A}'$ as follows: Simulate experiment Q-IND to obtain π_i, π_j from $\mathcal{A}$. i and j are the required indices. Now relay the input $\mathsf{view}_{\mathfrak{S}}^{\pi_b}(\mathsf{Enc}(d,K))$ (for $b \in \{i,j\}$) to $\mathcal{A}$. Output $\mathcal{A}$'s guess b'.

Q-IND $\implies$ PIR:
Assume any $i, j \in \{0, \dots, n-1\}$ and any efficient algorithm $\mathcal{A}'$ such that $\mathcal{A}'$ violates the privacy condition of Definition 46 at indices i and j. We construct an efficient adversary $\mathcal{A}$ that has a non-negligible advantage over guessing in the experiment Q-IND: Output π_i and π_j as the challenge queries. Output $b' = \mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_b}(\mathsf{Enc}(d,K)))$ (for $b \in \{i,j\}$). (For any adversary with a success probability $< \frac{1}{2}$ there is an adversary with a success probability $> \frac{1}{2}$, easily obtained by flipping its guess.) □

6. Generalised Security Notions for Data Outsourcing Schemes

In this section, we generalise the security notions of Data Privacy, Query Privacy, and Result Privacy. This allows for restricting the adversary's power. We use two generalisations on our notions from Section 3.1:

- *bounds* that limit the number of oracle calls the adversary is allowed to
- *leakage relations* that limit the adversaries choices for data sets and queries

A special case for a bound is 1 which renders the notions non-adaptive. Further, we explicitly model the issuing of queries independently of handing out the results. This allows us to capture security notions where the adversary can alter the state of the database, but can not see the immediate result (e. g. she can only observe the result of the last issued query).

In Section 5, we showed that Query Privacy is equivalent to private information retrieval. As it is notoriously hard to realise practical (single server) PIR protocols [SC07] a natural consequence is to investigate weaker security notions with an eye on practical feasibility. Similarly, protocols that, in order to execute queries efficiently, base decisions on the content of the queried data leak information about the data.

In order to model potential information leakage about the data and the queries (also see Section 2.4), we introduce the *leakage relations* R_d and R_q. Challenges the adversary can choose are subject to equivalence under these relations. This way, one can explicitly rule out specific distinction advantages. To model the leakage of the length of a data set for example, one would define $R_d \subset D^2$ as the set of all data set pairs with equal length.

Goh et al. [Goh03] introduce restricting parameters into their security notion as well. They allow for a bound on the running time, the advantage, and the number of oracle calls. Our approach is similar to these concepts but introduces further generalisations.

In the following, we define the generalised security notions for Static Security, Data Privacy, Query Privacy, and Result Privacy. In Section 6, we use the generalised notion of Static Security in order to instantiate Ind-ICP, our notion for secure database outsourcing and we use the generalised notion of Data Privacy to instantiate a generalisation of Ind-ICP that holds in the presence of queries.

Since, in the static case, the adversary has no access to a query oracle and does not issue queries, only the introduction of the leakage relation R_d is meaningful.

Security Game 10 ($\mathsf{IND\text{-}CDA}^{\mathcal{A},R_d}_{(\mathsf{Gen},\mathsf{Enc})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \mathsf{Gen}(1^k)$ *and a random bit* $b \leftarrow \{0,1\}$.
2. *The adversary* $\mathcal{A}$ *is given input* 1^k *and oracle access to* $\mathsf{Enc}(\cdot, K)$.
3. $\mathcal{A}$ *outputs two data sets* d_0 *and* d_1 *to the experiment. The choice of* d_0 *and* d_1 *is restricted to data set pairs that are equivalent with regard to equivalence relation* $R_d \subseteq D^2$*, i. e.* $(d_0, d_1) \in R_d$.
4. $\mathcal{A}$ *is given* $\mathsf{Enc}(d_b, K)$.
5. $\mathcal{A}$ *outputs a guess* b' *for* b.

The result of the experiment is 1 *if* $b' = b$ *and* 0 *else.*

Definition 47 (Static Security)**.** *An outsourcing scheme* (Gen, Enc) *has* indistinguishable encryptions under chosen-data-attacks (Ind-CDA) *or* Static Security *with respect to* R_d*, if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function negl such that:*

$$\Pr[\mathsf{IND\text{-}CDA}^{\mathcal{A},R_d}_{(\mathsf{Gen},\mathsf{Enc})}(k) = 1] \leq \frac{1}{2} + negl(k)$$

The generalisations for the notions Data Privacy, Query Privacy, and Result Privacy are straightforward. We introduce the bounds n_1, n_2, and n_3 that limit the number of times, the adversary can access the oracles given to her in each security game. The bound n_1 limits the times, the adversary can access the oracle $\mathsf{view}_{\mathcal{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$. This allows to model security notions, where the adversary is limited in what she is allowed to learn about outputs of the scheme. For example setting the bound n_1 to zero allows to model security notions for deterministic schemes. The bound n_2 limits the times, the adversary can access the challenge oracle. The bound n_3 limits the times, the adversary can access a run oracle, that has no output but can be used to simulate query executions. This run oracle serves the purpose to capture hypothetical schemes that, for example upload the encryption key after $n_2 + t$ (with $t < n_3$) queries. For Query Privacy and Result Privacy, analogous to Static Security, we introduce R_q a leakage relation of queries, that restricts the adversary.

Security Game 11 ($\text{D-IND}^{\mathcal{A},R_d,n_1,n_2,n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \mathsf{Gen}(1^k)$.
2. $\mathcal{A}$ *receives access to an oracle for* $\mathsf{view}_{\mathcal{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$*, and continues to have access to it.* $\mathcal{A}$ *is only allowed to query* $\mathsf{view}_{\mathcal{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$ *for a total number of* n_1 *times.*
3. $\mathcal{A}$ *outputs two data sets* d_0 *and* d_1 *to the experiment. The choice of* d_0 *and* d_1 *is restricted to data set pairs that are equivalent with regard to equivalence relation* $R_d \subseteq D^2$*, i. e.* $(d_0, d_1) \in R_d$.
4. *The experiment draws a random bit* $b \leftarrow \{0, 1\}$.
5. *Challenge:* $\mathcal{A}$ *is given access to an oracle for* $\mathsf{view}_{\mathcal{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$*, and continues to have access to it.* $\mathcal{A}$ *may call the challenge oracle for a total number of* n_2 *times.*
6. *Run oracle:* $\mathcal{A}$ *is given access to an oracle* $\mathsf{run}_{\mathcal{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$*. The run oracles execute queries just as the view oracle does, but has no output.* $\mathcal{A}$ *is allowed to call the run oracle for a total number of* n_3 *times.*
7. $\mathcal{A}$ *outputs a guess* b' *for* b.

The result of the experiment is 1 *if* $b' = b$ *and* 0 *else.*

Please note that if an adversary is given access to an oracle, she continues having access to it until to the end of the security game. For example, the adversary can call the oracle for $\mathsf{view}_{\mathcal{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$ after calling the oracle $\mathsf{run}_{\mathcal{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$ as long as she does not exceed the number of calls allowed by the respective bounds.

Definition 48 (n_1, n_2, n_3-Data Privacy)**.** *An outsourcing scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ *has* n_1, n_2, n_3-Data Privacy *with respect to* R_d*, if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function negl such that:*

$$\Pr[\text{D-IND}^{\mathcal{A},R_d,n_1,n_2,n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k) = 1] \leq \frac{1}{2} + negl(k)$$

Security Game 12 ($\text{Q-IND}^{\mathcal{A},R_q,n_1,n_2,n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \mathsf{Gen}(1^k)$.

2. *$\mathcal{A}$ receives access to an oracle for* $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$*, and continues to have access to it. $\mathcal{A}$ is only allowed to query* $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$ *for a total number of n_1 times.*
3. *$\mathcal{A}$ outputs two queries q_0 and q_1 to the experiment. The choice of q_0 and q_1 is restricted to query pairs that are equivalent with regard to equivalence relation $R_q \subseteq \Pi^2$, i. e. $(q_0, q_1) \in R_q$.*
4. *The experiment draws a random bit $b \leftarrow \{0, 1\}$.*
5. *Challenge: $\mathcal{A}$ is given access to an oracle for* $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$*. $\mathcal{A}$ may call the challenge oracle for a total number of n_2 times.*
6. *Run oracle: $\mathcal{A}$ is given access to an oracle* $\mathsf{run}_{\mathfrak{S}}^{\pi_b}(\mathsf{Enc}(\cdot, K))$*, and continues to have access to it. The run oracle executes queries just as the view oracle does, but has no output. $\mathcal{A}$ is allowed to call the run oracle for a total number of n_3 times.*
7. *$\mathcal{A}$ outputs a guess b' for b.*

The result of the experiment is 1 *if $b' = b$ and* 0 *else.*

Definition 49 (n_1, n_2, n_3-Query Privacy)**.** *An outsouring scheme* (Gen, Enc, Q) *has* n_1, n_2, n_3-Query Privacy *with respect to R_q, if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that:*

$$\Pr[\text{Q-IND}^{\mathcal{A}, R_q, n_1, n_2, n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k) = 1] \leq \frac{1}{2} + negl(k)$$

Security Game 13 ($\text{R-IND}^{\mathcal{A}, R_d, R_q, n_1, n_2, n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. *The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.*
2. *$\mathcal{A}$ receives access to an oracle for* $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$*, and continues to have access to it. $\mathcal{A}$ is only allowed to query* $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$ *for a total number of n_1 times.*
3. *$\mathcal{A}$ outputs two data-set-query pairs (d_0, q_0) and (d_1, q_1) to the experiment. The choice of d_0, d_1, q_0, and q_1 is restricted to $(d_0, d_1) \in R_d$ and $(q_0, q_1) \in R_q$.*
4. *The experiment draws a random bit $b \leftarrow \{0, 1\}$.*
5. *Challenge: $\mathcal{A}$ is given access to the oracles for* $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(d_b, K))$*,* $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$*, and* $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$ *and continues to having acces to them. $\mathcal{A}$ may call these oracles for a total number of n_2 times.*
6. *Run oracle: $\mathcal{A}$ is given access to the oracles* $\mathsf{run}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$*, and* $\mathsf{run}_{\mathfrak{S}}^{\pi_b}(\mathsf{Enc}(\cdot, K))$*, and continues having access to them. The run oracle executes queries just as the view oracle does, but has no output. $\mathcal{A}$ is allowed to call the run oracles for a total number of n_3 times.*
7. *$\mathcal{A}$ outputs a guess b' for b.*

The result of the experiment is 1 *if $b' = b$ and* 0 *else.*

Definition 50 (n_1, n_2, n_3-Result Privacy)**.** *An outsourcing scheme* (Gen, Enc, Q) *has* n_1, n_2, n_3-Result Privacy *with respect to R_d and R_q, if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that:*

$$\Pr[\text{R-IND}^{\mathcal{A}, R_d, R_q, n_1, n_2, n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k) = 1] \leq \frac{1}{2} + negl(k)$$

6. Security Notions for Database Outsourcing

This chapter is partially based on work already published in [AGH11; Hub+13; Hei+10; Hub+11; HMN13; HMN14; HH14; Har11] and [Ach+16].

1. Introduction

In order to be efficient, a database outsourcing scheme needs to support queries with a sublinear search time with regards to the size of the database. As of today, there are no known efficient (single server) database outsourcing schemes that fulfil a strong, classical security notion such as IND-CPA. If the data is stored on the server, efficient schemes have to execute at least parts of the queries on the server. Application specific, weaker security notions leak information about the data to the server. This information, then, can be exploited to build efficient schemes.

This is one of the main difference between classical security notions for encryption and notions tailored for secure database outsourcing and database privacy. The goal of strong encryption schemes is to hide every aspect of the plaintext (except for the length). The intention schemes for database outsourcing, searchable encryption, and database privacy is to hide certain aspects of the plain text or the queries, while preserving other aspects in order to provide some form of utility.

For database privacy, this utility is the accuracy results of analyses on the disclosed database in comparison to results of analyses on the original database. For database outsourcing and searchable encryption this utility is the support for efficient execution of queries. Therefore, a privacy notion can be seen as a trade off between confidentiality and utility.

Based on the generalised security notions for data outsourcing presented in Chapter 5, in this Chapter, we define indistinguishability over independent column permutations (IND-ICP), a static notion for secure database outsourcing that allows for efficient execution of SQL queries. Based on this notion, we define the notion /-IND-ICP, that provides security when the adversary is allowed to observe the execution of queries. In Chapter 7, we provide a database outsourcing scheme that fulfils IND-ICP as well as an extension of this scheme that fulfils /-IND-ICP.

Structure of this Chapter In this Chapter, we will define our security notions for database outsourcing. We will define IND-ICP, a static security notion, in Section 2. Furthermore, we show, that IND-ICP is an instance of generalised IND-CDA, the generalised Static Security notion for data outsourcing defined in Chapter 5. In Section 3, we will define /-IND-ICP, a generalisation of IND-ICP, and a dynamic variant of IND-ICP that provides security if the adversary observes query executions. This dynamic variant, is a direct instance of the generalised notion for Data Privacy from Chapter 5.

2. Indistinguishability under Independent Column Permutation

In this section, we present the notion *Indistinguishability under Independent Column Permutation* (IND-ICP), a security notion for secure database outsourcing. It allows for database outsourcing schemes that enable efficient execution of queries on the outsourced database. In Chapter 7, we will present such a scheme, discuss its efficiency, present and discuss implementations of this scheme, and provide benchmarks.

Informally, IND-ICP guarantees that an adversary does not learn the relations of the attribute values in the original database. The idea of IND-ICP is inspired from k-anonymity. The intention of the notion k-anonymity is to hide the relations of values of the sensitive predicate and individuals. While this intention seems reasonable, the notion k-anonymity is implemented as a syntactical notion without considering adversaries explicitly. We improve on this by carrying over the intention of k-anonymity to a semantic, game-based notion that considers PPT adversaries. Since IND-ICP is a cryptographic privacy notion, it overcomes some of the shortcomings of k-anonymity (cf. Chapter 4, Section 2.2). Furthermore, we do not differentiate between identifiers, quasi-identifiers, and sensitive information, resulting in a notion of a more general applicability. For example, a scheme that provides IND-ICP can also be used in scenarios that do not involve data related to individuals.

2.1. Formalisations

Since we operate on databases, we define mechanisms that transform databases as follows:

Definition 51 (Database Transformation)**.** *A* database transformation *is a PPT algorithm* $f : \mathsf{DB}' \to \mathsf{DB}''$ *with* $\mathsf{DB}', \mathsf{DB}'' \subseteq \mathsf{DB}$*. If* f *is a function, we call* f *a* database function*.*

An example of database transformations are database anonymisation mechanisms. Note that not all database transformations are functions. Consider for example transformations that involve probabilistic processes. A special case of database functions are *independent column permutations*:

Definition 52 (Independent Column Permutation)**.** *A column permutation is a database function that applies a permutation to a single column of a database. An* icp *is a set of column permutations. We call* Φ *the set of all icp.*

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

6.1.1: Database d

A	B	C
a_3	b_3	c_1
a_2	b_1	c_2
a_1	b_2	c_3

6.1.2: Database $p(d)$

Figure 6.1.: An example for a database before 6.1.1 and after 6.1.2 an independent column permutation $\mathfrak{p} = \{p_1, p_2, p_3\}$ with $p_1 = (13)$, $p_2 = (123)$, and $p_3 = id$.

An icp permutes the attribute values of a database within columns. Consider for example the databases depicted in Figure 6.1. The right database (Figure 6.1.2) is the result

of applying the permutation $\mathfrak{p}_1 = (13)$ to the first column, the permutation $\mathfrak{p}_2 = (123)$ to the second column, and the permutation $\mathfrak{p}_3 = id$ to the last column of the left database (Figure 6.1.1).

We use these independent column permutations to define a relation between databases. This relation will be the leakage relation (cf. Chapter 5, Section 6) for our security notion. Our security notion allows the adversary to learn the equivalence class of a given database. This is modelled with the following security game:

Security Game 14 ($\text{IND-ICP}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \text{Gen}(1^k)$.
2. *The adversary* $\mathcal{A}$ *is given input* 1^k *and oracle access to* $\text{Enc}(\cdot, K)$ *and continues having access to it.*
3. $\mathcal{A}$ *outputs a database* d *and an icp* $\mathfrak{p}$.
4. *The experiment chooses a random bit* $b \leftarrow \{0, 1\}$ *and sets* $d_0 = d$ *and* $d_1 = \mathfrak{p}(d)$.
5. $\mathcal{A}$ *is given* $\text{Enc}(d_b, K)$.
6. $\mathcal{A}$ *outputs a guess* b' *for* b.

The result of the experiment is 1 *if* $b' = b$ *and* 0 *else.*

In this security game, the adversary chooses a database and an icp. Then, the adversary has to guess whether this icp has been applied to the database prior to encryption. The oracle $\text{Enc}(\cdot, K)$ serves as an encryption oracle that lets the adversary encrypt arbitrary databases without revealing the encryption key K to the adversary. The experiment returns 1 if the adversary guessed right and 0 otherwise. Based on this security game, we can define the IND-ICP notion:

Definition 53 (Indistinguishability under Independent Column Permutation)**.** *A database outsourcing scheme* (Gen, Enc, Q) *has* Indistinguishability under Independent Column Permutation *(IND-ICP), if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function negl such that:*

$$\Pr[\textit{IND-ICP}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}(k) = 1] \leq \frac{1}{2} + \textit{negl}(k)$$

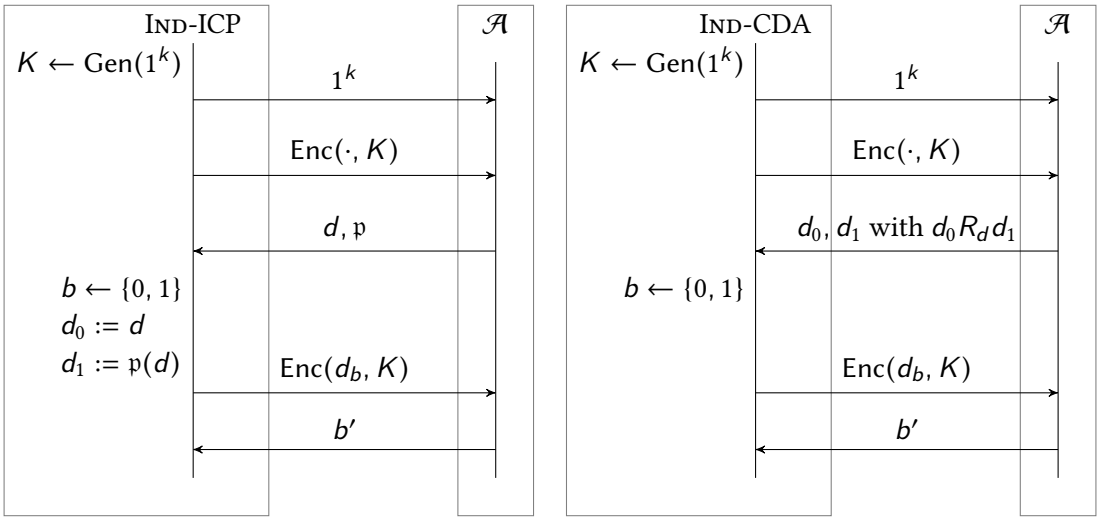
Since the relations are destroyed by independent column permutations, the notion IND-ICP prevents the adversary from learning them from the encrypted database. This security notion is weaker than classical cryptographic security notions for encryption. In classical experiments used to define security notions for encryption, the adversary is allowed to freely choose two elements either from a set of plaintexts (IND-CPA) or from a set of ciphertexts (IND-CCA). In the security game $\text{IND-ICP}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}$, the adversary can only select two databases that can be transformed into each other with an icp. This effectively weakens the security of the notion based on this game. For example, the adversary is allowed to learn the attribute values that occur in the database. We will later use this information to build an efficient and IND-ICP secure database outsourcing scheme.

Please note that the notion IND-ICP does not compose with arbitrary background knowledge. According to the No-Free-Lunch Theorem [KM11], this is not possible while maintaining some utility. We will discuss this and other shortcomings of the notion IND-ICP as well as shortcomings of schemes that fulfil this notion in Chapter 7, Section 8.

2.2. IND-ICP as an Instance of IND-CDA

IND-ICP is a direct instance of the generalised Static Security notion IND-CDA (Definition 47 in Chapter 5). We define the leakage relation as the relation induced by Φ, the set of all independent column permutations.

Theorem 8. *IND-ICP is equivalent to Static Security for databases with respect to* $R_\Phi = \mathrm{DB}_{/\Phi(\mathrm{DB})}$.



6.2.1: IND-ICP 6.2.2: IND-CDA

Figure 6.2.: The security games $\text{IND-ICP}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}$ and $\text{IND-CDA}^{\mathcal{A},R_d}_{(\mathsf{Gen},\mathsf{Enc})}$.

Proof. We set $D = \mathrm{DB}$. Each adversary that has a non-negligible advantage in $\text{IND-ICP}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}$ can efficiently be reduced to an adversary that has non negligible advantage in $\text{IND-CDA}^{\mathcal{A},R_\Phi}_{(\mathsf{Gen},\mathsf{Enc})}$ and vice versa. The reductions are straightforward (cf. Figure 6.2): The reduction from $\text{IND-ICP}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}$ to $\text{IND-CDA}^{\mathcal{A},R_\Phi}_{(\mathsf{Gen},\mathsf{Enc})}$ sets $d_0 := d$ and $d_1 := \mathfrak{p}(d)$, while the reduction $\text{IND-CDA}^{\mathcal{A},R_\Phi}_{(\mathsf{Gen},\mathsf{Enc})}$ to $\text{IND-ICP}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}$ sets $d = d_0$ and determines an icp $\mathfrak{p}$ with $\mathfrak{p}(d_0) = d_1$. □

2.3. IND-ICP as a Meaningful Security Notion

Security notions are not an end in itself. They have to be meaningful. Designing a scheme first and a security notion that is fulfilled by this scheme second can lead to a notion that very well describes the security properties of the scheme but is not meaningful for the context the scheme is used in. Since IND-ICP is a security notion for database outsourcing it has to be meaningful in the context database outsourcing. Databases are sets of tuples of the same domain. These tuples, for example, describe individuals or measurement results. In our model, the adversary knows the scheme of the database. Therefore, potential attribute values are not a secret.

In reality, often not the data points itself but their context – their relation to other points are of increased interest. For example for in a medical database not a single symptom

itself but its relations to other symptoms and to properties of the individual showing them may be relevant for diagnosis or finding side effects of drugs. Applications that combine large amounts of data and analyse relations, so-called *Big Data* applications, promise huge benefits for example for automation, artificial intelligence, and optimisation of infrastructures. Therefore, the assumption that not the data points itself but their relations are valuable is reasonable for many scenarios. Often, these valuable relations are also sensitive and have to be protected. Therefore, a security notion that hides relations is a meaningful notion.

Note that, of course, there are cases where the occurrence of a certain attribute value itself, independent of its relations to values of other attributes, deserves protection. A simple example are licence codes of software. The notion IND-ICP is a application specific and not an one fits all solution. As with all security measures, an evaluation in respect to the security requirements is necessary prior to a potential deployment of an IND-ICP secure mechanism.

3. l-Indistinguishability under Independent Column Permutation

In order to hide sensitive information, methods that fulfil the privacy notion k-anonymity partition the database into blocks of at least size k (cf. Chapter 3 Section 2.2.1). Based on this idea, the notion IND-ICP can be generalised.

In the security game $\text{IND-ICP}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}$ (Definition 14), the adversary may choose a arbitrary independent column permutation. Now, we partition the database into smaller blocks and restrict the icp to only permute values within these blocks. We use l for the size of the blocks in order to avoid confusion with the security parameter k of the encryption scheme. We define a subset of all independent column permutations, that only permute attribute values within a block of the database. The result of restricting the adversary in IND-ICP security game to only chose independent column permutations of this subset is a notion that generalises IND-ICP. We call this notion l-IND-ICP. We will use l-IND-ICP to define a database outsourcing scheme that provides l-IND-ICP in the presence of queries in Chapter 7, Section 5.

In order to define this generalised notion, we need to define independent column permutations that only operate within a specific partition of a database. Therefore, we define a partitioning scheme of a database:

Definition 54. *(Partitioning Scheme) Let* $f : \text{DB} \to 2^{\text{DB}}$ *be a function that partitions a database* d *into subsets of* d *with:*

- $d = \bigcup_{p_i \in f(d)} p_i$.
- $|f(d_1)| = |f(d_2)|$ *and* $\forall p_i \in f(d_1), q_i \in f(d_2) : |p_i| = |q_i|$

We call f *a partitioning scheme for databases and* i *the partition id of partition* p_i.

The first condition ensures that the result of $f(d)$ actually is a partitioning of d. The second condition guarantees that f only depends on the size of d, but not on the data in d.

Definition 55 (Partitioning Scheme with Partitions of Size l)**.** *Let f be a partitioning scheme for databases with $\forall p_i \in f(d) : |p_i| \geq l$. we call f a* partitioning scheme with partitions of size l. *For a database $d \in DB$, we call an element of $f(d)$ a partition of d or an l-bucket of d.*

Please note, that if the database has less then $2 \cdot l$ tuples, than $f(d) = d$ for any partitioning scheme f with partitions of size l. With a partitioning scheme, we now can define independent column permutations that respect partitions of a database. This means that the individual permutations only operate within a partition.

Definition 56 (Partitioning Respecting Independent Column Permutation)**.** *Let f be a partitioning scheme for databases. A column permutation $\mathfrak{p}$ that respects f is a column permutation where $f(d)$ and $f(\mathfrak{p}(d))$ only differ in one element. An* independent column permutation *(icp) that respects f is a set multiple column permutations that respect f.*

Note, that an icp that respects a partitioning scheme can permute attribute values in all partitions and columns, since an icp is comprised of multiple column permutations. With this definition of special independent column permutations, we can define l-IND-ICP as follows:

Definition 57 (l-Indistinguishability under Independent Column Permutation)**.** *A database outsourcing scheme* (Gen, Enc, Q) *has* l-Indistinguishability under Independent Column Permutation for f *(l-IND-ICP), if for all PPT adversaries $\mathcal{A}_f$ that only choose independent column permutations that respect f, there exists a negligible function negl such that:*

$$\Pr[\mathit{IND\text{-}ICP}^{\mathcal{A}_f}_{(\mathsf{Gen},\mathsf{Enc})}(k) = 1] \leq \frac{1}{2} + \mathit{negl}(k)$$

Note that if n_{max} is the size of the biggest database in DB, and if $l \geq \lfloor \frac{n_{max}}{2} \rfloor$ than l-IND-ICP is equivalent to IND-ICP, since the only partitioning of a database with partitions of at least size l is the database itself. If, however, l is smaller than $\lfloor \frac{n_{max}}{2} \rfloor$ a method that fulfils l-IND-ICP does not necessary fulfil IND-ICP. Consider the example depicted in Figure 6.3. Figure 6.3.1 shows the original database while Figure 6.3.2 shows

	Name	Surname
1	Bob	Jones
2	Alice	Smith
3	Carol	Jones
4	Eve	Brown

6.3.1: input

Name	Surname
{Alice, Bob}	{Jones, Smith}
{Carol, Eve}	{Brown, Jones}

6.3.2: transformed database

Figure 6.3.: An example for the input 6.3.1 and the output 6.3.2 of a mechanism that fulfils l-IND-ICP for $l = 2$ but not IND-ICP.

the database transformed by a mechanism that fulfils 2-IND-ICP for this database. The original database can be partitioned into two 2-buckets comprised of the rows 1 and 2, and rows 3 and 4. An adversary does not learn more about the relations of the attribute values than the fact that the attribute values within the 2-buckets are related. The adversary can, however, given the database in Figure 6.3.2, infer that no one named Alice Brown is in

the original database. If the mechanism used to transform the database fulfilled IND-ICP, the adversary would not be able to infer this.

Based on the static notion l-IND-ICP, we can instantiate a dynamic notion, that considers the adversary observing the execution of queries. We call this notion no query advantage over l-IND-ICP:

Definition 58. *(No Query Advantage over l-IND-ICP) A database outsourcing scheme* (Gen, Enc, Q) *for databases of size n has* no query advantage over l-IND-ICP *if it has* n_1, n_2, n_3-Data Privacy *with respect to R_{Φ_f} with:*

- *R_{Φ_f} is the relation implied by independent column permutations that respect a partitioning scheme f*
- $n_1 = \mathsf{poly}(k)$
- $n_2 = \mathsf{poly}(k)$
- $n_3 = \mathsf{poly}(k)$

In Chapter 7, Section 5, we provide a scheme that fulfils no query advantage over l-IND-ICP.

7. Mechanisms for Database Outsourcing

This chapter is partially based on work already published in [AGH11; Hub+13; Hub+11; Hei+10; HMN13; HMN14; HH14; Boe13; Bar14; Har11; Mar13] and [Ach+16].

1. Introduction

Formal notions, their relations, and composability properties are important for cryptographic research. For example, they enable the comparison of the security of mechanisms and the reasoning about security properties without the need to consider mechanisms explicitly. In order to realise secure applications, however, there also have to be mechanisms that fulfil these notions. Considering the trend of outsourcing data and calculations and the fact that many applications rely on databases, mechanisms for secure database outsourcing are especially desirable. Clients outsourcing their data with mechanisms that do not provide any security guarantees are dependent on the goodwill of the service provider and all potential internal adversaries. They lose control over their data and risk misuse or disclosure to third parties.

In order to be used in applications, mechanism with security guarantees have to be practical. The overhead they introduce has to be reasonable. One may argue that the overhead of a security mechanism for outsourcing has to be lower than the benefit of outsourcing. The overhead can be considered at design time in the O notation and measured after implementation of the mechanism. Since the O notation does not consider constant factors that can play a huge role for implementations, both, theoretical scaling properties as well as actual benchmarks have to be considered when choosing a implementation of a security mechanism.

Furthermore, implementing a scheme with provable security properties introduces a context switch. The security properties are proven in an abstract model whereas the implementation of the scheme is used in the real world. This can introduce side channels that allow to break the security of the scheme.

In this section, we will define the MimoSecco database outsourcing scheme, discuss its scaling properties in the O notation, prove that it fulfils IND-ICP, present implementations of this scheme, and provide benchmarks. One implementation, the MimoSecco research prototype, has been integrated in a prototype of a commercial database abstraction and synchronisation layer. The other implementation, the Cumulus4j plug-in, is free software under the GNU Afero General Public License (AGPL) and already used in a commercial invoice processing software as a service. Furthermore, we present optimisations of the MimoSecco scheme with different performance properties, depending on the data stored and the queries. In the last section of this chapter, we discuss potential side channels of implementations of database outsourcing schemes.

Structure of this Chapter In Section 2, we provide the preliminaries for this chapter: We provide an overview of database outsourcing schemes found in literature, discuss efficient query execution in the context of database outsourcing schemes, and define the classes of queries used by our schemes. Section 3 discusses Differential Privacy in the context of database outsourcing. In Section 4, we define the MimoSecco scheme, discuss its efficiency and prove that it fulfils IND-ICP. In Section 5, we provide a variant of this database outsourcing scheme that provides provable security even if the adversary is allowed to observe query executions. Section 6 presents and discusses the implementations MimoSecco and Cumulus4j and provides benchmark results. In Section 7, we present performance optimisations of the index structures of our scheme and provide benchmarks. Section 8 discusses side channels of database outsourcing schemes and implementations.

2. Preliminaries

In this section, we provide an overview over database outsourcing schemes, that can be found in literature. These schemes use cryptographic primitives such as encryption and hash functions as well as fragmentation of databases and generation of indices in order to support efficient execution of queries. In contrast to our scheme, many of these schemes, however, do not provide a formal definition of their security properties. We do not focus on searchable encryption schemes, since database outsourcing schemes, in general, provide support for a larger range of queries (e. g. SQL). Since many searchable encryption schemes provide a formal definition of their security, they already heave been discussed in the context of privacy notions for data outsourcing in Section 2 in Chapter 5.

In the remainder of this section, we discuss what can be considered an efficient execution of a query for a database outsourcing scheme and define the classes of queries that we differentiate and consider in this chapter.

2.1. Database Outsourcing Schemes in Literature

To our knowledge, the first practical approach to secure database outsourcing was presented by Hacigümüs et. al. in [HIM02]. Based on this approach, additional solutions have been suggested [Hac+02; Dam+03; HMT04; Ces+05]. The basic idea of these approaches is to encrypt the database row-wise and, in order to support efficient query processing, to enhance this encrypted database with coarse grained indices. A coarse grained index for an attribute partitions its value space and assigns a unique id to each partition. An attribute value can be accessed by the id of its partition. Figure 7.1 shows an example from [Hac+02]. The goal of this approach is to increase the uncertainty of the adversary about concrete attribute values. An approach to model and assess inference exposure for coarse grained indices can be found in [Ces+05]. Since coarse grained indices lead to false positives in the (intermediate) query result sets, such an approach, depending on the data, introduces a large overhead.

Another idea is to achieve practical Data Privacy by data partitioning. The approaches in [Agg+05; De +10; SHJ12] introduce privacy constraints with the goal to fulfil them by fragmenting or encrypting the data [Agg+05; De +10; De +13] or fragmentation and addition of dummy data [SHJ12].

In [NC11], Nergiz et. al. use the Anatomy approach [XT06] with the goal to achieve Data Privacy in a database outsourcing scenario. Since this approach focuses on hiding re-

eid	*ename*	*salary*	*addr*	*did*
23	Tom	70k	Maple	40
860	Mary	60k	Main	80
320	John	50k	River	50
875	Jerry	55k	Hopewell	110

7.1.1: emp

etuple	eid^S	$ename^S$	$salary^S$	$addr^S$	did^S
1100110011110010…	2	19	81	18	2
1000000000011101…	4	31	59	41	4
1111101000010001…	7	7	7	22	2
1010101010111110…	4	71	49	22	4

7.1.2: emp^S

Figure 7.1.: The tables *emp* and emp^S (taken from [Hac+02]). The attribute *etuple* of table emp^S holds encrypted rows of the table *emp*. The attributes eid^S, $ename^S$, $salary^S$, $addr^S$, and did^S are coarse grained indices for the attributes *eid*, *ename*, *salary*, *addr*, *did*. An attribute value of a coarse graind index stands for a partition of the corresponding attribute's value space.

lations, it is related to the database outsourcing scheme we present in Section 4. Instead of explicitly defining privacy constraints, similar to k-anonymity (cf. Chapter 3, Section 2.2), the Anatomy approach introduces blocks of tuples (*quasi-identifier-group (QI-group)* or *equivalence class*, cf. Section 2.2 in Chapter 3). Instead of generalising or suppressing the attribute values of the quasi-identifier attributes, the Anatomy approach partitions the original table into two tables: One containing the quasi-identifier attributes and one containing the sensitive information. Additionally, an attribute *GID* is introduced. This attribute maps a row to its equivalence class. In order to reconstruct the original database, Nergiz et. al. introduce a unique sequence number for each tuple of the original database. The table with the quasi-identifier attributes contains these sequence numbers in plain text, in the table with the sensitive information, the sequence numbers are hashed. Figure 7.2 shows an example. One advantage of this scheme is that the attribute *GID* allows to compute a table containing a join of multiple tables on the server. While the Anatomy approach for database outsourcing also intends to hide relations between attribute values, it does not provide a formal security notion. Furthermore it requires, similar to k-anonymity, a classification of attributes into quasi identifier and a single sensitive information attribute, which can be unclear for certain databases, for example databases with attributes that are sensitive but also can be used to identify individuals.

The SECURUS approach [KJ14], also, uses privacy constraints. A privacy constraint, for example, prohibits unencrypted storage of certain attributes on the same server. The focus of SECURUS is to automatically find an outsourcing scheme based on a catalogue of privacy preserving techniques such as data partitioning and encryption that fulfils a given set of privacy constraints and supports the efficient execution of queries from a given set. An approach similar to SECURUS is described in [Cir+10]. The advantage of privacy constraints is that they can easily be defined by a domain expert. The domain expert, however, also has to keep in mind the underlying mechanisms used to fulfil the privacy

Patient	Age	Adress	GID	SEQ
Ike	41	Dayton	1	1
Eric	22	Richmond	1	2
Olga	30	Lafayette	2	3
Kelly	35	Lafayette	2	4
Faye	24	Richmond	3	5
Mike	47	Richmond	3	6
Jason	45	Lafayette	4	7
Max	31	Dayton	4	8

7.2.1: $Patient_{QIT}$

HSEQ	GID	Disease
$H_k(1)$	1	Cold
$H_k(2)$	1	Fever
$H_k(3)$	2	Flu
$H_k(4)$	2	Cough
$H_k(5)$	3	Flu
$H_k(6)$	3	Fever
$H_k(7)$	4	Cough
$H_k(8)$	4	Flu

7.2.2: $Patient_{SNT}$

Figure 7.2.: An example for an anatomised database (taken from [NC11]). The quasi-identifier attributes are separated from the sensitive information (the attribute *disease*). Equivalence classes are defined by the attribute *GID*. The original data can be restored with the attributes *SEQ* and *HSEQ*. Only the client has the key k and can calculate the hashes $H_k(\cdot)$.

constraints. Otherwise he may define privacy constraints that cannot be fulfilled or can only be fulfilled by a scheme that does not support efficient query execution. Another drawback of these privacy constraints is that they do not consider adversaries. There are no guarantees for which adversaries the constraints hold. Furthermore, fragmentation-approaches involve multiple servers, while this work focuses on single server solutions. Note that there are other multi server solutions, for example Blind Seer [Pap+14], a multi server DBMS based on Bloom filters that supports arbitrary Boolean queries. The security properties of Blind Seer are modelled in a simulation-based framework. Leakage profiles describe what each party learns.

CryptDB [Pop+11] is an approach different to the approaches above. Instead of relying on explicit indices or fragmentation, it uses the concept of onions of encryptions. Onions nest encryptions of attribute values encrypted with schemes with different properties. If a query cannot be executed because of the current encryption level of an attribute, the outermost layer of all entries of this attribute is removed. The security of CryptDB is not defined as formal security notion and depends on the queries issued to the database. In a model where the adversary may choose queries CryptDB provides no security at all.

Another interesting, more recent approach is the Shuffle Index [Vim+15]. This approach combines dummy accesses, caching of accesses, and shuffling of nodes of the index tree in order to provide data privacy as well as query privacy. This approach reminds of oblivious RAMs [GO96] (cf. Chapter 5, Section 2.2). The authors do provide an elaborate security analysis and show that the server can not distinguish searches from inserts, which is also similar to oblivious RAMs. They do not use formal security notions, for example from previous work on oblivious RAMs, but show that the entropy of the adversary increases over time and provide experimental results.

2.2. Efficient Query Execution

When speaking about query execution, we consider the computational overhead on the client and the server as well as the message overhead between them. In [KC05],

Kantarcioglu and Clifton propose that a practical database outsourcing scheme should support efficient execution of queries. The argue that a database outsourcing scheme where the cost of every query is linear in the size of the database cannot be considered as efficient. The requirement of an execution time that is sublinear in the size of the database, however, can not be met for any query. For example, there are queries where the whole database is the result set. Another example are queries where every tuple in the database influences the result set. Therefore, when speaking about efficient query support, we compare the cost of the query execution on the plain database with the complexity of the query execution on the outsourced database and call the difference the overhead. We call a data outsourcing scheme efficient, if it has a low overhead in the O notation. Optimally, the overhead is in $O(1)$.

2.3. Queries in this Work

In Definition 37 in Section 3, we defined a query as a PPT that operates on data sets. A query returns a result set and an updated data set. For our database outsourcing schemes, we focus on relational databases and relational algebra or SQL as query language. Therefore, we differentiate between SELECT, DELETE, INSERT and UPDATE queries. We define these queries with the use of projections and selections from relational algebra (cf. Chapter 2, Section 3).

Definition 59 (SELECT queries)**.** *We define the set $Q_{select} \subset Q$ such that $q \in Q_{select}$ iff for all $d \in$ DB the following conditions hold:*

- $q : DB \rightarrow DB \times DB$
- *q contains a selection σ and a projection* Π.
- $q_d = d$
- $q(d) = \Pi(\sigma(d))$

A SELECT query only retrieves information and does not change the state of the database.

Definition 60 (DELETE queries)**.** *We define the set $Q_{delete} \subset Q$ such that $q \in Q_{delete}$ iff for all $d \in$ DB the following conditions hold:*

- $q : DB \rightarrow DB \times DB$
- *q contains a selection σ*
- $d_q = d \setminus \sigma(d)$
- $q(d) = \{\}$

A DELETE query deletes tuples from a database.

Definition 61 (INSERT queries)**.** *We define the set $Q_{insert} \subset Q$ such that $q \in Q_{insert}$ iff for all $d \in$ DB with attributes A the following conditions hold:*

- $q : DB \rightarrow DB \times DB$
- *q contains a tuple t with attributes A_t*
- *If $A_t = A$:*

  - $d_q = d \cup \{t\}$
  - $q(d) = \{id_t\}$
- *If* $A_t \neq A$*:*
  - $d_q = d$
  - $q(d) = \{\}$

An INSERT query inserts a single tuple t into a database and returns the (unique) row id of the newly inserted tuple. In this work, we only consider *correct* INSERT queries. This means, we only consider the case where $A_t = A$.

Definition 62 (UPDATE queries)**.** *We define the set* $Q_{update} \subset Q$ *such that* $q \in Q_{update}$ *iff for all* $d \in$ DB *with attributes* A *the following conditions hold:*

- $q : DB \rightarrow DB \times DB$
- q *contains a selection* σ *of an attribute* A_q *and an attribute value* v *of* A_q
- *If* $A_q \in A$*:* $d_q = (d \setminus \sigma(d)) \cup \sigma(d)_v$*, where* $\sigma(d)_v$ *is* $\sigma(d)$ *with all attribute values of attribute* A_q *changed to* v
- *If* $A_q \notin A$*:* $d_q = d$
- $q(d) = \{\}$

According to this definition, an UPDATE query allows to change the values of a single attribute. In order to change the values of more than one attribute or in order to apply more than one change to the values of a single attribute, one can combine multiple update queries. In this work, we only consider *correct* UPDATE queries. This means, we only consider the case where $A_t = A$. For examples of queries, we will use a representation similar to SQL.

3. Differential Privacy and Database Outsourcing

Since its introduction, the notion Differential Privacy [Dwo08a], has received much attention in the fields of data privacy and privacy preserving data analysis [Dwo08b; KM14; McS09]. In this chapter, we examine Differential Privacy as a notion for secure database outsourcing. We recall Definition 20:

Definition 20 (ϵ-Differential Privacy)**.** *A randomized function* $\mathcal{K}$ *gives* Differential Privacy *if for all data sets* D_1 *and* D_2 *differing on at most one element, and all* $S \subseteq Range(\mathcal{K})$

$$\Pr[\mathcal{K}(D_1) \in S] \leq e^{\epsilon} \times \Pr[\mathcal{K}(D_2) \in S]$$

Since our goal is a database outsourcing mechanism, we can not remove or alter tuples in the original database. Therefore, the idea for an database outsourcing mechanism (or the mechanism Enc of an outsourcing scheme, to be more precise), that fulfils Differential Privacy is to add so-called dummy tuples to the database prior to outsourcing. For the sake of simplicity, we assume, that the client has an oracle to tell original tuples and dummy tuples apart. In implementations, this can be done by an encrypted dummy bit. With this oracle of the client, we can use the statistic Differential Privacy definition that is much simpler than the computational counterparts [MT07].

With such a *dummy mechanism*, however, we can not achieve Differential Privacy under practicability constraints:

Theorem 9 (Impracticality of Dummy Mechanisms that fulfil ϵ-Differential Privacy)**.** *Let U be the universe of all possible tuples of databases with attributes A. A mechanism $\mathcal{K} : U^* \rightarrow U^*$, that adds dummy tuples to a database but does not always output all tuples of U does not fulfil ϵ-Differential Privacy.*

The following proof is a proof by contradiction. It assumes the existence of a dummy mechanism that fulfils Differential Privacy and does not always output all tuples in the universe.

Proof. Let U be the universe of all possible tuples for a database with attributes A. Furthermore, let $\mathcal{K} : U^* \rightarrow U^*$ be a mechanism that adds tuples from U to a given database but does not always output all tuples of U.

Then, there is a database $d_0 = \{r_1^0, r_2^0, \ldots ..r_n^0, \} \in U^*$ and a tuple $r_1 \in U$ with $\Pr[r_1 \notin \mathcal{K}(d_0)] > 0$. Then, there is a database $d' \in \mathit{Range}(\mathcal{K})$ with $r_1 \notin d'$ and $\Pr[\mathcal{K}(d_0) = d'] > 0$. We set the database $d_1 = \{r_1, r_2^0, \ldots r_n^0, \}$ and get $\Pr[\mathcal{K}(d_1) = d'] = 0$. The databases d_0 and d_1 differ in exactly one tuple. Consequently, $\mathcal{K}$ does not fulfil differential privacy, since there is no ϵ such that the following holds:

$$0 < \Pr[\mathcal{K}(d_0) = d'] \leq e^{\epsilon} \times \Pr[\mathcal{K}(d_1) = d'] = 0$$

This concludes our proof. □

According to Theorem 9, there are no practical mechanisms, that only add dummy data and fulfil ϵ-Differential Privacy. There are however more relaxed definitions of Differential Privacy. One of them is (ϵ, δ)-Differential Privacy, that allows for mechanisms that do not fulfil ϵ-Differential Privacy for unlikely events [Dwo+06b; MT07]:

Definition 63 ((ϵ, δ)-Differential Privacy)**.** *A randomised function $\mathcal{K}$ gives* (ϵ, δ)-Differential Privacy *if for all data sets D_1 and D_2 differing on at most one element, and all $S \subseteq \mathit{Range}(\mathcal{K})$*

$$\Pr[\mathcal{K}(D_1) \in S] \leq e^{\epsilon} \times \Pr[\mathcal{K}(D_2) \in S] + \delta$$

In the following, we define a dummy mechanism and discuss its bounds for δ in order for the mechanism to fulfil (ϵ, δ)-Differential Privacy.

Definition 64 (Dummy Mechanism)**.** *Let U be the universe of all possible tuples for a database with attributes A. Furthermore, let* DB *be the set of all databases with attributes A, $r \in U$, $d \in$ DB and $p \in [0, 1]$. We define the* dummy mechanism $\mathcal{K} :$ DB $\rightarrow$ DB *as follows:*

- $\mathcal{K}(d) = d$, *with probability* p
- $\mathcal{K}(d) = \mathcal{K}(d) \cup r$, *with probability* $(1 - p) \cdot \Pr[r]$

The idea of this mechanism is to output the original database with a probability p and to recursively add a random tuple to the output with probability $1 - p$. We assume a uniform and independent probability distribution for the tuples in universe U and get:

- $\Pr[\mathcal{K}(d) = d] = p$
- $\Pr[\mathcal{K}(d) = d | d'] = p \cdot (1 - p)^{|d'|} \prod_{r \in d'}(\Pr[r])$, for $d' \in$ DB

For this mechanism, it is possible that $\Pr[\mathcal{K}(d_0) \in S] > 0$ and $\Pr[\mathcal{K}(d_1) \in S] = 0$. For example, this is the case for $S = \mathcal{K}(d_0) = d_0$. According to Definition 64, this event has probability p.

In order to find a lower bound for δ, we have to consider all cases where $\Pr[\mathcal{K}(d_0) \in S] > 0$ and $\Pr[\mathcal{K}(d_1) \in S] = 0$ for all d_0, d_1 with $d_0 \Delta d_1 = 1$.

When two databases d_0, d_1 differ in one row, w.l.o.g. the database d_1 contains a tuple r_1 that is not contained in d_0. The biggest $S \subseteq \mathit{Range}(\mathcal{K})$ with $\Pr[\mathcal{K}(d_0) \in S] > 0$ and $\Pr[\mathcal{K}(d_1) \in S] = 0$ is the set of all databases, that contain all tuples of d_0 and, depending on their size, additional tuples of U but not the tuple r_1. We call this set S'. The dummy mechanism chooses and adds an $r \in U$ with $r \neq r_1$ to a given database with probability $\frac{n-1}{n}$. Therefore, the probability of $\mathcal{K}(d_0) \in S'$ is:

$$\sum_i p(1-p)^i \left(\frac{n-1}{n}\right)^i = \frac{p}{1 - \frac{n(1-p)}{n-1}} = \frac{p(n-1)}{np-1} = \frac{n-1}{n-\frac{1}{p}}$$

Since n is the size of the universe U (i.e. the number of all possible tuples), this lower bound for δ gets close to 1 for real world examples. Consequently, one can argue that it does not make much sense to describe the privacy of the dummy mechanism in terms of (ϵ, δ)-Differential Privacy. Since the dummy mechanism has similarities to randomised response [War65], intuitively, the dummy mechanism provides a certain level of privacy. A suitable formal notion for the privacy of the dummy mechanism could certainly provide more insights into this mechanism.

4. An Ind-ICP Secure Database Outsourcing Scheme

In Definition 53 in Chapter 6, Section 2, we defined the notion Ind-ICP. We argued that this notion allows for efficient and secure database outsourcing. In this section, we will define such an outsourcing scheme and discuss its efficiency.

4.1. Formalisation of the MimoSecco Database Outsourcing Scheme

A database outsourcing scheme (Gen, Enc, Q) consists of PPTs for key generation (Gen) and encryption (Enc) and of a set Q of efficient two party protocols that execute a query for outsourced databases (cf. Definition 40 in Chapter 5). Thus, in order to define an outsourcing scheme, we need to define Gen, Enc, and Q. Our scheme will use an internal encryption scheme and inherit the key generation from it. In order to distinguish the PPTs from each other and from the PPTs of the framework of Section 3, we will call the internal encryption scheme $(\mathsf{Gen}_\mathrm{I}, \mathsf{Enc}_\mathrm{I}, \mathsf{Dec}_\mathrm{I})$ and our outsourcing scheme $(\mathsf{Gen}_\mathrm{DB}, \mathsf{Enc}_\mathrm{DB}, \mathsf{Q}_\mathrm{DB})$. We will start with the definition of the PPT Enc_DB for our scheme.

Encryption Mechanism We define the mechanism Enc_DB as a set of tables that is parametrised by the original database. Therefore, we need some preliminary definitions. The set of all attribute values of a column of a database is defined as follows:

Definition 65 (Set of all Values of a Column)**.** *Let d be database of size n and with m attributes and $j \in \{1, \dots, m\}$. The* set of all values of column j *is defined as $\mathcal{V}(d(\cdot, j)) := \{d(i, j) | i \in \{1, \dots, n\}\}$. Let this set be ordered.*

Note, that $\mathcal{V}(d(\cdot, j))$ is a set in contrast to a database which is a multiset. The set $\mathcal{V}(d(\cdot, j))$ contains each attribute value of a column only once, even if the attribute value occurs multiple times in this column.

Next, we define the set of all row indices an attribute value occurs in a certain column:

Definition 66 (Set of all Row Ids of an Attribute Value)**.** *Let d be a database of size n and with m attributes and $j \in \{1, \dots, m\}$. For a value $v \in d(\cdot, j)$, we define $\mathcal{R}(d, j, v) := \{i \mid d(i, j) = v\}$, the set of all indices of all rows containing value v in column j.*

row	A_1 = name	A_2 = surname
1	Alice	Smith
2	Bob	Smith
3	Alice	Jones

Figure 7.3.: The table *names*. The sets $\mathcal{V}$ and $\mathcal{R}$ as defined in Definitions 65 and 66 are depicted in Figure 7.4

The tables in Figures 7.3 and 7.4 show an example for $\mathcal{V}(d(\cdot, j))$ and $\mathcal{R}(d, j, v)$. Table *names* in Figure 7.3 contains two attributes *name* and *surname* and three rows. The row ids are indicated with the additional column *row id*. The order of the attributes are indicated with A_1 and A_2, respectively. For this table, the set of all attribute values of the

$\mathcal{V}(names(\cdot, \cdot))$
$\mathcal{V}(names(\cdot, 1)) = \{\text{Alice}, \text{Bob}\}$ $\mathcal{V}(names(\cdot, 2)) = \{\text{Smith}, \text{Jones}\}$

7.4.1: $\mathcal{V}(names(\cdot, \cdot))$

$\mathcal{R}(names, 1, \cdot)$	$\mathcal{R}(names, 2, \cdot)$
$\mathcal{R}(names, 1, Alice) = \{1, 3\}$	$\mathcal{R}(names, 2, Smith) = \{1, 2\}$
$\mathcal{R}(names, 1, Bob) = \{2\}$	$\mathcal{R}(names, 2, Jones) = \{3\}$

7.4.2: $\mathcal{R}(names, \cdot, \cdot)$

Figure 7.4.: The sets $\mathcal{V}(names(\cdot, \cdot))$ and $\mathcal{R}(names, \cdot, \cdot)$ for the example table *names* in Figure 7.3.

first attribute (*name*) and the second attribute (*surname*) as well as the list of row ids for each attribute value are depicted in Figure 7.4.

As mentioned above, our encryption mechanism $\mathsf{Enc}_{\mathrm{DB}}$ transforms a database into a set of tables. With the help of Definitions 65 and 66 we now define these tables:

Definition 67 (Data Table)**.** *Let $(\mathsf{Gen}_I, \mathsf{Enc}_I, \mathsf{Dec}_I)$ be an encryption scheme and let d be a database of size n with attributes $\{A_1, A_2, \dots, A_m\}$. We define the table d_{data} as:*

$$d_{data} := \{(i, \mathsf{Enc}_I(d(i, \cdot), K) \mid i \in \{1, \dots, n\}\}$$

with attributes $\{row,\ data\ (A_1, A_2, \dots, A_m)\}$. This table is ordered according to the first column and the order of $\mathbb{N}$.

The data table contains a row-wise encryption of the original database d. Additionally to the data table, we define an index table for each attribute of the original database. These index tables will be used by the protocols in order to efficiently execute queries.

Definition 68 (Index Table)**.** *Let* $(\mathsf{Gen}_I, \mathsf{Enc}_I, \mathsf{Dec}_I)$ *be an encryption scheme,* d *a database with attributes* $\{A_1, A_2, \dots, A_m\}$*, and* $j \in \{1, \dots, m\}$*. We define* d_{index,A_j} *as the table*

$$d_{index,A_j} := \{(v, \mathsf{Enc}_I(\mathcal{R}(d, j, v)), K) \mid v \in \mathcal{V}(d(\cdot, j))\}$$

with the attributes $\{values, rows\}$*. This table is ordered according to first column and the order of* $\mathcal{V}(d(\cdot, j))$*.*

With the definition of the data table and the index tables, we can define the encryption mechanism of our outsourcing scheme:

Definition 69 ($\mathsf{Enc}_{\mathrm{DB}}$)**.** *Let* d *be a database with attributes* $\{A_1, A_2, \dots, A_m\}$ *and let* $(\mathsf{Gen}_I, \mathsf{Enc}_I, \mathsf{Dec}_I)$ *be an encryption scheme. We define the PPT* Enc_{DB} *as follows:*

$$\mathsf{Enc}_{DB}(d) = \{d_{data}, d_{index,A_1}, \dots, d_{index,A_m}\}$$

The PPT $\mathsf{Enc}_{\mathrm{DB}}$ takes a database as input and outputs the data table a well as an index table for each attribute of the input database.

For an example consider the database depicted in Figure 7.5. This database has the the attributes *name* and *surname*. Figure 7.6 depicts $\mathsf{Enc}_{\mathrm{DB}}(d)$. Our encryption mechanism

name	surname
Alice	Smith
Bob	Smith
Alice	Jones

Figure 7.5.: A simple example database d. Figure 7.6 depicts $\mathsf{Enc}_{\mathrm{DB}}(d)$.

generates the data table d_{data} that contains each row of the original table encrypted with the mechanism $\mathsf{Enc}_{\mathrm{I}}$. Furthermore, for each attribute of the original table, $\mathsf{Enc}_{\mathrm{DB}}$ generates an index table. Each index table contains the attribute values that occur in the corresponding column and for each attribute value an encrypted list of the rows, the value occurs as a value of this attribute. For example the first row of the table $d_{index,name}$

row	data (name,surname)
1	$\mathsf{Enc}_{\mathrm{I}}((Alice, Smith), K)$
2	$\mathsf{Enc}_{\mathrm{I}}((Bob, Smith), K)$
3	$\mathsf{Enc}_{\mathrm{I}}((Alice, Jones), K)$

7.6.1: d_{data}

values	rows
Alice	$\mathsf{Enc}_{\mathrm{I}}((1, 3), K)$
Bob	$\mathsf{Enc}_{\mathrm{I}}(2, K)$

7.6.2: $d_{index,name}$

values	rows
Smith	$\mathsf{Enc}_{\mathrm{I}}((1, 2), K)$
Jones	$\mathsf{Enc}_{\mathrm{I}}(3, K)$

7.6.3: $d_{index,surname}$

Figure 7.6.: The tables $\mathsf{Enc}_{\mathrm{DB}}(d, K)$: The data table d_{data} 7.6.1 and the index tables $d_{index,name}$ 7.6.2 and $d_{index,surname}$ 7.6.3. The index tables contain encrypted row indices of the data table.

in Figure 7.6.2 contains the attribute value *Alice* and an encrypted list of the indices {1, 3} since the surname *Alice* occurs in the first and the second row in the original database in Figure 7.5.

In Section 2.3, we defined the sets of queries Q_{select}, Q_{delete}, Q_{insert}, and Q_{update} (cf. Definitions 59-62). In the following, we will define the protocols of $\mathrm{Q_{DB}}$ that execute these queries for outsourced databases $\mathsf{Enc}_{\mathrm{DB}}(d, K)$.

We define the sets of protocols $\pi_{q_{select}}$, $\pi_{q_{delete}}$, $\pi_{q_{insert}}$, and $\pi_{q_{update}}$, that execute SELECT, DELETE, INSERT, and UPDATE queries, respectively. Furthermore, we argue their correctness. Consistent with Definition 38 in Chapter 5, the parties of each (meta) protocol are a server $\mathfrak{S}$ and a client $\mathfrak{C}$. The input of the client $\mathfrak{C}$ is a query and a secret encryption key K. The input of the server is the encrypted database $\mathsf{Enc}_{\mathrm{DB}}(d, K)$. The output of the client is the result set and the output of the server is the (updated) outsourced database.

Protocol for SELECT Queries The following protocol executes SELECT queries for $\mathsf{Enc}_{\mathrm{DB}}(d, K)$:

Protocol 1 ($\pi_{q_{select}}$: Protocol for SELECT Queries)**.**
Input of client $\mathfrak{C}$*:* $q_{select} \in Q_{select}$, K
Input of server $\mathfrak{S}$*:* $\mathsf{Enc}_{\mathrm{DB}}(d, K)$
Let σ *be the selection of* q_{select}, Π *be the projection of* q_{select}, *and* c *be the conditions of* σ.

1. *The client* $\mathfrak{C}$ *sends the conditions* $c = c_1, \ldots c_l$ *of* σ *to the server* $\mathfrak{S}$.
2. *For each condition* c_i *with attribute* A_i, *the server* $\mathfrak{S}$ *returns the tuples of* d_{index,A_i} *where* c_i *holds for the attribute* values *to the client* $\mathfrak{C}$.
3. *The client* $\mathfrak{C}$ *decrypts the value of the attribute* rows *of every tuple received by the server* $\mathfrak{S}$ *and receives for each* $c_i \in c$ *a set of ids of tuples for which the condition* c_i *holds.*
4. *The client* $\mathfrak{C}$ *computes* ids_c, *the set of ids of tuples for which the condition* c *holds. If* ids_c *is empty, client* $\mathfrak{C}$ *sends* $\perp$ *to the server* $\mathfrak{S}$, *outputs an empty database and the protocol ends, otherwise the client* $\mathfrak{C}$ *sends* ids_c *to the server* $\mathfrak{S}$.
5. *The server* $\mathfrak{S}$ *returns the tuples of* d_{data} *whose ids are in* ids_c *to the client* $\mathfrak{C}$.
6. *The client* $\mathfrak{C}$ *decrypts the received tuples and applies the projection* Π *to them (and returns them).*

Output of client $\mathfrak{C}$*: projection* Π *of received tuples*
Output of server $\mathfrak{S}$*:* $\mathsf{Enc}_{DB}(d, K)$

In Steps 1 and 2, the client queries the index tables for tuple ids relevant to the query. Since the conditions c may contain more than one atomic condition, the client may get *false positives* in this step. This means that the client gets ids of rows that do fulfil a single condition of c but do not fulfil c. Therefore, the client has to compute the set of ids according to the operators in c.

Correctness. In order to show the correctness of Protocol $\pi_{q_{select}}$, this means that the Protocol $\pi_{q_{select}}$ correctly executes SELECT queries $q_{select} \in Q_{select}$ for $\mathsf{Enc}_{\mathrm{DB}}(d, K)$, we

have to show that the output of the client is $q_{select}(d)$. This is straightforward due to the construction of the protocol and $\mathrm{Enc}_{\mathrm{DB}}(d, K)$: The tuples selected in Steps 4 and 5, are the tuples that match the conditions of the query q_{select}. After applying the projection Π, the tuples are identical to $q_{select}(d)$.

Efficiency. In order to discuss the efficiency of our protocols, we make the following assumptions:

- The server can access a single tuple in d_{data} in $O(log(n))$, where $n > 1$ is the size of the database d.
- The number $|A|$ of attributes of d as well as the number of possible different attribute values is fixed and independent from n.
- There is an upper bound for the number of conditions in a query that is independent of n.

A consequence from the second assumption is, that the server is able retrieve a tuple in the index tables in $O(1)$. The third assumption is reasonable due the fixed number of attributes and possible attribute values. Figure 7.7 provides an overview of the scaling properties of all protocols presented in this section.

query	single atomic condition	multiple atomic conditions
SELECT	$O(\|q(d)\| \cdot log(n))$	$\max\{O(m), O(\|q(d)\| \cdot log(n))\}$
UPDATE	$O(\|t\| \cdot (log(n) + l))$	$\max\{O(m), O(\|t\| \cdot (log(n) + l))\}$
INSERT	$O(m + l)$	
DELETE	$O(\|t\| \cdot (log(n) + l))$	$\max\{O(m), O(\|t\| \cdot (log(n) + l))\}$

7.7.1: overview

symbol	meaning
m	maximum number an attribute value that is part of the conditions occurs in d
t	the tuples to be deleted/the tuple to be updated
l	time needed to update the server's internal indices

7.7.2: legend

Figure 7.7.: Overview of the scaling properties of the protocols. Note that INSERT queries do not have a condition.

The number of messages in the protocol for SELECT queries is constant. The size of the message in Step 1 depends on the number of conditions in the query. For a SELECT query with one condition, the communication of Step 2 is in $O(|q(d)|)$ and the encrypted list of row ids sent to the client contains the same number of ids than there are tuples in the result set. For a SELECT query with multiple conditions, the communication of Step 2 is in $O(n)$: Consider for example an attribute, that has the same value for all tuples in the database. If this attribute value fulfils an atomic condition, the server sends an encrypted list of all row ids to the client. If, furthermore, no tuple fulfils the condition c of the query, the size of the result set is 0 which is independent of n. Then, the complexity of the SELECT Protocol depends on the size of the database, while size of the result set does

not. Optimisations for the index structures of our scheme that alleviate this effect, are discussed in Chapter 7, Section 7. There, we will present methods, that allow to efficiently ($\in O(log(n))$) select a single row id in an index.

The computation complexity in Step 4 depends linearly on the size of messages in step 3. The computation in Step 5 is in $O(|q(d)| \cdot log(n))$: The server has to retrieve each tuple that is part of the result set. The communication of Steps 5 and 6 as well as the computation of Step 6 is in $O(|q(d)|)$.

Consequently, for SELECT queries with one condition, the the protocol for SELECT queries is in $O(|q(d)| \cdot log(n))$, which can be considered as optimal, since this is a lower bound for retrieving $|q(d)|$ random tuples from a database. For SELECT queries with multiple conditions, the the protocol for SELECT queries is in $\max\{O(m), O(|q(d)| \cdot log(n))\}$, where m is the maximum number an attribute value that is part of the conditions occurs in d.

The following example provides more insight into the SELECT Protocol. Consider the following query q_{select} for the database d depicted in Figure 7.5:

$$\text{SELECT } * \text{ FROM } d \text{ WHERE } name = Alice \text{ AND } surname = Smith$$

The condition of this query (*name* = Alice AND *surname* = Smith) is comprised of two atomic conditions. In the first two steps of Protocol 1, the Client gets all ids of rows that fulfil an atomic condition of the query. This can be realised with the following two queries to the index tables (cf. Figure 7.6):

$$\text{SELECT } rows \text{ FROM } d_{index,name} \text{ WHERE } values = Alice$$

$$\text{SELECT } row \text{ FROM } d_{index,surname} \text{ WHERE } values = Smith$$

After decrypting the values of the *rows* attribute of the results, the client has two sets of tuple ids: $\{1, 3\}$ and $\{1, 2\}$. Since the two atomic conditions are joined with an AND ($\wedge$), the client has to compute $\{1, 3\} \cap \{1, 2\} = \{1\}$. Thus, in the next step, the client gets the tuple with id 1 from the data table:

$$\text{SELECT } data(name, surname) \text{ FROM } d_{data} \text{ WHERE } row \text{ IN } \{1\}$$

After decrypting the result, the protocol ends, since the query q does not contain a non trivial projection ($\Pi = id$) and the client returns the result set $\{(Alice, Smith)\}$.

Protocol for DELETE Queries A DELETE query deletes tuples from a database that match a condition given in the query. The following protocol executes DELETE queries:

Protocol 2 ($\pi_{q_{delete}}$: Protocol for DELETE Queries)**.**
Input of client $\mathfrak{C}$*:* $q_{delete} \in Q_{delete}, K$
Input of server $\mathfrak{S}$*:* $\text{Enc}_{DB}(d, K)$

1. *The client* $\mathfrak{C}$ *sends the conditions* $c = c_1, \ldots c_l$ *of* σ *to the server* $\mathfrak{S}$*.*

2. *For each condition* c_i *with attribute* A_i*, the server* $\mathfrak{S}$ *returns the tuples of* d_{index,A_i} *where* c_i *holds for the attribute* values *to the client* $\mathfrak{C}$*.*

3. *The client $\mathfrak{C}$ decrypts the value of the attribute* rows *of every tuple received by the server $\mathfrak{S}$ and receives for each $c_i \in c$ a set of ids of tuples for which the condition c_i holds.*

4. *The client $\mathfrak{C}$ computes ids_c, the set of ids of tuples for which the condition c holds. If ids_c is empty, client $\mathfrak{C}$ sends $\perp$ to the server $\mathfrak{S}$, the server outputs $\mathrm{Enc}_{DB}(d, K)$ and the protocol ends, otherwise the client $\mathfrak{C}$ sends ids_c to the server $\mathfrak{S}$.*

5. *The server $\mathfrak{S}$ returns the tuples of d_{data} whose ids are in ids_c to the client $\mathfrak{C}$.*

6. *The server $\mathfrak{S}$ deletes the tuples of d_{data} whose ids are in ids_c.*

7. *The client $\mathfrak{C}$ decrypts the received tuples.*

8. *For each attribute $A_j \in A$ and each (unique) value v of the tuples:*

   8.1. *The client $\mathfrak{C}$ sends the condition $A_j = v$ to the server $\mathfrak{S}$.*

   8.2. *The server $\mathfrak{S}$ returns the tuple of d_{index,A_j} where the condition values = v holds to the client $\mathfrak{C}$.*

   8.3. *The client $\mathfrak{C}$ decrypts the rows value of the received tuple and removes all indices from the list that are also in ids_c.*

   8.4a. *If the resulting list is empty:*

      8.4a.1 *The Client $\mathfrak{C}$ sends DELETE to the Server $\mathfrak{S}$.*

      8.4a.2 *The Server deletes the tuple from d_{index,A_i} where the condition values = v holds*

   8.4b. *If the resulting list is not empty:*

      8.4b.1 *The Client $\mathfrak{C}$ encrypts the resulting list and sends it the Server $\mathfrak{S}$.*

      8.4b.2 *The Server $\mathfrak{C}$ updates the value of attribute rows of the tuple from d_{index,A_i} where the condition values = v holds with the encrypted list received from the client.*

Output of client $\mathfrak{C}$: {}
Output of server $\mathfrak{S}$: updated outsourced database

Steps 1-5 are identical to the SELECT protocol. In these steps, the client determines the tuples to be deleted. In the next steps, the tuples are deleted and the index entries are updated. If an index entry is no longer needed, it is also deleted. This protocol can be optimised in terms of rounds of communications. The index tables that are queried in Steps 1 and 2 of the protocol are queried again in the Steps 8.1. and 8.2. In an optimised protocol, the client can check in Step 8.1 if it already received the tuple in Step 2.

Correctness. In order to show the correctness of Protocol $\pi_{q_{delete}}$, this means that the Protocol $\pi_{q_{delete}}$ correctly executes DELETE queries $q_{delete} \in Q_{delete}$ for $\mathrm{Enc}_{\mathrm{DB}}(d, K)$, we have to show that the output of the server is $\mathrm{Enc}_{\mathrm{DB}}(d_{q_{delete}}, K)$. The encrypted database $\mathrm{Enc}_{\mathrm{DB}}(d_{q_{delete}}, K)$ is an encryption of the database d without the tuples deleted by the query q_{delete}. This means that the index tables of $\mathrm{Enc}_{\mathrm{DB}}(d_{q_{delete}}, K)$ do not contain entries for the deleted tuples and the data table of $\mathrm{Enc}_{\mathrm{DB}}(d_{q_{delete}}, K)$ does not contain the deleted tuples. In Step 8.4 of the Protocol $\pi_{q_{delete}}$, the index entries of the tuples to be deleted are

removed from the index tables. In Step 6, the tuples to be deleted are removed from the data table.

Efficiency. Let t be the tuples to be deleted, let m be the maximum number of times that an attribute value that is part of the conditions occurs in d. Furthermore, let updating the server's internal indices be in $O(I)$.

Steps 1-5 are identical to the SELECT Protocol ($\in \max\{O(m), O(|t| \cdot log(n))\}$). Step 6 is in $O(|t| \cdot (log(n) + I))$, since the server has to search $|t|$ times for a tuple in order to delete all tuples to be deleted and update its internal indices accordingly. The number of loops in Step 8 depends the number of attributes of d and the number of unique attribute values of each attribute, which are independent from n. The complexity of Steps 8.1 - 8.4 is in $\max\{O(m), O(I)\}$, since the encrypted tuple returned by the server in the worst case contains m row ids.

Therefore, the DELETE Protocol is in $\max\{O(m), O(|t| \cdot (log(n) + I))\}$. For DELETE queries with a single condition, since then $O(m) = O(|t|)$, the DELETE Protocol is in $O(|t| \cdot (log(n) + I))$, which can be considered as optimal, since this is a lower bound for deleting $|t|$ random tuples from a database and updating the indices accordingly.

As an example for the execution of a DELETE query consider the following query:

$$\text{DELETE FROM } d \text{ WHERE } surname = \mathit{Jones}$$

In order to execute the protocol for this query on our example data set $\mathsf{Enc}_{\mathrm{DB}}(d, K)$, the client queries the server for tuples, that fulfil the condition $surname = Jones$. Therefore, the client first has to query the corresponding index table:

$$\text{SELECT } rows \text{ FROM } d_{index,surname} \text{ WHERE } values = \mathit{Jones}$$

After encrypting the result, the client can query the tuples to be deleted, and instruct the server to delete the tuples.

$$\text{SELECT } data(name, surname) \text{ FROM } d_{data} \text{ WHERE } row \text{ IN } \{3\}$$

$$\text{DELETE FROM } d_{data} \text{ WHERE } row \text{ IN } \{3\}$$

Now, the client has to update the indices. The client retrieves the relevant indices with the following queries:

$$\text{SELECT } rows \text{ FROM } d_{index,surname} \text{ WHERE } values = \mathit{Jones}$$

$$\text{SELECT } rows \text{ FROM } d_{index,name} \text{ WHERE } values = Alice$$

Since the client already queried $d_{index,surname}$ for the value *Jones*, the first of these two queries can be omitted. Now, the client has to decrypt the index lists, remove the id 3 from them, encrypt the results and send them to the server:

$$\text{DELETE FROM } d_{index,surname} \text{ WHERE } values = \mathit{Jones}$$

$$\text{UPDATE } d_{index,name} \text{ SET } rows = \mathsf{Enc}_I(1,K) \text{ WHERE } values = Alice$$

Protocol for INSERT Queries An INSERT query contains a tuple t (cf. Definition 61). Since we only consider INSERT queries with the same attributes as the database (correct INSERT queries), we omit checking for correctness in Step 1. The following protocol executes insert queries:

Protocol 3 ($\pi_{q_{insert}}$: Protocol for INSERT Queries)**.**
Input of client $\mathfrak{C}$*:* $q_{insert} \in Q_{insert}$, K
Input of server $\mathfrak{S}$*:* $\mathsf{Enc}_{DB}(d, K)$
Let t *be the tuple in* q_{insert}*,* A *be the attributes of* t *and* d*.*

1. *The Client* $\mathfrak{C}$ *sends* $\mathsf{Enc}_I(t, K)$ *to the server* $\mathfrak{S}$*.*
2. *The Server* $\mathfrak{S}$ *appends* d_{data} *with* $\mathsf{Enc}_I(t, K)$ *and sends the row id* r *of* $\mathsf{Enc}_I(t, K)$ *to the client* $\mathfrak{C}$*.*
3. *For each attribute* $A_i \in A$
   - 3.1. *The client* $\mathfrak{C}$ *sends the condition* $A_i = t_i$ *to the server* $\mathfrak{S}$*.*
   - 3.2. *The server* $\mathfrak{S}$ *returns the tuple of* d_{index,A_i} *where the condition values* $= t_i$ *holds to the client* $\mathfrak{C}$*.*
   - 3.3a. *If the client received a non empty tuple:*
     - 3.3a.1. *The client* $\mathfrak{C}$ *decrypts the rows value of the tuple, adds the row id* r *to the list, encrypts the list, and sends it to the server* $\mathfrak{S}$*.*
     - 3.3a.2. *The Server* $\mathfrak{S}$ *replaces the value of attribute* rows *of the tuple in* d_{index,A_i} *where the condition values* $= t_i$ *holds with the encrypted list received by the client.*
   - 3.3b. *If the client received an empty tuple:*
     - 3.3b.1. *The client* $\mathfrak{C}$ *sends* $\mathsf{Enc}_I(r, K)$ *to the server* $\mathfrak{S}$*.*
     - 3.3b.2. *The server* $\mathfrak{S}$ *appends* d_{index,A_i} *with the tuple* $(t_i, \mathsf{Enc}_I(r, K))$

Output of client $\mathfrak{C}$*:* r
Output of server $\mathfrak{S}$*:* $\mathsf{Enc}_{DB}(d_{q_{insert}}, K)$

In this protocol, the client encrypts the tuple to be inserted and sends it to the server. The server inserts the encrypted tuple into the data table. Then, for each attribute value, the client updates the corresponding index table. If the attribute value did not occur previously in the database d, a new index entry has to be created.

Correctness. In order to show the correctness of Protocol $\pi_{q_{insert}}$, this means that the Protocol $\pi_{q_{insert}}$ correctly executes INSERT queries $q_{delete} \in Q_{insert}$ for $\mathsf{Enc}_{\mathrm{DB}}(d, K)$, we have to show that the output of the client is the row id of the inserted tuple and the output of the server is $\mathsf{Enc}_{\mathrm{DB}}(d_{q_{insert}}, K)$. The row ids in d are the same ids as the row ids of the corresponding tuples in d_{data} inserting a new tuple into d_{data} (Step 2) returns the same row id as inserting a new tuple into d. Therefore the client returns the same row id as the query $q_{insert}(d)$. The encryption $\mathsf{Enc}_{\mathrm{DB}}(d_{q_{insert}}, K)$ is the encryption of d after inserting tuple t. After executing Protocol $\pi_{q_{insert}}$, the data table is appended with $(r, \mathsf{Enc}_{\mathrm{I}}(t, K))$, therefore the new data table is (modulo the randomness used in $\mathsf{Enc}_{\mathrm{I}}$) identical to the data table of $\mathsf{Enc}_{\mathrm{DB}}(d_{q_{insert}}, K)$. Since in Step 3, the index tables are updated accordingly, the same is correct for the index tables.

Efficiency. Let m be the maximum number an attribute value that occurs in t also occurs in d (in the corresponding column). Furthermore, let updating the server's internal indices be in $O(l)$.

The complexity of Step 1 linearly depends on the number of attributes. Step 2 is in $O(l)$, since the server has to update its internal indices after inserting a tuple. The number of loops in Step 3 is constant. Steps 3.1 - 3.3a are in $O(m)$, since the encrypted list returned by the server in the worst case contains m row ids. Step 3.3b is in $O(l)$, since the server adds a new tuple. Therefore, the INSERT Protocol is in $O(m + l)$.

As an example for the execution of an INSERT query, consider the following query q_{insert} for our database in Figure 7.5:

$$\text{INSERT INTO } d \text{ VALUES } \textit{(Carol, Jones)}. \tag{7.1}$$

The new database $d_{q_{insert}}$ is depicted in Figure 7.8. Executing Protocol 3 for query q_{insert}

name	surname
Alice	Smith
Bob	Smith
Alice	Jones
Carol	Jones

Figure 7.8.: A simple example database $d_{q_{\text{insert}}}$ from Figure 7.5 after application of the INSERT query Query 7.1. Figure 7.9 depicts the result of the transformation Enc_{DB} of this database.

and $\text{Enc}_{\text{DB}}(d, K)$ can be done as follows: In the first step, the client encrypts the tuple given by the query and sends it to the server to be appended to the data table:

$$\text{INSERT INTO } d_{data}(data\,(name, surname)) \text{ VALUES } (\text{Enc}_I\ \textit{((Carol, Jones), K))}$$

After this step, the server returns the row id 4. With this new row id, the client can update the index tables. The client retrieves the row ids for attribute value *Carol* of attribute *name* with:

$$\text{SELECT } rows \text{ FROM } d_{index,name} \text{ WHERE } values = Carol$$

Since there is no index entry for this value, the client has to create one:

$$\text{INSERT INTO } d_{index,name} \text{ VALUES } (Carol, \text{Enc}_I\ (4, K))$$

There is an index entry for the surname *Jones*. Thus, the client can update it:

$$\text{SELECT } rows \text{ FROM } d_{index,surname} \text{ WHERE } values = Jones$$

$$\text{UPDATE } d_{index,surname} \text{ SET } rows = \text{Enc}_I\ ((3, 4), K) \text{ WHERE } values = Jones$$

Figure 7.9 depicts the outsourced database after the execution of the insert protocol for our query q_{insert}. Compared to the tables in Figure 7.6, the data table, as well as the index table of the attribute *name* has a new entry. The index table of the attribute *surname* has an updated entry.

row	data (name,surname)
1	$\mathsf{Enc}_{\mathrm{I}}$ ((Alice, Smith), K)
2	$\mathsf{Enc}_{\mathrm{I}}$ ((Bob, Smith), K)
3	$\mathsf{Enc}_{\mathrm{I}}$ ((Alice, Jones), K)
4	$\mathsf{Enc}_{\mathrm{I}}$ ((Carol, Jones), K)

7.9.1: d'_{data}

values	rows
Alice	$\mathsf{Enc}_{\mathrm{I}}$ ((1, 3), K)
Bob	$\mathsf{Enc}_{\mathrm{I}}$ (2, K)
Carol	$\mathsf{Enc}_{\mathrm{I}}$ (4, K)

7.9.2: $d'_{index,name}$

values	rows
Smith	$\mathsf{Enc}_{\mathrm{I}}$ ((1, 2), K)
Jones	$\mathsf{Enc}_{\mathrm{I}}$ ((3, 4), K)

7.9.3: $d'_{index,surname}$

Figure 7.9.: The database $\mathsf{Enc}_{\mathrm{DB}}(d_{q_{\mathrm{insert}}}, K)$. For the sake of readability, we set $d' = d_{q_{\mathrm{insert}}}$.

Protocol for UPDATE Queries The UPDATE Protocol is the most complex one. UPDATE queries can be simulated by a SELECT query, a DELETE query, and several INSERT queries. The following protocol, however, executes UPDATE queries more efficiently:

Protocol 4 ($\pi_{q_{update}}$: Protocol for UPDATE Queries)**.**
Input of client $\mathfrak{C}$*:* $q_{update} \in Q_{update}$, K
Input of server $\mathfrak{S}$*:* $\mathsf{Enc}_{DB}(d, K)$
Let σ *be the selection,* A_q *the attribute, and* v *be the attribute value in* q_{update}*.*

1. *The client* $\mathfrak{C}$ *sends the conditions* $c = c_1, \ldots c_l$ *of* σ *to the server* $\mathfrak{S}$*.*
2. *For each condition* c_i *with attribute* A_i*, the server* $\mathfrak{S}$ *returns the tuples of* d_{index,A_i} *where* c_i *holds for the attribute* values *to the client* $\mathfrak{C}$*.*
3. *The client* $\mathfrak{C}$ *decrypts the value of the attribute* rows *of every tuple received by the server* $\mathfrak{S}$ *and receives for each* $c_i \in c$ *a set of ids of tuples for which the condition* c_i *holds.*
4. *The client* $\mathfrak{C}$ *computes* ids_c*, the set of ids of tuples for which the condition* c *holds. If* ids_c *is empty, client* $\mathfrak{C}$ *sends* $\perp$ *to the server* $\mathfrak{S}$*, the server outputs* $\mathsf{Enc}_{DB}(d, K)$ *and the protocol ends, otherwise the client* $\mathfrak{C}$ *sends* ids_c *to the server* $\mathfrak{S}$*.*
5. *The server* $\mathfrak{S}$ *returns the tuples of* d_{data} *whose ids are in* ids_c *to the client* $\mathfrak{C}$*.*
6. *The client* $\mathfrak{C}$ *decrypts the encrypted tuples in the result received by the server.*
7. *For each unique attribute value* $v_i \neq v$ *of attribute* A_q *in the decrypted tuples:*
   - 7.1. *The client* $\mathfrak{C}$ *sends the condition* $A_q = v_i$ *to the server* $\mathfrak{S}$*.*
   - 7.2. *The server* $\mathfrak{S}$ *returns the tuple of* d_{index,A_q} *where values* $= v_i$ *holds.*
   - 7.3. *The client* $\mathfrak{C}$ *decrypts the list of row ids received by the server.*
   - 7.4. *The client* $\mathfrak{C}$ *removes all ids from the list that are also in* ids_c*.*
   - 7.5a. *If the resulting list is empty:*
     - 7.5a.1. *The Client* $\mathfrak{C}$ *sends DELETE to the Server* $\mathfrak{S}$*.*
     - 7.5a.2. *The Server deletes the tuple from* d_{index,A_i} *where the condition values* $= v_i$ *holds*
   - 7.5b. *If the resulting list is not empty:*
     - 7.5b.1. *The Client* $\mathfrak{C}$ *encrypts the resulting list and sends it the Server* $\mathfrak{S}$*.*

7.5b.2. *The Server $\mathfrak{S}$ updates the value of attribute rows of the tuple from d_{index,A_i} where the condition values = v_i holds with the encrypted list received from the client.*

8. *The client $\mathfrak{C}$ sends the condition $A_q = v$ to the server $\mathfrak{S}$.*

9. *The server $\mathfrak{S}$ returns the tuple of d_{index,A_i} where the condition values = v holds to the client $\mathfrak{C}$.*

10a. *If the client received a non empty tuple:*

10a.1. *The client $\mathfrak{C}$ decrypts the rows value of the tuple, adds the row ids ids_c, that are not already in the list, to the list, encrypts the list, and sends it to the server $\mathfrak{S}$.*

10a.2. *The Server $\mathfrak{S}$ replaces the value of attribute* rows *of the tuple in d_{index,A_i} where the condition values = v holds with the encrypted list received by the client.*

10b. *If the client received an empty tuple:*

10b.1. *The client $\mathfrak{C}$ sends $(v, \mathsf{Enc}_I(ids_c, K))$ to the server $\mathfrak{S}$.*

10b.2. *The server $\mathfrak{S}$ appends d_{index,A_q} with the tuple $(v, \mathsf{Enc}_I(ids_c, K))$*

11. *The client $\mathfrak{C}$ changes each value of attribute A_q of the decrypted tuples to v.*

12. *The client $\mathfrak{C}$ encrypts the tuples and sends them to the server $\mathfrak{S}$.*

13. *The server $\mathfrak{S}$ replaces the encrypted values of the tuples of d_{data} sent to the client in Step 5 with the encrypted tuples received by the client.*

Output of client $\mathfrak{C}$: {}
Output of server $\mathfrak{S}$: updated outsourced database

Correctness. In order to show the correctness of Protocol $\pi_{q_{update}}$, this means that the Protocol $\pi_{q_{update}}$ correctly executes UPDATE queries $q_{update} \in Q_{update}$ for $\mathsf{Enc}_{\mathrm{DB}}(d, K)$, we have to show that the output of the server is $\mathsf{Enc}_{\mathrm{DB}}(d_{q_{update}}, K)$. The database $d_{q_{update}}$ is the database d after changing each value of attribute A_q of each tuple fulfilling the conditions of the query to v. In the Steps 1-6, the client determines the tuples to be changed by the query q_{update}. In Step 7, the client updates the index entries for the old values. If an index entry is no longer needed (if the list of row ids is empty), it is deleted. In Steps 8-10, the client updates the index entry for the new attribute value v. If there is no index entry for attribute A_q and value v, one is created. In Steps 11-13, the data table is updated with the updated tuples.

This protocol can be optimised similarly to the DELETE Protocol: by checking in Step 7.1, if the client already received d_{index,A_q} is Step 2. Additionally, in Step 12, the client can only send encrypted tuples to the server that were changed in Step 11.

Efficiency. Let updating the server's internal indices be in $O(l)$, let t be the tuples affected by the UPDATE query, and let m be the maximum number of times that an attribute value that is part of the conditions occurs in d.

Steps 1 - 5 are identical to Steps 1 - 5 of the SELECT Protocol ($\in \max\{O(m), O(|t| \cdot log(n))\}$) The number of loops in Step 7 depends on the number of unique attribute values of attribute A_q, the attribute in the selection of the query q_{update}. Steps 7.1 - 7.4 are in

$O(m)$, since the encrypted list returned by the server in the worst case contains m row ids. Step 7.5a is in $O(l)$, since the server has to insert a tuple. Step 7.5b, again, is in $O(m)$. Steps 8 - 10 are in the same complexity class as Step 7. Steps 11 and 12 is in $O(|t|)$, since the client changes a value of every tuple in t, and sends the result to the server after encrypting it. In Step 13, the server updates |t| tuples in d_{data}. therefore, Step 13 is in $O(|t| \cdot l)$

Consequently, the UPDATE Protocol is in $\max\{O(m), O(|t| \cdot (log(n) + l))\}$. As with the SELECT, and the DELETE Protocol, for queries with a single condition, where $O(m) = O(|t|)$, the UPDATE Protocol is in $O(|t| \cdot (log(n) + l))$. This also can be considered as optimal since this is a lower bound for updating $|t|$ random tuples in a database and updating the indices accordingly.

Definition of the MimoSecco Database Outsourcing Scheme With the definition of $\mathsf{Enc}_{\mathrm{DB}}$ and the definitions of the protocols $\pi_{q_{select}}$, $\pi_{q_{delete}}$, $\pi_{q_{insert}}$, and $\pi_{q_{update}}$, we can define our outsourcing scheme.

Definition 70 (MimoSecco Database Outsourcing Scheme)**.** *Let* $(\mathsf{Gen}_I, \mathsf{Enc}_I, \mathsf{Dec}_I)$ *be an encryption scheme with* IND*-CPA security. We define the* MimoSecco database outsourcing scheme *as* $(\mathsf{Gen}_{DB}, \mathsf{Enc}_{DB}, \mathsf{Q}_{DB})$ *with:*

- $\mathsf{Gen}_{DB} = \mathsf{Gen}_I$
- Enc_{DB} *as defined in Definition 69*
- $\mathsf{Q}_{DB} = \pi_{q_{select}} \cup \pi_{q_{delete}} \cup \pi_{q_{insert}} \cup \pi_{q_{update}}$ *as defined in Protocols 1 - 4*

4.2. The MimoSecco Database Outsourcing Scheme has IND-ICP Security

In the last section, we define the MimoSecco database outsourcing scheme. In Definition 53 in Chapter 6, Section 2, we defined IND-ICP, a security notion for database outsourcing. In this section, we prove that MimoSecco database outsourcing scheme provides IND-ICP security.

Theorem 10. *The MimoSecco database outsourcing scheme has* IND*-ICP security.*

In the following proof, we reduce a successful adversary on the MimoSecco database outsourcing scheme to a successful adversary to the underlying IND-CPA secure encryption scheme.

Proof. We prove Theorem 10 by contradiction. We assume there is an adversary $\mathcal{A}_{\text{IND-ICP}}$, that has non negligible advantage over guessing in experiment $\text{IND-ICP}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}$ (Security Game 14). Furthermore, we assume that there is no adversary $\mathcal{A}$ that has non negligible advantage over guessing in experiment $\text{IND-MULT-CPA}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}$ (Security Game 2). Recall that the security notions IND-CPA and IND-MULT-CPA imply each other (cf. Theorem 1).

In the following, we provide an efficient reduction from $\mathcal{A}_{\text{IND-ICP}}$. Consider Figure 7.10. In the first step, we forward the length of the security parameter 1^k to the adversary $\mathcal{A}_{\text{IND-ICP}}$. Our reduction $\mathcal{A}$ has oracle access to $\mathsf{Enc}_{\mathrm{I}}(\cdot, K)$. We use this oracle, to simulate an oracle for $\mathsf{Enc}_{\mathrm{DB}}(\cdot, K)$ (cf. Definition 69) with the internal encryption mechanism

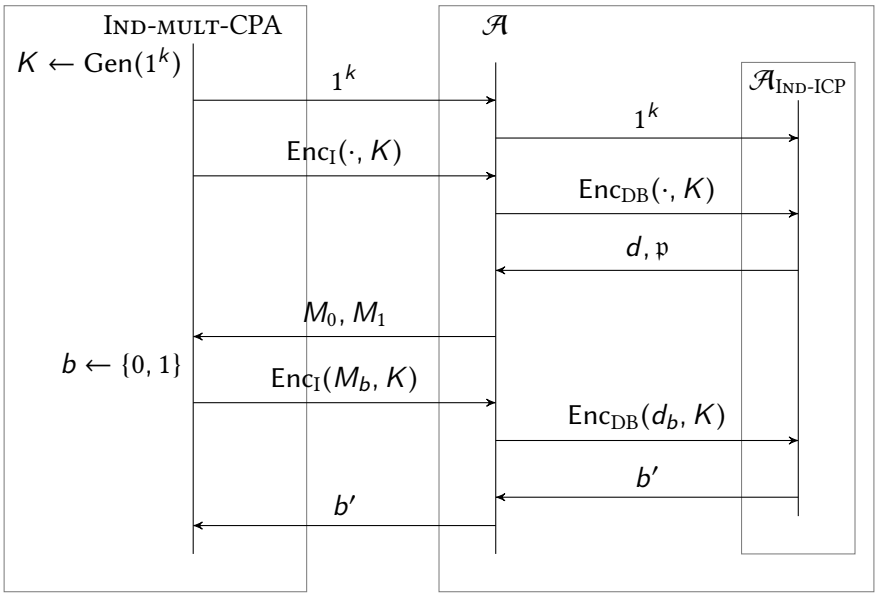


Figure 7.10.: Sketch of the proof for Theorem 10: An efficient reduction of an adversary $\mathcal{A}_{\text{IND-ICP}}$ that breaks IND-ICP to an adversary $\mathcal{A}$ that breaks IND-MULT-CPA.

$\mathrm{Enc}_{\mathrm{I}}$ and give the adversary $\mathcal{A}_{\text{IND-ICP}}$ access to it. Now, the adversary $\mathcal{A}_{\text{IND-ICP}}$ returns a database d and an independent column permutation $\mathfrak{p}$. We set $d_0 = d$ and $d_1 = \mathfrak{p}(d)$ and return two vectors M_0 and M_1 of plaintexts defined as follows:

$$M_b = \begin{pmatrix} d_b(\cdot, 1) \\ \vdots \\ d_b(\cdot, n) \\ \mathcal{R}(d_b, 1, v_{1,1}) \\ \vdots \\ \mathcal{R}(d_b, 1, v_{1,l_1}) \\ \vdots \\ \mathcal{R}(d_b, m, v_{m,1}) \\ \dots \\ \mathcal{R}(d, m, v_{m,l_m}) \end{pmatrix}$$

Where:

- $n = |d_0| = |d_1|$ is the size of d.
- $m = |A|$ is the number of attributes of d.
- l_i is the number of (different) attribute values of attribute A_i in d.
- $v_{j,l_k} = \mathcal{V}(d(\cdot, j))_{l_k}$ ($\mathcal{V}(d(\cdot, j))$ is the set of all values of column j, cf. Definition 65.)

The vector M_b contains the plaintexts of all encrypted entries of $\mathrm{Enc}_{\mathrm{DB}}(d_b, K)$ (cf. Definition 69). The experiment returns the challenge $\mathrm{Enc}_{\mathrm{I}}(M_b, K)$, an element-wise encryption of M_b. Since $\mathrm{Enc}_{\mathrm{I}}(M_b, K)$ contains all encrypted entries of $\mathrm{Enc}_{\mathrm{DB}}(d_b, K)$, we use $\mathrm{Enc}_{\mathrm{I}}(M_b, K)$ to construct $\mathrm{Enc}_{\mathrm{DB}}(d_b, K)$, the challenge for the adversary $\mathcal{A}_{\text{IND-ICP}}$. The adversary returns her guess b', which we return to the experiment. We win the

security game if $b' = b$. Therefore, we inherit the success probability of the adversary $\mathcal{A}_{\text{IND-ICP}}$, and get a successful adversary for IND-MULT-CPA, which is a contradiction to our assumption. □

5. A Database Outsourcing Scheme with /-IND-ICP Security in the Presence of Queries

The MimoSecco database outsourcing scheme defined in the previous section only provides static security. If the adversary is given access to the messages sent to the server, she can break static security: Consider the example database d from Section 4 depicted in Figure 7.5 (restated in this section in Figure 7.11) and $\text{Enc}_{\text{DB}}(d, K)$ depicted in Figure 7.6 (restated in this section in Figure 7.12). Furthermore, consider a SELECT query with the

name	surname
Alice	Smith
Bob	Smith
Alice	Jones

Figure 7.11.: A simple example database d from Figure 7.5.

conditions $name = Alice \wedge surname = Smith$. During execution of this query (cf. Protocol 1), the client sends these conditions to the server (Step 1). In Step 4, the client sends all row ids of tuples that fulfil these conditions to the server. In our example, this is row id 1. Now, the server learns that (only) the encrypted tuple in row 1 of the data table contains the attribute values *Alice* and *Smith*. Therefore, an adversary given access to the view of the server can break IND-ICP. In this simple example, the view of the server for one query is sufficient to reconstruct the original database.

An obvious solution for this problem is to alter all messages sent from the client to the server that depend on the actual data in the outsourced database. For the SELECT protocol, this is the message sent in Step 4.

By adding *dummy row ids* to the row ids sent in this step, we can prevent the adversary from breaking IND-ICP. If the number of row ids sent to the server is identical for all independent column permutations $\mathfrak{p}(d)$ of d, the adversary can not use this step to distinguish the database $\text{Enc}(d, K)$ from $\text{Enc}(\mathfrak{p}(d), K)$. For our example database the client needs to send 2 row ids, since there is an independent column permutation of d that has two tuples with the attribute values *Alice* and *Smith*. The client can compute this number with the row ids received in in Step 3 of the SELECT Protocol and add and additional row id to the message to the server.

This alteration of the select protocol provides IND-ICP security in the presence of one query. If the adversary is allowed to observe additional queries, however, she can break IND-ICP: In our example, the client has two possibilities for the needed dummy id in Step 3. Either the client sends the ids $\{1, 2\}$ or the client sends the ids $\{1, 3\}$. If the client sent $\{1, 2\}$, the adversary knows that if there are two tuples with attribute values *Alice* and *Smith* in the database, they are in rows 1 and 2 of d_{data}. Now, if the adversary observes the execution of a SELECT query with the condition $name = Alice$, she knows that the tuples with attribute value *Alice* are in rows 1 and 3 of d_{data}. Together with the static view of the

row	data (name,surname)
1	$\mathsf{Enc}_{\mathrm{I}}((\mathit{Alice}, \mathit{Smith}), K)$
2	$\mathsf{Enc}_{\mathrm{I}}((\mathit{Bob}, \mathit{Smith}), K)$
3	$\mathsf{Enc}_{\mathrm{I}}((\mathit{Alice}, \mathit{Jones}), K)$

7.12.1: d_{data}

values	rows
Alice	$\mathsf{Enc}_{\mathrm{I}}((1, 3), K)$
Bob	$\mathsf{Enc}_{\mathrm{I}}(2, K)$

7.12.2: $d_{index,name}$

values	rows
Smith	$\mathsf{Enc}_{\mathrm{I}}((1, 2), K)$
Jones	$\mathsf{Enc}_{\mathrm{I}}(3, K)$

7.12.3: $d_{index,surname}$

Figure 7.12.: The example tables $\mathsf{Enc}_{\mathrm{DB}}(d, K)$ from Figure 7.6.

index tables, now, the adversary can reconstruct the original database. If the client sent $\{1, 3\}$, the adversary knows that if there are two tuples with attribute values *Alice* and *Smith* in the database, they are in rows 1 and 3 of d_{data}. Now, if the adversary observes the execution of a SELECT query with the condition *surname = Smith*, she knows that the tuples with attribute value *Smith* are in rows 1 and 2 of d_{data}. Together with the static view of the index tables, the adversary, again, can reconstruct the original database.

This example shows that there is a difference between security notions for outsourced data sets that allow the adversary to observe the execution of one query and notions that allow the adversary to observe the execution of two or more queries. Such notions can be captured by the generalised notions of our framework defined in Chapter 5, Section 6. Since an outsourced database that can only be queried a single time is of limited use, we omit a formal definition of the scheme and security proofs.

A trivial scheme that provides security for arbitrary many queries is one that for each query downloads the outsourced database, decrypts it, and executes the query locally. This, however, induces a huge communication overhead, especially for large databases.

In the following, we define a scheme that provides no query advantage over l-Ind-ICP (cf. Definitions 57 and 58 in Chapter 6). This scheme is based on the MimoSecco database outsourcing scheme. Since the notion Ind-ICP implies the notion l-Ind-ICP, we do not need to alter the mechanism $\mathsf{Enc}_{\mathrm{DB}}$. Therefore, we only alter the protocols of the MimoSecco database outsourcing scheme Since the notion no query advantage over l-Ind-ICP is defined for databases of fixed size, we only consider the protocols for SELECT and UPDATE queries.

Note that database outsourcing schemes that allow to change the size of the database by allowing INSERT or DELETE queries introduce a significant overhead. The adversary must not be able to distinguish two outsourced databases that have been queried multiple times by their size. For example, a row removed by a DELETE query has to be replaced by dummy data if, in the security game, the adversary can choose a database, where this DELETE query does not remove a row.

In order to provide no query advantage over l-Ind-ICP we alter the protocols to operate on buckets instead of single tuples. For the SELECT Protocol, we need to alter Step 4. Here, instead of sending the row ids of tuples that fulfil the conditions of the query, the client has to send the ids of the buckets that potentially contain tuples that fulfil the conditions. Formally, we define potentially containing tuples that fulfil a condition as follows:

Definition 71 (Fulfilling a Condition under icp)**.** *A database d fulfils a condition under icp, iff there is an icp $\mathfrak{p}$ such that there is at least one tuple $t \in \mathfrak{p}(d)$ that fulfils the condition. A partition of a database d fulfils a condition c under icp, iff there is an icp $\mathfrak{p}$ that respects*

the partitioning scheme of d such that there is at least one tuple $t \in \mathfrak{p}(d)$ that fulfils the condition c.

Additionally, we need to introduce a partitioning scheme with partitions of size l to our database. This can, for example, be implemented by introducing an additional attribute to the index tables an the data table. Consider for example Figure 7.13. Here, we introduced

values	rows (surname)	buckets (surname)
Alice	Enc_I ((1,2), K)	1,2
Bob	Enc_I (3, K)	2
Carol	Enc_I (4, K)	1

7.13.1: $d_{\text{index,surname}}$

values	rows (name)	buckets (name)
Jones	Enc_I ((2,3,4), K)	1,2
Smith	Enc_I (1, K)	1

7.13.2: $d_{\text{index,name}}$

row	bucket	data (name,surname)
1	1	Enc_I ((Alice, Smith), K)
2	2	Enc_I ((Alice, Jones), K)
3	2	Enc_I ((Bob, Jones), K)
4	1	Enc_I ((Carol, Jones), K)

7.13.3: d_{data}

Figure 7.13.: An outsourced database d with buckets introduced to the data table and to the index tables. Here, the size of the buckets is 2.

bucket ids to the data table and the index tables. This, however, implies an adapted mechanism Enc_DB. For the sake of simplicity, in the following protocols, we assume that there is a partitioning scheme with partitions of size l that is known to the client and to the server.

Protocol 5 ($\pi^{l\text{-IND-ICP}}_{q_{select}}$: Protocol for SELECT Queries)**.**
Input of client $\mathfrak{C}$: $q_{select} \in Q_{select}$, K
Input of server $\mathfrak{S}$: $\mathsf{Enc}_{DB}(d, K)$
Let σ be the selection of q_{select}, Π be the projection of q_{select}, and c be the conditions of σ.

1. *The client $\mathfrak{C}$ sends the conditions $c = c_1, \ldots c_j$ of σ to the server $\mathfrak{S}$.*

2. *For each condition c_i with attribute A_i, the server $\mathfrak{S}$ returns the tuples of d_{index,A_i} where c_i holds for the attribute* values *to the client $\mathfrak{C}$.*

3. *The client $\mathfrak{C}$ decrypts the value of the attribute* rows *of every tuple received by the server $\mathfrak{S}$ and receives for each $c_i \in c$ a set of ids of rows and a set of ids of buckets that contain tuples for which the condition c_i holds.*

4. *The client $\mathfrak{C}$ computes ids_c, the set of ids of tuples for which the condition c holds and $pids_c$, the set of ids of buckets that fulfil c under icp. If $pids_c$ is empty, client $\mathfrak{C}$ sends $\perp$ to the server $\mathfrak{S}$, the client outputs an empty database and the protocol ends, otherwise the client $\mathfrak{C}$ sends $pids_c$ to the server $\mathfrak{S}$.*

5. *The server $\mathfrak{S}$ returns the partitions of d_{data} whose ids are in $pids_c$ to the client $\mathfrak{C}$.*

6. *The client $\mathfrak{C}$ decrypts the received tuples and applies the selection σ and the projection Π to them (and returns them).*

Output of client $\mathfrak{C}$: projection Π of selection σ of received tuples
Output of server $\mathfrak{S}$: $\mathsf{Enc}_{DB}(d, K)$

This protocol is similar to the SELECT Protocol of the MimoSecco database outsourcing scheme in Section 4. It operates on buckets instead of tuples. Therefore, the client has to sort out the false positive tuples in Step 6 by applying the selection σ. Note that it is possible that the list ids_c is empty, while the list $pids_c$ is not empty. This means, that there are buckets, that fulfil the conditions under icp but no tuples that fulfil them. In this case, in Step 6, the client returns an empty database.

The modified UPDATE Protocol also operates on buckets:

Protocol 6 ($\pi^{l\text{-IND-ICP}}_{q_{update}}$: Protocol for UPDATE Queries)**.**
Input of client $\mathfrak{C}$: $q_{update} \in Q_{update}$, K
Input of server $\mathfrak{S}$: $\mathsf{Enc}_{DB}(d, K)$
Let σ be the selection, A_q the attribute, and v be the attribute value in q_{update}.

1. *The client $\mathfrak{C}$ sends the conditions $c = c_1, \ldots c_j$ of σ to the server $\mathfrak{S}$.*

2. *For each condition c_i with attribute A_i, the server $\mathfrak{S}$ returns the tuples of d_{index,A_i} where c_i holds for the attribute* values *to the client $\mathfrak{C}$.*

3. *The client $\mathfrak{C}$ decrypts the value of the attribute* rows *of every tuple received by the server $\mathfrak{S}$ and receives for each $c_i \in c$ a set of ids of rows and a set of ids of buckets that contain tuples for which the condition c_i holds.*

4. *The client $\mathfrak{C}$ computes ids_c, the set of ids of tuples for which the condition c holds and $pids_c$, the set of ids of buckets that fulfil c under icp. If $pids_c$ is empty, client $\mathfrak{C}$ sends $\perp$ to the server $\mathfrak{S}$, the server outputs $\mathsf{Enc}_{DB}(d, K)$ and the protocol ends, otherwise the client $\mathfrak{C}$ sends $pids_c$ to the server $\mathfrak{S}$.*

5. *The server $\mathfrak{S}$ returns the buckets of d_{data} whose ids are in $pids_c$ to the client $\mathfrak{C}$.*

6. *The client $\mathfrak{C}$ decrypts the encrypted tuples in the result received by the server.*

7. *For each unique attribute value $v_i \neq v$ of attribute A_q in the decrypted tuples:*
   - 7.1. *The client $\mathfrak{C}$ sends the condition $A_q = v_i$ to the server $\mathfrak{S}$.*
   - 7.2. *The server $\mathfrak{S}$ returns the tuple of d_{index,A_q} where values $= v_i$ holds.*
   - 7.3. *The client $\mathfrak{C}$ decrypts the list of row ids received by the server.*
   - 7.4. *The client $\mathfrak{C}$ removes all ids from the list that are also in ids_c.*
   - 7.5. *The Client $\mathfrak{C}$ encrypts the resulting list and sends it the Server $\mathfrak{S}$.*

7.6. *The Server* $\mathfrak{C}$ *updates the value of attribute rows of the tuple from* d_{index,A_i} *where the condition values* $= v_i$ *holds with the encrypted list received from the client.*

8. *The client* $\mathfrak{C}$ *sends the condition* $A_q = v$ *to the server* $\mathfrak{S}$.

9. *The server* $\mathfrak{S}$ *returns the tuple of* d_{index,A_i} *where the condition values* $= v$ *holds to the client* $\mathfrak{C}$.

10a. *If the client received a non empty tuple:*

10a.1. *The client* $\mathfrak{C}$ *decrypts the rows value of the tuple, adds the row ids* ids_c, *that are not already in the list, to the list, encrypts the list, and sends it to the server* $\mathfrak{S}$.

10a.2. *The Server* $\mathfrak{S}$ *replaces the value of attribute* rows *of the tuple in* d_{index,A_i} *where the condition values* $= v$ *holds with the encrypted list received by the client.*

10b. *If the client received an empty tuple:*

10b.1. *The client* $\mathfrak{C}$ *sends* $(v, \mathsf{Enc}_I(ids_c, K))$ *to the server* $\mathfrak{S}$.

10b.2. *The server* $\mathfrak{S}$ *appends* d_{index,A_q} *with the tuple* $(v, \mathsf{Enc}_I(ids_c, K))$

11. *The client* $\mathfrak{C}$ *changes each value of attribute* A_q *of the decrypted tuples that fulfil the conditions* c *to* v.

12. *The client* $\mathfrak{C}$ *encrypts the tuples and sends them to the server* $\mathfrak{S}$.

13. *The server* $\mathfrak{S}$ *replaces the encrypted values of the tuples of* d_{data} *sent to the client in Step 5 with the encrypted tuples received by the client.*

Output of client $\mathfrak{C}$*:* {}
Output of server $\mathfrak{S}$*: updated outsourced database*

The omission of the case distinction in Step 7.5 potentially generates unnecessary index entries with empty encrypted lists. This leaves potential for optimisation. While maintaining the security property, an empty index entry can be deleted, if after the execution of the UPDATE query there are no buckets that fulfil its conditions under icp. For the sake of understandability of the security proof, we omitted this case distinction. Please note that in Step 7, all indices for tuples in the buckets sent to the client are re-encrypted.

With these two protocols, we can define our modified database outsourcing scheme.

Definition 72 (l-IND-ICP MimoSecco Database Outsourcing Scheme)**.** *Let* $\mathsf{DB}_n \subset \mathsf{DB}$ *be all databases of size* n*, let* s *be the maximal length of an encrypted index entry for any* $d \in \mathsf{DB}_n$*, let* f *be a partitioning scheme with partitions of size* l*, and let* $(\mathsf{Gen}_I, \mathsf{Enc}_I, \mathsf{Dec}_I)$ *be an encryption scheme with* IND*-CPA security that pads index entries to length* s*. We define the* l-IND-ICP MimoSecco database outsourcing scheme *as* $(\mathsf{Gen}_{DB}, \mathsf{Enc}_{DB}, \mathsf{Q}_{DB})$ *with:*

- $\mathsf{Gen}_{DB} = \mathsf{Gen}_I$
- Enc_{DB} *as defined in Definition 69 constrained to* DB_n
- $\mathsf{Q}_{DB} = \pi^{l\text{-IND-ICP}}_{q_{select}} \cup \pi^{l\text{-IND-ICP}}_{q_{update}}$ *as defined in Protocols 5 and 6.*

Theorem 11. *The l-IND-ICP MimoSecco database outsourcing scheme has IND-ICP security and no query advantage over l-IND-ICP.*

Proof. Since we did not change the encryption mechanism $\mathsf{Enc}_{\mathrm{DB}}$ the proof for Theorem 10 (The MimoSecco database outsourcing scheme has IND-ICP security.) still holds. For the property no query advantage over l-IND-ICP, we show that the transcript of the execution of a query q on $\mathsf{Enc}_{\mathrm{DB}}(d, K)$ is indistinguishable from the transcript of the execution of query q on $\mathsf{Enc}_{\mathrm{DB}}(\mathfrak{p}(d), K)$ for any icp $\mathfrak{p}$ that respects the partitioning scheme f. This means, $\mathfrak{p}$ only permutes attribute values within the partitions $f(d)$ of d. If the two transcripts are indistinguishable, we can use any successful adversary on l-IND-ICP to build a successful adversary for the underlying encryption scheme $(\mathsf{Gen}_{\mathrm{I}}, \mathsf{Enc}_{\mathrm{I}}, \mathsf{Dec}_{\mathrm{I}})$.

Since the scheme already provides IND-ICP, we only need to consider steps with messages from the client to the server. In the SELECT Protocol, steps that contain messages from the client to the server are Steps 1 and 4. The message in Step 1 only depends on the query. In Step 4, the client sends $pids_c$, the ids of buckets that fulfil the conditions of the query under icp, to the server. For a query with conditions c, these bucket ids $pids_c$ are identical for $\mathsf{Enc}_{\mathrm{DB}}(d, K)$ and $\mathsf{Enc}_{\mathrm{DB}}(\mathfrak{p}(d), K)$ for any icp $\mathfrak{p}$ that respects f (cf. Definition 71): The icp $\mathfrak{p}$ only permutes attribute values within buckets. Therefore, if a bucket of d contains certain attribute values (for example those that match the condition), the corresponding bucket of $\mathfrak{p}(d)$ contains these attribute values as well.

Now for the modified UPDATE Protocol: Steps with messages from the client to the server are Steps 1, 4, 7.1, 7.5, 8, 10a.1, 10b.1, and 12. We already covered Steps 1 and 4 above. The messages in Step 7.1 and in Step 8 only depend on the query. The messages in Steps 10a.1 , and 10b.1 only depend on the index tables, which can not be used to distinguish $\mathsf{Enc}_{\mathrm{DB}}(d, K)$ from $\mathsf{Enc}_{\mathrm{DB}}(\mathfrak{p}(d), K)$ for any p, since our internal encryption scheme pads index lists to a fixed length. This leaves Step 7.5. which also can not be used to distinguish $\mathsf{Enc}_{\mathrm{DB}}(d, K)$ from $\mathsf{Enc}_{\mathrm{DB}}(\mathfrak{p}(d), K)$ for any p, for the same reason. This concludes our proof. □

Note, that we did only need the padding of index entries to length s for the proof of the modified UPDATE protocol. If we only want to retrieve information from a database, we do not need an encryption scheme with this property.

6. Implementations and Benchmarks

Estimations of theoretical scaling properties play an important role in the design of an algorithm or a scheme and in the selection of alternatives. For implementations, the theoretical overhead in the O notation is only one performance influencing factor that has to be considered. For practical applications, also factors that are irrelevant in the O notation can play an important role. Such factors for, are example, constant overhead factors or performance overheads introduced by concrete implementation choices of data structures or programming languages. Optimisations of implementations, while irrelevant from a theoretical point of view, now, affect the performance of the application.

In this chapter, we present two implementations of the MimoSecco database outsourcing scheme: the *MimoSecco* implementation and the *Cumulus4j* implementation.

The *MimoSecco implementation* is a research prototype written in Java that provides an SQL interface and uses an SQL backend. It has been presented at CeBIT, the biggest fair for information technology, as well as on the yearly reception of the KIT. In the context

of the MimoSecco research project, it has been integrated into a prototype of a database abstraction and synchronisation layer for CRM applications.

The *Cumulus4j implementation* is a Java plug-in for the database abstraction layer DataNucleus [Dat] and provides Java Persistence API (JPA), Java Data Objects (JDO), and SQL interfaces. DataNucleus can use numerous backends including most SQL databases (e. g. PostgreSQL, MySQL, SQLite, Oracle), map based databases (e. g. Cassandra) and web based backends (e. g. Amazon S3, Google Storage, JSON). The Cumulus4j implementation is free software under the AGPL. In the context of the Cumulus4j research project, it has been integrated in a commercial invoice processing software.

The overhead of our database outsourcing scheme depends on the queries as well as on the data stored. We provide benchmark results for both implementations for different scenarios, queries, and data. In Section 7, we present and discuss different optimisations of the index structures of our scheme and provide benchmark results for these optimisations.

Please note, that for our implementations, we used AES for the internal encryption scheme. Therefore, strictly speaking, the implementations do not provide IND-CPA security, since AES in CBC mode does not provably provide IND-CPA security. This is a potential source for side channels. It is, however believed, that the AES is a pseudorandom function. If this holds true, AES in CBC mode provides IND-CPA security. Additionally, the internal encryption scheme can be exchanged for one that provides IND-CPA security.

6.1. The MimoSecco Implementation

The MimoSecco implementation is a research prototype of a database adaptor that provides IND-ICP security [HH14]. It has been used in the MimoSecco Project [AGH11]. In this context, it has been integrated into a prototype of a database abstraction and synchronisation layer for CRM applications of the CAS Software AG [CAS]. The MimoSecco implementation has been presented in 2011, in 2012, and in 2014 at CeBIT, the biggest fair for information technology, as well as on the yearly reception of the KIT in 2012 (cf. Figure 7.14).

6.1.1. Scheme and Implementation Details

The MimoSecco implementation is written in Java. It is realised as an adaptor that can be deployed between the application and the database. It provides an SQL interface and uses a Java Database Connectivity (JDBC) driver for the back end database connection. For our benchmarks, we used a PostgreSQL database for the back end database. Additionally, the MimoSecco adaptor uses a local PostgreSQL database internally. The internal database is used for query processing and only stores data during the execution of a query. The usage of an internal database as a part of the query engine minimised implementation time and effort. Moreover, since the result set returned by the adaptor is generated in the internal database, SQL compliant result sets are generated automatically.

The scheme implemented in the MimoSecco implementation deviates from the scheme described in Section 4: Instead of generating a separate index table for each attribute of the original database, the MimoSecco adaptor manages index tables for different data types of attributes. For example all indices for Integer attributes are stored in a single index table while the indices for String attributes are stored in a different index table. Additionally, the MimoSecco adaptor stores meta information that allows to map index entries to attributes and to tables. Figure 7.15 shows the back end database scheme with

7.14.1: The MimoSecco prototype at CeBIT

7.14.2: The MimoSecco prototype at the yearly reception of the KIT

Figure 7.14.: The MimoSecco prototype at CeBIT and at the yearly reception of the KIT: Tables are stored on the PCs in the saves/castles and can be accessed by a separate client. In order to demonstrate the security properties of the MimoSecco scheme, a laptop shows parts of the encrypted contents of the outsourced database.

the index tables for String and Integer attributes. The table *TableMeta* holds names of tables stored in the back end. The name as well as the type of each column or attribute is stored in the table *FieldMeta*. Since a table has at least one column, there is at least one entry in *FieldMeta* for each entry in *TableMeta*. As mentioned above, the index entries of an attribute are stored in the index table for the data type of the attribute. For example, if the data type of the attribute is Integer, its index entries are stored in the table *IntIndex*. For the sake of clarity, Figure 7.15 only shows two different types of index tables (String and Integer). Index entries and data are stored according to the MimoSecco scheme: The index tables hold attribute values and an encrypted list of ids (keys of the table *Data*), the value occurs in. The table *Data* holds encrypted rows of the original databases. While preserving its security property, this extension to the original scheme has the following benefits:

- The scheme allows for the storage of multiple databases.
- The back end database scheme is independent from the original databases stored. There is no need to change the scheme in order to store a new or an additional database.
- Different data types have different orders. Storing them in separate index tables, each with the correct data type, ensures correct results for example for range queries and for comparisons executed in the back end DBMS.

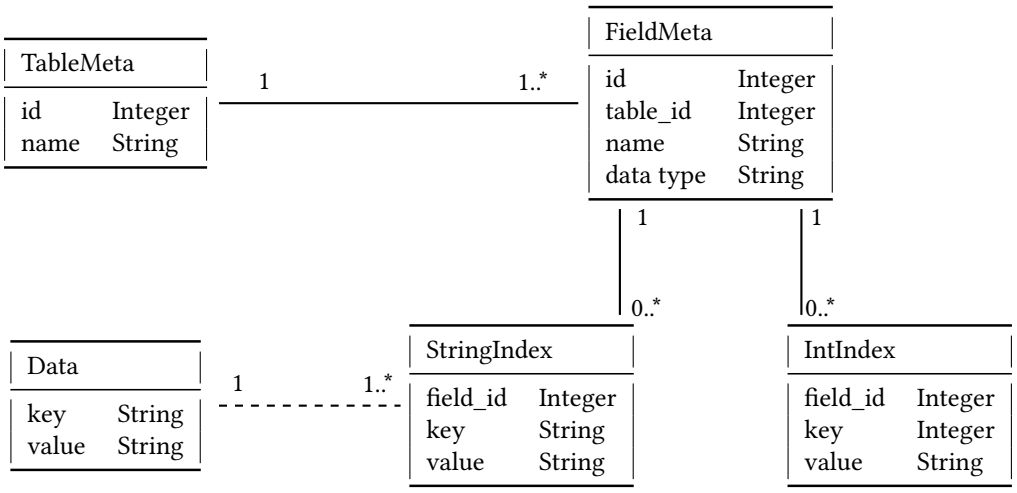


Figure 7.15.: The back end data structure of the MimoSecco and the Cumulus4j implementations. The attribute *value* of the *Index* tables holds encrypted ids of the *Data* table. This is indicated by the dashed line. The attribute *value* of the table *Data* holds encrypted rows of the original databases. Due to the meta tables, this scheme allows for IND-ICP secure storage of multiple databases.

6.1.2. Benchmarks

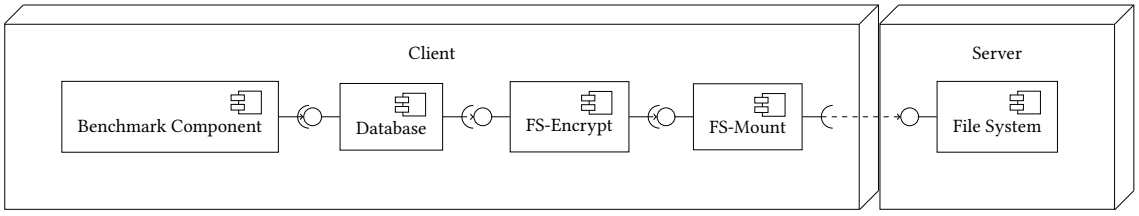
We measured the time from issuing a query until retrieving the result in three different scenarios. For each scenario, we used the same hardware, queries, and, as far as possible, software. In the following, we will describe the benchmarks of the MimoSecco implementation, the scenarios, the hard- and software, the queries, and the data in detail. In the last paragraph of this section, we present and discuss the benchmark results.

Benchmark Scenarios In order to provide meaningful benchmarks, we deployed the database back end on a different machine than the MimoSecco implementation. Figure 7.16 depicts the deployment used for the benchmarks. We compared the MimoSecco implementation to two different scenarios. All three scenarios involve a client and a server. The time needed to process a query is measured on the client, while the database sever is used for storage.

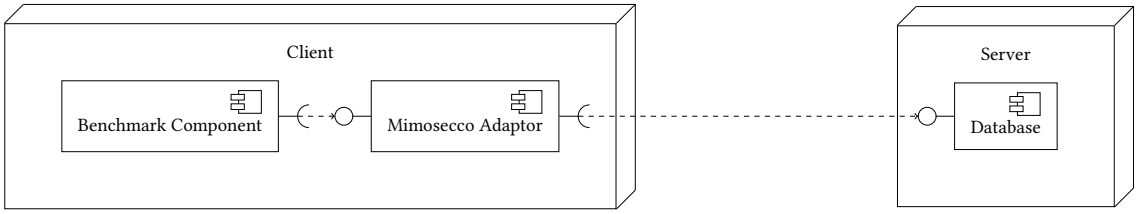
The first scenario (JDBC, Figure 7.16.1) is without the usage of any encryption. The original database is stored on the server in plaintext. Our benchmark component connects via a Java JDBC driver directly to the database component on the server. Since in this scenario, queries are processed directly by the database component with no additional software as well as no encryption involved, it should provide the best benchmark results.

For the second scenario (SSHFS, Figure 7.16.2), we deployed the database component on the client. The idea of this scenario is to use the server as a file server for the database component. Therefore, we mounted parts of the file system of the server on the client. Additionally, we encrypted the file system on the side of the client. As a result, the database component writes its data into an encrypted file system stored on the server. We realised the encryption of the outsourced file system by a chain of Filesystem in Userspace (fuse) drivers. For mounting the file system on the server, we used the Secure SHell FileSystem (SSHFS). For file system encryption, we used encfs, an encrypted file system extension for fuse. Assuming encfs provides an IND-CPA-secure encryption,

7.16.1: JDBC



7.16.2: SSHFS



7.16.3: MimoSecco

Figure 7.16.: Benchmark scenarios for the benchmarks of the MimoSecco implementation

compared to the other two scenarios, this scenario has the strongest security properties. However, this scenario has the drawback that the database component has to use the potentially slow network connection for each file system access. The extend in which parts of queries can be outsourced to the server depends on the file system on the server, the fuse chain and the usage of the file system by the database component. Therefore, this solution might scale differently than the JDBC scenario or the MimoSecco scenario. Additionally, the client has to support a fully fledged database server component.

The third scenario is the MimoSecco scenario (Figure 7.16.3). Therefore, the MimoSecco adapter, which connects via JDBC to the database component on the server, is deployed on the client. The MimoSecco adapter is deployed between the benchmark component and the back end database.

Hard- and Software The database component used in each scenario is a PostgreSQL (version 9.1) database. The hardware specifications of the server are as follows:

- Processor: 2 AMD Opteron, 6 Cores @ 2.4GHz
- RAM: 32 GB

The hardware specifications of the client are a follows:

- Processor: Intel Pentium Dualcore @ 2GHz
- RAM: 2 GB

The client and the server are connected with 100Mbit/s Ethernet.

Queries and Data The queries and data used in the benchmark are from the *AS/sup 3/AP* benchmark [TOB89]. Since the current MimoSecco implementation does not fully support SQL, we chose a subset of the queries of the *AS/sup 3/AP* benchmark. We used SELECT, INSERT, UPDATE, DELETE, ALTER TABLE, and DROP TABLE queries. The times measured in the next paragraph are averages over different queries and multiple runs.

For the benchmarks, we used two different data sets of the *AS/sup 3/AP* benchmark, each with 10,000 tuples. In the data set *uniques*, all attributes have unique values. In the data set *hundreds*, most of the attributes have exactly 100 different values. Furthermore, the attribute values are correlated. The *hundreds* data set has a selectiveness of 100. A more detailed explanation of the data sets and the queries used can be found in the appendix in Chapter A.1.

Results Figure 7.17 provides an overview over the benchmark results. Here, the results from the benchmarks for the two data sets *hundreds* and *uniques* are averaged. Detailed benchmark results can be found in the appendix in Section A.1 in Figures A.4, A.5, and A.6. This benchmark does not show the scaling properties of the three scenarios but rather allows for a comparison of the overhead of different query types. Scaling benchmarks of the MimoSecco implementation are presented in Section 7. The benchmark results

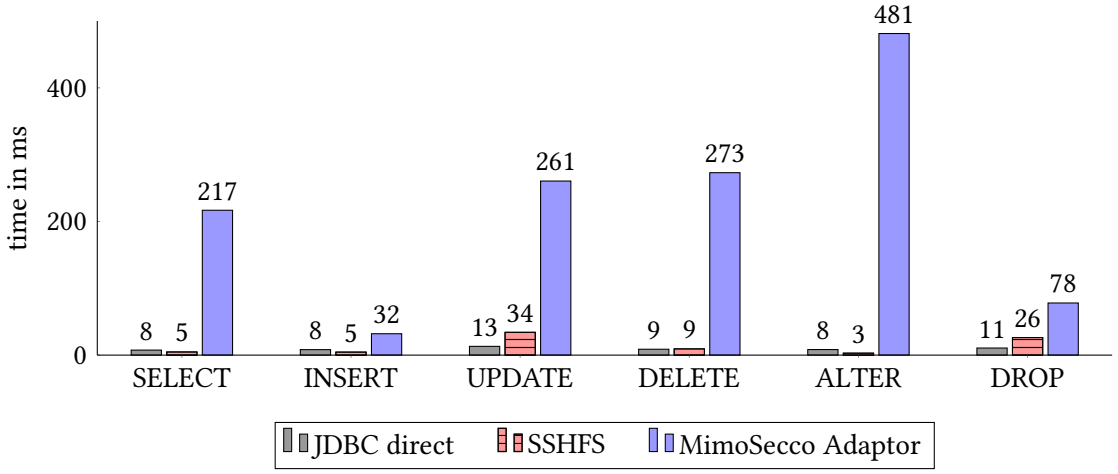


Figure 7.17.: An overview over the benchmarks of the MimoSecco implementation

show that the JDBC and the SSHFS scenarios do have about the same overhead. Only for UPDATE and DROP queries, the SSHFS solution is slower than direct database access. Furthermore, the results show a significant overhead of the MimoSecco implementation compared to direct database access. This overhead can be explained by the additional queries introduced by the MimoSecco adaptor, the overhead for de- and encryption, and the overhead introduced by the query parser. In the MimoSecco adaptor, query parsing, query generation, and de- and encryption are realised using String operations in Java, which can be very expensive from a performance point of view.

Additionally, the focus of the MimoSecco research prototype is to demonstrate realisability. However, some performance optimisations were introduced in a later stage. Figure 7.18 compares benchmark results of an early variant of the MimoSecco implementation with results of a recent variant that, for example, uses batch statements. This shows

the optimisation potential of the MimoSecco research prototype. A more efficient String processing in the query parser and in the internal database engine or parallelisation of queries potentially lead to additional performance enhancements.

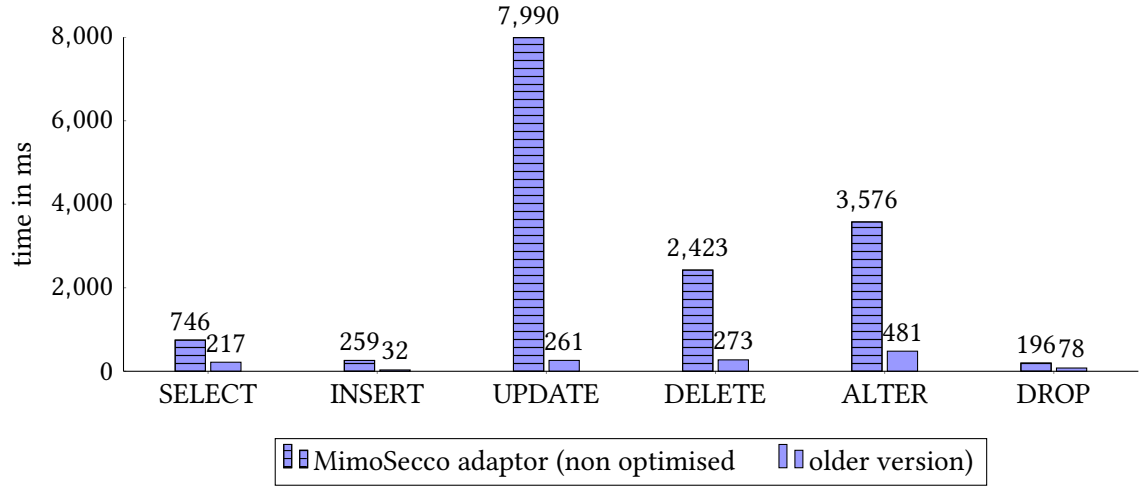


Figure 7.18.: During this work, the MimoSecco research prototype has been improved and extended. This chart shows a comparison of the benchmark results of one of the first and one of the most recent version of the MimoSecco prototype showing the optimisation potential. The recent version uses batch statements and has improved query handling.

The MimoSecco benchmark results in Figure 7.17 show that the overheads for SELECT, UPDATE and DELETE queries are about the same. The overhead for INSERT queries is significantly lower. This can be explained by the fact that all inserted values are unique. Therefore, the adaptor created new index entries. Inserting values into existing index entries, which involves a SELECT query, a decryption, String operations, an encryption, and an UPADATE query is not necessary. This also explains the lower overhead of the DROP queries: They also no not involve retrieval end decryption of index entries, which, in turn, involves potentially expensive String operations.

6.2. The Cumulus4j Implementation

The Cumulus4j implementation is a plug-in for the database abstraction layer DataNucleus. It has been developed and used in the Cumulus4j project [Hub+13], where it has also been integrated in Ax Easy [AX], a commercial invoice processing software as a service. The full source code of Cumulus4j as well as the benchmarks are available online [Nig] under the AGPL.

6.2.1. Scheme and Implementation Details

Cumulus4j integrates into DataNucleus as an OSGi plug-in. Figure 7.19 depicts the coarse architecture of a database application that uses DataNucleus with Cumulus4j. The application component uses DataNucleus and DataNucleus accesses the underlying database. Cumulus4j enhances DataNucleus. The integration of our plug-in, and therefore the encryption of the underlying database, is, except for a minimum of Cumulus4j API calls controlling the key management, fully transparent to the application. Thus, existing

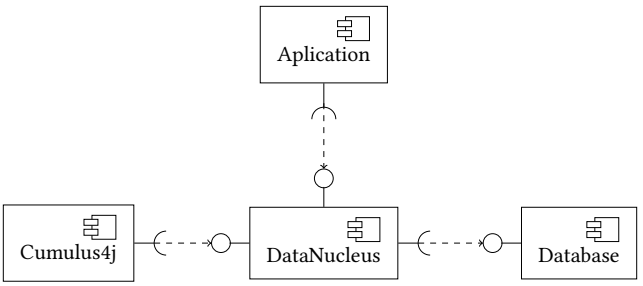


Figure 7.19.: Coarse architecture of an database application with DataNucleus and Cumulus4j. The integration of Cumulus4j as a plug-in for DataNucleus is transparent for the application.

applications using DataNucleus can be easily enhanced with Cumulus4j. In contrast to MimoSecco, the encryption keys have to be unlocked for the Cumulus4j plug-in, for example by the application itself. During database operations, the encryption keys are in the memory of the server Cumulus4j is deployed to. Note, that the back end databases can and should be deployed to different servers. The Cumulus4j implementation uses a back end data structure similar to the back end data structure of the MimoSecco implementation (cf. Figure 7.15).

6.2.2. Benchmarks

We measured the performance of Cumulus4j with PolePosition [Polb], a generic open source Java benchmark suite for object relational mappings. We measured two scenarios: DataNucleus without Cumulus4j and DataNucleus with Cumulus4j. In both cases, in order to minimise the impact of the effects of the OS and the Java garbage collection, the benchmark ran multiple times and the results were averaged. For our benchmarks, we used two machines: a client application and benchmark server and a database server. The application and benchmark server hosted DataNucleus and Cumulus4j and ran PolePosition. The back end database (MySQL 5.5) was deployed on the database server.

Queries and Data The PolePosition benchmark suite is organised in so-called *circuits*. These circuits are different benchmark scenarios. They differ in the object data structures and the queries. The idea of these circuits is to replicate different complex application scenarios. A detailed description of the different circuits can be found in the appendix in Chapter A.2 and on the PolePosition website [Pola].

Hardware As mentioned above, we used two servers: an application and benchmark server and a database server. The application and benchmark server has the following hardware:

- Processor: Intel Core i7-2700K
- RAM: 16 GB

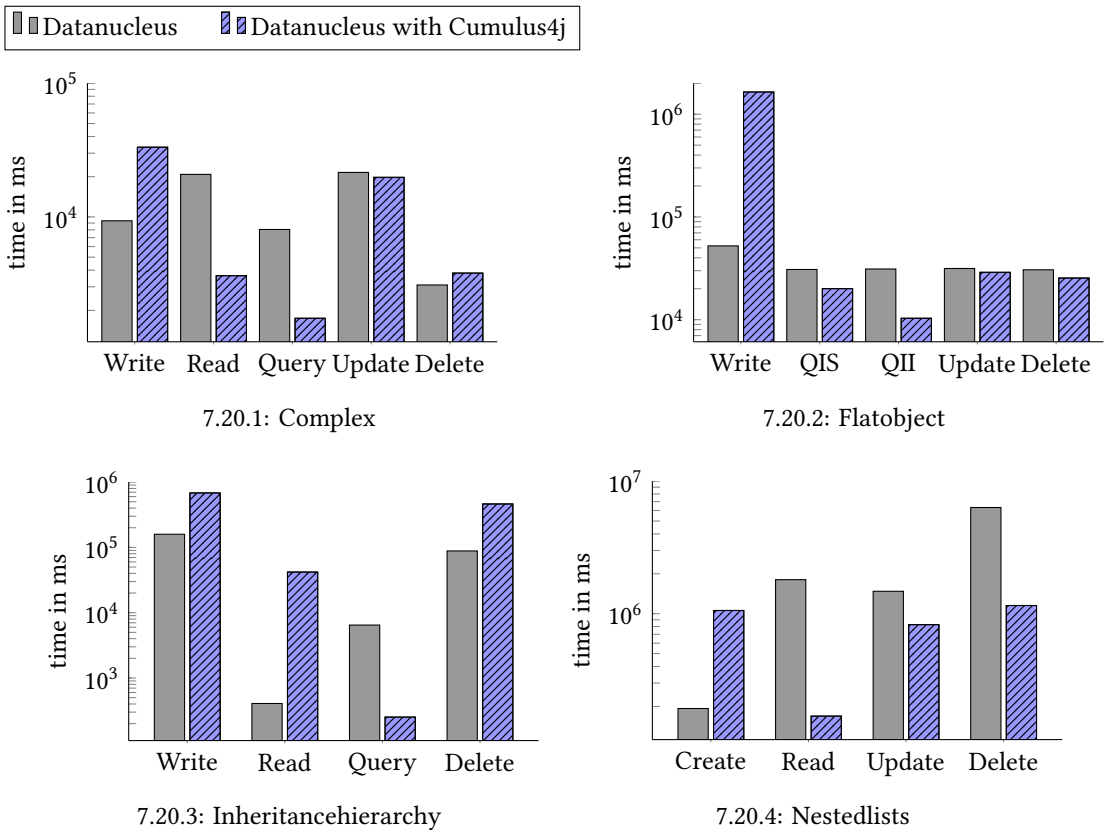
The hardware of the database server is as follows:

- Processor: AMD Phenom II X6

- RAM: 8 GB RAM

The servers are connected via Gigabit Ethernet.

Results An overview over the benchmark results is shown in Figure 7.20. Detailed benchmark results can be found in the Appendix in Chapter A.2.2. In order to cope with



7.20.1: Complex

7.20.2: Flatobject

7.20.3: Inheritancehierarchy

7.20.4: Nestedlists

Figure 7.20.: Benchmark results in ms of the PolePosition benchmark circuits *Complex*, *Flatobject*, *Inheritancehierarchy*, and *Nestedlists* for Datanucleus with and without Cumulus4j. In order to cope with outliers, the time axis is in logarithmic scale. QIS are queries of indexed Strings and QII are queries of indexed Integers.

outliers, the mean time for all results is shown on a logarithmic scale.

The initial creation of the data sets and insertion of new data sets (WRITE and CREATE) is more complex due to the additional overhead introduced by the Cumulus4j plug-in. Data encryption and the creation of indices result in an overhead of roughly factor 5.

Since the implicit use of indices in Cumulus4j and the fact that MySQL does not use indices per default, reading data (QUERY and READ) with the Cumulus4j plug-in is faster than without it, except for the Inheritancehierarchy circuit. Enabling indexing techniques of MySQL also speeds up queries of Cumulus4j, but may reduce the benefit of the Cumulus4j indices and, therefore, the speed-up relative to queries without Cumulus4j. The higher overhead of the Inheritancehierarchy queries can be explained with the data structures used in this circuit. Objects in this circuit have a class hierarchy with a depth of five levels. This results in subsequent queries to the index structures and additional

decryption operations between these queries. Updates take almost the same time while deletes are marginally slower with Cumulus4j. The benchmark results show, that the overhead introduced by Cumulus4j highly depends on the data structures and the queries used. According to the benchmark results, Cumulus4j is best suited for applications with many read and few write queries. Please note that when using Cumulus4j in web application, end users would probably not notice any slowdown at all, because the overall response time is only marginally affected by Cumulus4j.

7. Optimisations of Index Structures

The index tables of the MimoSecco database outsourcing scheme presented in Section 4 enable sublinear search of individual index values with respect to the size of the database and, therefore, allow for efficient execution of queries. There are, however, scenarios where the overall database outsourcing scheme does not scale sublinearly. In an index table, the rows ids of index values are stored in lists. In general, the size of these lists linearly scales with the size of the database. Therefore, checking if an id is in a list can involve a linear (in the size of the database) overhead.

Consider for example the database depicted in Figure 7.21. The index tables after

row	name	gender	salary
1	Johnson	male	10
2	Johnson	female	10
3	Johnson	male	10
4	Johnson	-	10
5	Johnson	male	9
6	Johnson	male	9
7	Jones	male	8
8	Smith	male	9
9	Jones	male	7

row	name	gender	salary
10	Jones	male	7
11	Jones	male	7
12	Jones	male	7
13	Williams	female	8
14	Williams	male	8
15	Jones	male	7
16	Johnson	female	8
17	Johnson	female	10
18	Williams	male	8

Figure 7.21.: An example employee database *empl*. We use this database to illustrate examples of the index optimisations. The attribute row is added for easier traceability. We use this example throughout this section in order to illustrate the optimisations.

outsourcing are depicted in Figure 7.22. Now consider for an example that we want to check if there are any male employees with the name Smith. The query for this example is as follows:

$$\text{SELECT } * \text{ FROM } \mathit{empl} \text{ WHERE } \mathit{gender} = \mathit{male} \text{ AND } \mathit{name} = \mathit{Smith}$$

Executing this query on the outsourced database results in the following queries to the index tables:

$$\text{SELECT } \mathit{rows} \text{ FROM } \mathit{empl}_{\mathit{index},\mathit{gender}} \text{ WHERE } \mathit{gender} = \mathit{male}$$
$$\text{SELECT } \mathit{rows} \text{ FROM } \mathit{empl}_{\mathit{index},\mathit{name}} \text{ WHERE } \mathit{name} = \mathit{Smith}$$

values	rows
Johnson	$\mathsf{Enc}_\mathrm{I}((4, 3, 17, 5, 1, 2, 6, 16), K)$
Jones	$\mathsf{Enc}_\mathrm{I}((10, 12, 11, 15, 7, 9), K)$
Smith	$\mathsf{Enc}_\mathrm{I}(8, K)$
Williams	$\mathsf{Enc}_\mathrm{I}((18, 13, 14), K)$

7.22.1: $empl_{index,name}$

values	rows
10	$\mathsf{Enc}_\mathrm{I}((3, 1, 17, 4, 2), K)$
9	$\mathsf{Enc}_\mathrm{I}((8, 6, 5), K)$
8	$\mathsf{Enc}_\mathrm{I}((14, 16, 7, 18, 13), K)$
7	$\mathsf{Enc}_\mathrm{I}((12, 15, 9, 10, 11), K)$

7.22.2: $empl_{index,salary}$

values	rows
male	$\mathsf{Enc}_\mathrm{I}((15, 18, 6, 7, 8, 9, 10, 11, 1, 3, 5, 12, 14), K)$
female	$\mathsf{Enc}_\mathrm{I}((2, 17, 13, 16), K)$
-	$\mathsf{Enc}_\mathrm{I}(4, K)$

7.22.3: $empl_{index,gender}$

Figure 7.22.: The index tables for our example database in Figure7.21 for the attributes *name*, *salary* and *gender*. The lists of row ids are encrypted with an internal encryption mechanism Enc_I and an encryption key K.

The result of the first query is $\mathsf{Enc}_\mathrm{I}((1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 18), K)$ and the result of the second query is $\mathsf{Enc}_\mathrm{I}(8, K)$. Now, the client has to decrypt both results and execute a set intersection on the decrypted sets of row ids. Since the first set is comparatively large and the second set only contains one element (the id 8), the client had to fetch a large number of false positives from the database, which induces a large overhead.

Therefore, in the following section, we propose different optimisations for the index tables in order to reduce this overhead. We will see that the individual optimisations are suited for different scenarios that depend on the data in the database and the queries. Note that these optimisations can be used independently of each other. For each attribute a different optimisation can be applied. Furthermore, some of these optimisations can be combined.

The first class of optimisations are methods for compressing the lists of row ids in order to directly reduce the transportation and decryption overhead. After presenting these compression optimisations, we will examine different data structures for the index tables and their impact of the efficiency of query execution.

7.1. Compression of Index Lists

In this section, we present different ways of compressing the row lists in the index entries. The basic idea is to decrease the size of the lists in order to decrease the transportation and the decryption overhead. The three compression methods discussed in this section are *Intervals*, *Exclusive Labels*, and *Normalisation*. Since in an index list every entry is unique, there are no recurring patterns and, therefore, classic compression algorithms such as LZW are not considered here.

7.1.1. Intervals

The lists of index entries can be compressed by replacing sequences rows ids with a representation of the interval. The table in Figure 7.23 shows the index table for

values	rows
Johnson	$\mathsf{Enc}_\mathrm{I}((1-6, 16, 17), K)$
Jones	$\mathsf{Enc}_\mathrm{I}((7, 9-12, 15), K)$
Smith	$\mathsf{Enc}_\mathrm{I}(8, K)$
Williams	$\mathsf{Enc}_\mathrm{I}((13, 14, 18), K)$

Figure 7.23.: Interval compression: More than two subsequent ids of rows are replaced with a representation of their interval.

the attribute *name* for our example database *empl* with an interval compression. This compression is well suited for attributes with few different attribute values and long sequences. The drawback of this optimisation is the overhead for sorting the lists of row ids.

7.1.2. Exclusive Labels

Instead of storing the ids of rows where an attribute value occurs, you can store the id of rows where an attribute value does not occur. This is suited for attributes with values that occur in more than half of the rows of the database. Consider for example the index

values	rows
male	$\mathsf{Enc}_\mathrm{I}((\neg 2, 4, 13, 16, 17), K)$
female	$\mathsf{Enc}_\mathrm{I}((2, 13, 16, 17), K)$
-	$\mathsf{Enc}_\mathrm{I}(4, K)$

Figure 7.24.: Exclusive Labels compression: The attribute value *male* occurs in more than half of the rows in the database ($n = 15$). Therefore, the row id list of this attribute value is replaced by the symbol ¬ followed by the list of ids of the rows, the attribute value does not occur.

table in Figure 7.24. Here, the attribute value *male* occurs in 13 of 18 rows. With the use of the exclusive label, we have only to store 5 row ids in its index list if we store the ids of the rows it does not occur. Here, the symbol ¬ marks the use of exclusive labels.

The drawback of this compression is the overhead introduced to INSERT and DELETE queries. If a query, for example, inserts a row with *gender* = *female*, we have also to change the index entry for all attribute values to of the attribute *gender* that are different to *female* and use exclusive labels.

7.1.3. Normalisation

If two attributes are correlated, certain values of these attributes often occur together. Then, there are multiple row ids that occur in both index entries of these attribute values.

We can store such correlations normalised in an separate table. We call this table the normalisation table. This normalisation table can save space and communication overhead. For an example, consider Figure 7.25. The tables $empl_{index_n,name}$ and $empl_{index_n,salary}$ show

values	rows
Johnson	$\mathrm{Enc_I}((*1, *2, 16), K)$
Jones	$\mathrm{Enc_I}((7, *3, 15), K)$
Smith	$\mathrm{Enc_I}(8, K)$
Williams	$\mathrm{Enc_I}(*4, K)$

7.25.1: $empl_{index_n,name}$

values	rows
10	$\mathrm{Enc_I}(*1, K)$
9	$\mathrm{Enc_I}((*2, 8), K)$
8	$\mathrm{Enc_I}((7, *4, 16), K)$
7	$\mathrm{Enc_I}((*3, 15), K)$

7.25.2: $empl_{index_n,salary}$

reference	rows
1	$\mathrm{Enc_I}((1, 2, 3, 4, 17), K)$
2	$\mathrm{Enc_I}((5, 6), K)$
3	$\mathrm{Enc_I}((9, 10, 11, 12), K)$
4	$\mathrm{Enc_I}((13, 14, 18), K)$

7.25.3: $empl_{n(name,salary)}$

Figure 7.25.: Normalisation optimisation: The row ids of values of different attributes that occur together twice ore more are stored together in a separate table.

the normalised index tables for the attributes *name* and *salary*. The symbol $*$ marks the use of a normalisation while the number afterwards represents the number of the normalisation. The table $empl_{n(name,salary)}$ holds the normalised correlations.

The downside of this optimisation is the additional overhead for normalisation management. Consider for example the following query:

$$\text{UPDATE } empl \text{ SET } salary = 10 \text{ WHERE } row = 5 \tag{7.2}$$

This query changes the row with (5, Johnson, male, 9) in the original database to (5, Johnson, male, 10). The execution of this query induces the following changes to the index tables:

- table $empl_{n(name,salary)}$, row 1:
  update value of attribute *rows* to $\mathrm{Enc_I}((1, 2, 3, 4, 17, 5), K)$
- table $empl'_{n(name,salary)}$:
  deletion of row 2
- table $empl_{index_n,salary}$, row 2:
  update value of attribute *rows* to $\mathrm{Enc_I}((6, 8), K)$
- table $empl_{index_n,name}$, row 1:
  update value of attribute *rows* to $\mathrm{Enc_I}((*1, 6, 16), K)$

Figure 7.26 shows the resulting tables after execution of our example query.

There are, however, queries that can be executed very efficiently with the normalisation optimisation. Replacing one normalised relation with another one can be done without changing any row ids, but only with changing pointers to the normalisation table.

values	rows
Johnson	$\mathsf{Enc}_{\mathrm{I}}((*1, 6, 16), K)$
Jones	$\mathsf{Enc}_{\mathrm{I}}((7, *3, 15), K)$
Smith	$\mathsf{Enc}_{\mathrm{I}}(8, K)$
Williams	$\mathsf{Enc}_{\mathrm{I}}(*4, K)$

7.26.1: $empl'_{index_n, name}$

values	rows
10	$\mathsf{Enc}_{\mathrm{I}}(*1, K)$
9	$\mathsf{Enc}_{\mathrm{I}}((6, 8), K)$
8	$\mathsf{Enc}_{\mathrm{I}}((7, *4, 16), K)$
7	$\mathsf{Enc}_{\mathrm{I}}((*3, 15), K)$

7.26.2: $empl'_{index_n, salary}$

refernece	rows
1	$\mathsf{Enc}_{\mathrm{I}}((1, 2, 3, 4, 17, 5), K)$
3	$\mathsf{Enc}_{\mathrm{I}}((9, 10, 11, 12), K)$
4	$\mathsf{Enc}_{\mathrm{I}}((13, 14, 18), K)$

7.26.3: $empl'_{n(name, salary)}$

Figure 7.26.: The normalised index tables after execution of Query 7.2. The execution of this query changes four entries in the index tables.

7.2. Sorted Index Lists - Binary Search

The implementationsMimoSecco and Cumulus4j use a block cipher (e. g. AES with a block length of 256 Bit) in CBC mode for en- and decryption of the row id lists. This mode allows to decrypt individual cipher blocks without the need to decrypt all previous cipher blocks.

If a list of row ids has to be split into several blocks in order to encrypt it, sorting the list prior to encryption can decrease the decryption overhead. For example, in order to check if a certain row id is in a list, instead of decrypting the complete list (ie. decrypting all cipher blocks), binary search on cipher blocks can be used. Figure 7.27 depicts the

values	rows
male	$\mathsf{Enc}_{\mathrm{I}}((1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 18), K)$
female	$\mathsf{Enc}_{\mathrm{I}}((2, 13, 16, 17), K)$
-	$\mathsf{Enc}_{\mathrm{I}}(4, K)$

Figure 7.27.: The index tables of the attribute *gender* of our example with sorted lists of row ids.

index table for the attribute *gender* of our example database with sorted lists of row ids.

Furthermore, assume we want to check if the entry in row 11 is *male* and that a cipher block can hold two row ids. Figure 7.28 depicts a representation of the sorted row id list of the attribute value*male*.

Applying a binary search pattern, we first have to decrypt cipher block 3. The decryption of cipher block 3 yields the ids 6 and 7. Since $11 > 7$, we have to search in the higher part of the list. We decrypt cipher block 5 and get the ids 10 and 11 which concludes our binary search. Due to the CBC mode, in order to decrypt cipher blocks 3 and 5, we also need to decrypt cipher blocks 2 and 4.

In this example, we decrypted 4 cipher blocks. If the list was not sorted, we had to

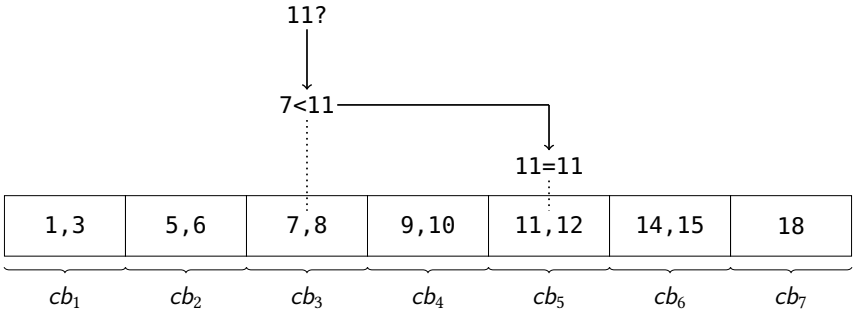


Figure 7.28.: A representation of the cipher blocks (cb_i) of the sorted row id list of the attribute value *male* assuming a cipher block can hold two ids.

decrypt 7 cipher blocks in the worst case. In the O calculus, the number of cipher blocks that need to be decrypted in order to check if a row id is in a list is in $O(\text{ld } n)$. Note, that in order optimise queries such as the query in the example above, we need to determine the number of row ids in a list. This can be done, for example, by storing this number in an additional column of the index tables. Additionally, these numbers can be encrypted.

7.3. Keyed Hash Index

In our scheme, the index lists are encrypted. This has the disadvantage that there are queries that introduce a large communication and decryption overhead. For example, the server can not calculate the result of a JOIN or the result of a WHERE clause containing more than one condition. A possible solution for this problem is the Keyed Hash Index. This is a similar approach as Anatomization [NC11].

Instead of encrypted lists of indices, the Keyed Hash Index introduces a key that points to an additional table, the Keyed Hash Index table. This additional table holds hashed pointers with the lists of row ids in plaintext. Consider Table 7.29 which depicts the

values	sequence
Johnson	1
Jones	2
Smith	3
Williams	4

7.29.1: $empl_{index_{kh},name}$

hsequence	rows
$H_\kappa(1)$	4,3,17,5,1,2,6,16
$H_\kappa(2)$	10,12,11,15,7,9
$H_\kappa(3)$	8
$H_\kappa(4)$	18,13,14

7.29.2: $empl_{index_{kh}}$

Figure 7.29.: The Keyed Hash Index. The index table holds pointers to the Keyed Hash Index table. The Keyed Hash Index table holds hashed pointers and the corresponding lists of row ids in plaintext.

Keyed Hash Index for the attribute *name* of our example database. The column *rows* of the index table is replaced with a column *sequence* holding sequence numbers. The additional Keyed Hash Index table (Table 7.29.2) holds hashes of sequence numbers and the corresponding row ids in plaintext. In order to retrieve the row ids for an attribute value, we first have to retrieve the sequence number for this attribute value, calculate the hash of this sequence number and then retrieve the corresponding list of the Keyed Hash

Index table. Consider for example the following query:

$$\text{SELECT } * \text{ FROM } \mathit{employees} \text{ WHERE } \mathit{name} = \mathit{Smith}$$

Executing this query on the outsourced example database results in the following two queries:

$$\begin{aligned} &\text{SELECT } \mathit{sequence} \text{ FROM } \mathit{empl}_{\mathit{index}_{kh},\mathit{name}} \text{ WHERE } \mathit{name} = \mathit{Smith} \\ &\text{SELECT } \mathit{rows} \quad \text{FROM } \mathit{empl}_{\mathit{index}_{kh}} \quad \text{WHERE } \mathit{hsequence} = H_\kappa(3) \end{aligned}$$

The client has to calculate the value of $H_\kappa(3)$ locally. Since there is no need to decrypt any indices and computing a hash, in general, can be done very efficiently, the Keyed Hash Index has great potential for reducing the overhead of database outsourcing schemes, even for simple queries. For example, joins and computing the result of two conditions in the WHERE clause combined by AND can be computed on the server. Therefore, there is no need to retrieve potentially long lists of row ids in order to execute such queries.

Using the Keyed Hash Index in our database outsourcing scheme, however, breaks the security notion IND-ICP. Consider for example the tables depicted in Figure 7.31 that show index tables for the two databases in Figure 7.30.2. From the first Keyed Hash Index,

row	name	gender
1	Jones	male
2	Johnson	female
3	Johnson	female

7.30.1: *empl*2

row	name	gender
1	Jones	female
2	Johnson	male
3	Johnson	female

7.30.2: $\mathfrak{p}(\mathit{empl}2)$

Figure 7.30.: An example employees database. Table 7.30.1 depicts the original database and Table 7.30.2 the database after application of an independent column permutation $\mathfrak{p}$ that permutes two attribute values of the attribute *gender*.

the adversary learns that there are two identical rows in the original database. Namely the rows 2 and 3. From the second Keyed Hash Index, the adversary learns that there are no identical rows in the database. Therefore, the adversary can distinguish the two original databases and break our security notion IND-ICP.

It is an open question to formalise the level of privacy a database outsourcing scheme with a Keyed Hash Index provides. Given the support for more efficient execution of queries, an analysis of the privacy guarantees of the Keyed Hash Index seems worthwhile.

7.4. Storing Index Lists as B-Trees

In order to support efficient execution of range queries, database management systems use B-Trees. B-Trees store a sorted list of attribute values that also can be accessed through a tree structure. B-Trees combine the advantages of Trees and Linked Lists. A starting value of a range query can be found in sublinear time, while the next value can be found in $O(1)$ since the leaves of the B-Tree are linked.

In order to support efficient query execution, Damiani et al. propose in [Dam+03] to use B-Trees for database outsourcing schemes as well. For security reasons, they propose to encrypt the leaves of the B-Tree.

values	sequence
Jones	1
Johnson	2

7.31.1: $empl2_{index_{kh},name}$

values	sequence
male	3
female	4

7.31.2: $empl2_{index_{kh},gender}$

hsequence	rows
$H_\kappa(1)$	1
$H_\kappa(2)$	2,3
$H_\kappa(3)$	1
$H_\kappa(4)$	2,3

7.31.3: $empl2_{index_{kh}}$

values	sequence
Jones	1
Johnson	2

7.31.4: $\mathfrak{p}(empl2)_{index_{kh},name}$

values	sequence
male	3
female	4

7.31.5: $\mathfrak{p}(empl2)_{index_{kh},gender}$

hsequence	rows
$H_\kappa(1)$	1
$H_\kappa(2)$	2,3
$H_\kappa(3)$	2
$H_\kappa(4)$	1,3

7.31.6: $\mathfrak{p}(empl2)_{index_{kh}}$

Figure 7.31.: The Keyed Hash indices of the tables in Figure 7.30.2. The adversary can distinguish these two indices since she learns from the Keyed Hash Index of $empl2$, there are two identical lines in the original database $empl2$, while she learns from the Keyed Hash Index of $\mathfrak{p}(empl2)$ that every line in $\mathfrak{p}(empl2)$ is different.

In the following, we adapt B-Trees for our indices. Here, the advantages of B-Trees are similar to the advantages of sorted lists. They support efficiently checking if a row in the original database has a certain attribute value. Additionally, B-Trees efficiently support INSERT queries if each child and leaf is stored in a different table cell. Therefore, the B-Tree optimisation can be seen as a extension of the sorted list optimisation proposed in Section 7.2. Figure 7.32 shows an example representation of the B-Tree for the attribute *name* and the attribute value *Johnson* of our example database *employees*. The interval and leaf sizes are small in order to show the tree structure. Implementations of this optimisation should have leaf sizes that are multiples of the length of a cipher block. Furthermore, in an implementation of this tree, the intervals can be replaced with keys and pointers. For example the node [1,6] can be stored as:

$$< \text{pointer to node } [1, 3] > 4 < \text{pointer to node } [4, 5] >$$

Storing pointers in separate cells allows for traversing a path in the tree on the server with a single query.

The leaves of the tree hold the row ids and are encrypted. In contrast to regular B-Trees, we need empty leaves and nodes that point to empty leaves. Otherwise, our secure database outsourcing scheme that uses the B-Tree optimisation does not fulfil IND-ICP. Consider for example two B-Trees for two attribute values of different attributes. If there is no leaf in both trees that potentially holds the same row id, these two attribute values can not be related in the original database.

For our example, this would be the case with the B-Trees for the attribute *name* and the value *Smith* and the attribute *gender* and the values *female* and -, the third value of the attribute *gender*. Assuming an interval size of 3, there would be no leaves in the B-Trees for *female* and - that potentially hold the row id 8 (cf. Figure 7.21). Therefore, the adversary learns that *Smith* in the original database is *male*.

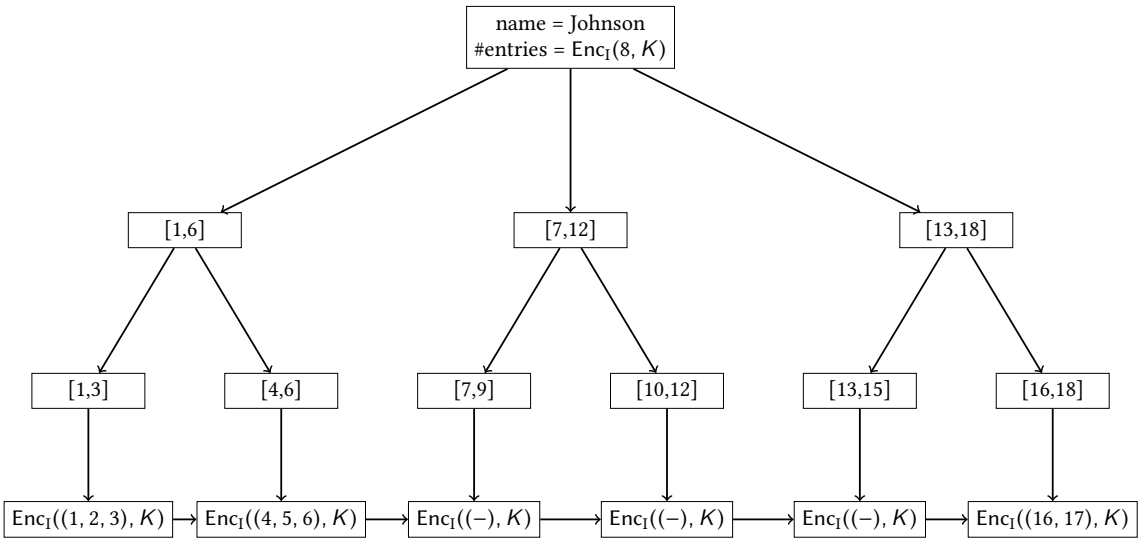


Figure 7.32.: An example B-Tree with $t = 3$ for the root node and an interval size of 3. This B-Tree stores the row ids for the attribute *name* and the value *Johnson* of our example database *employees*. Row ids are stored encrypted in the leaf nodes. The last inner nodes before the actual leaves are not needed and are only depicted to illustrate the interval sizes of the leaf nodes.

Storing empty leaves and nodes that point to empty leaves introduces a storage overhead, especially for attributes with a large range and sparse attribute values. Adaptively choosing different node intervals for each attribute value results in a lower storage overhead. Since the optimal parameters depend on the data stored in the tree, during runtime, the optimal intervals may change which also results in a reorganisation overhead for the whole tree.

Also, encrypting the nodes, in addition to the leaves, allows for B-Trees with no empty leaves or nodes that point to empty leaves, since the adversary does not learn anything about ranges of rows the attribute value occurs. This, however, introduces additional decryption overhead for each node. Additionally, as mentioned above, without encryption and depending on how links to other nodes are stored, traversing a path in the B-Tree can be implemented with a single query. If each node is encrypted, each node has to be retrieved an encrypted in order to determine the next node the be retrieved.

7.5. Storing Index Lists in Buckets

Instead of building a B-Tree manually, we also can roll out the tree into buckets and rely on the indexing by the database management system. The leaves of the B-Tree (cf. Section 7.4) partition the room of possible row ids. This also is the idea of the bucket optimisation. But instead of accessing a single partition with a tree structure, we store the interval for each partition. Consider for example Figure 7.33. The table in this figure depicts the Bucket Index for the attribute *name* and attribute value *Johnson*. Each row holds the start of the interval and the encrypted row ids of the interval. Similar to the B-Tree, in order to fulfil the security notion IND-ICP, we also have to store empty buckets. For accessing a bucket, we need the following formula:

$$b(r) = (r \operatorname{div} (i + 1)) \cdot i + 1$$

start	rows
1	$\mathsf{Enc}_{\mathrm{I}}((1,2,3),K)$
4	$\mathsf{Enc}_{\mathrm{I}}((4,5,6),K)$
7	$\mathsf{Enc}_{\mathrm{I}}(-,K)$
10	$\mathsf{Enc}_{\mathrm{I}}(-,K)$
12	$\mathsf{Enc}_{\mathrm{I}}(-,K)$
16	$\mathsf{Enc}_{\mathrm{I}}((16,17),K)$

Figure 7.33.: Bucket Index for the attribute *name* and the attribute value *Johnson*. A row in the Bucket Index holds the start of the interval of row ids as well as the encrypted list of row ids the attribute value occurs in this bucket. This optimisation is similar to the B-Tree Index but relies directly on the indexing mechanisms of the database management system.

Here, i is the length of the interval and r is the row id. Then, $b(r)$ is the start of the bucket for the row id r. Consider for example, we want to check if the value of attribute *name* in the rows 3 and 8 is *Johnson*. In our example the bucket size i is 3. Therefore, we calculate:

$$b(3) = (3 \text{ div } 4) \cdot 3 + 1 = 0 \cdot 3 + 1 = 1$$

and

$$b(8) = (8 \text{ div } 4) \cdot 3 + 1 = 2 \cdot 3 + 1 = 7$$

and fetch the rows of the Bucket Index with $start = 1$ and $start = 7$. In the first case, the attribute value of the row in question is *Johnson*, in the latter case it is not, since the Bucket that starts with row id 7 is empty. For a given row id, calculating the bucket can be done in $O(1)$, while fetching the bucket relies on the index structure of the database management system. The overhead for fetching a bucket usually is between $O(1)$ and $O(ld(\frac{n}{t}))$ while n is the number of rows in the database and t is the interval size of the buckets.

Similar to the B-Tree optimisation, the Bucket optimisation has the drawback of the need to store empty buckets. A solution for this is encrypting the *start* column of the Bucket Index. Then, as it is the case with the B-Tree optimisation, we trade storage space overhead for additional encryption overhead. For our benchmarks in Section 7.6.2 we chose to accept the additional storage overhead and store the start of a bucket in plaintext.

7.6. Comparison and Benchmarks

The goal of the optimisations proposed in Sections 7.1-7.5 is the reduction of the decryption overhead as well as of the message overhead for queries. In order to provide a comparison of the overhead, we compare the overhead of each optimisation on the O calculus in Section 7.6.1. Since, the overall performance of an optimisation depends on the data in the database as well as on the queries, we provide benchmarks of selected optimisations in Section 7.6.2.

7.6.1. Encryption Overhead and Space Requirements

For each optimisation, we examine the number of row ids we need to decrypt in order to check if a row id is in an index. Figure 7.34 provides an overview of the overheads for

each optimisation.

optimisation	decrypted row ids for retrieval of a single row id
List	$O(l)$
Compression	$O(l)$
Binary Search	$O(\text{ld } l)$
Keyed Hash	$O(1)$
B-Tree	$O(t)$
Bucket	$O(t)$

Figure 7.34.: Complexity class of the number id row ids needed to decrypt in order to look up a single row id, while l is the number of row ids of an attribute value and t is the interval, leaf, or bucket size.

Without any optimisation, the number of row ids, the adaptor needs to decrypt in order to look up a single row id scales linearly with the number of row ids of the corresponding attribute, which depend on the size of the database.

The Compression optimisations are similar to the standard indices. They only decrease the overhead by a value that depends on the compression method an on the data in the database.

The Binary Search decreases the decryption overhead to $O(\text{ld } l)$ while only needing to store an additional value, the length of the list (cf. Section 7.2).

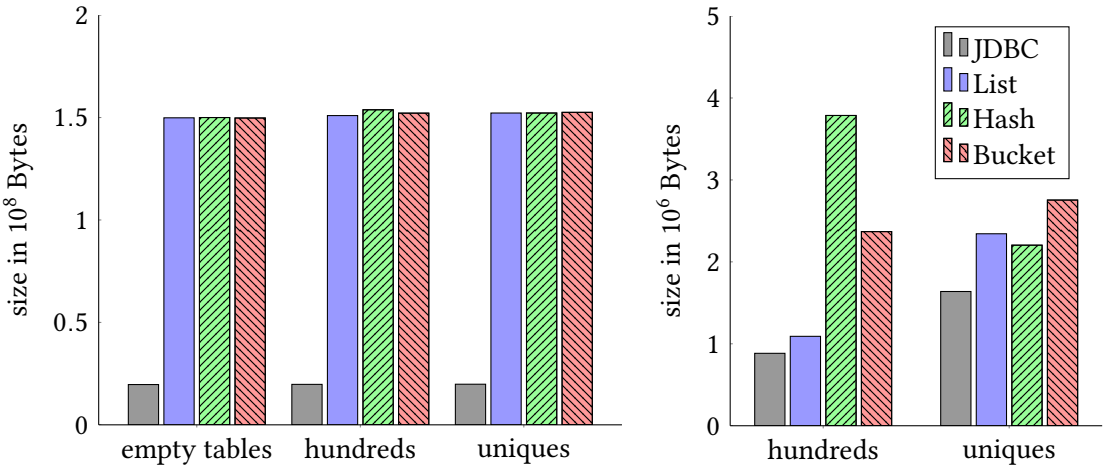
With the Keyed Hash optimisation, the row ids are stored in plaintext. Furthermore, this optimisation has the potential for efficient execution of two classes of queries. However, with this optimisation, the IND-ICP security notion is not fulfilled (cf. Section 7.29). The storage overhead of the Keyed Hash optimisation is the space needed for the sequence numbers, their hash values (h), as well as the row ids in plaintext.

Since the B-Tree optimisation and the Bucket optimisation, both, partition the space of row ids into buckets, the number of row ids needed to be decrypted for a single row id depends on the number of ids in a bucket ($O(t)$).

For the Bucket optimisation, we need to store each each bucket ($t + r + m$) as well as a starting value for each bucket ($m \cdot l$). The B-Tree optimisation needs additional storage space for the tree structure and, for each leaf, a pointer to the next leaf.

The Compression, the Binary Search and the Keyed Hash optimisations are more space efficient than the B-Tree and the Bucket optimisations. On the other hand, the Keyed Hash, the B-Tree, and the Bucket optimisations require less decryptions of row ids. The Keyed Hash seems to be the best optimisation for decreasing decryption overhead as well as in storage space overhead.

For real applications, we have to consider how a DBMS handles the additional space requirements. A DBMS may introduce additional overhead or compression. Therefore, we measured the space needed on the hard disk for the plain data set, the adaptor without optimisations (List), the adaptor with the Keyed Hash optimisation, and the adaptor with the Bucket optimisation for the uniques and the hundreds data set of the benchmarks from Section 6.1. Figure 7.35 shows the results. Figure 7.35.1 shows the absolute space required for just the empty tables, the hundreds data set, and the uniques data set. Figure 7.35.2



7.35.1: absolute 7.35.2: relative

Figure 7.35.: Comparison of absolute and relative disk space requirements of the plain data sets and the data sets transformed with the database outsourcing schemes and its optimisations. The figure for relative space requirements shows the space needed by the data without the space needed by the empty tables.

shows the disk space requirements of the data without the initial overhead for the table structures.

Irregardless of the optimisation, data transformed by the adaptor needs roughly 7-8 times the space of the plain data sets. The comparison of Figures 7.35.1 and 7.35.2 shows that the most space is needed for the data structures. For the data sets used (10,000 entries, each), the actual data does only make up a fraction of the space requirements. Depending on the optimisation, the data transformed by the adaptor needs about 1-4 times the space of the plain data.

7.6.2. Benchmarks

Hardware We use the same set-up and hardware as in the benchmarks of the MimoSecco adaptor in Section 6.1 in the MimoSecco scenario (cf. Figure 7.16.3). The benchmark component as well as the adaptor is deployed on the client with an Intel Pentium Dualcore @ 2Ghz and with 2 GB RAM. The back end database (PostreSQL 9.1) is deployed on a server with 2 AMD Opteron with 6 Cores @ 2.4Ghz, each, and with 32 GB of RAM.

Queries and Data Since we want to benchmark the scaling effects we did not use the queries and the data set of the *AS/sup 3/AP* benchmark [TOB89], but created data sets with with 1,000, 2,000, 4,000, 8,000, 16,000, and 32,000 entries, each. The scheme of the data set, as well as an excerpt is depicted in Figure 7.36. Each data set has the attributes *gender*, *prename*, *name*, and *age*. The attribute *gender* has a distribution of 7% female and 93% male. The names are from a service called *Fake Name Generator* [Cor]. The values of the attribute *age* are chosen uniformly at random from $[1, 99]$.

The queries used in this benchmark are the INSERT queries needed to initially create each data set and SELECT queries with different properties and result sets. Figure 7.37 depicts the SELECT queries used in this benchmark. In the benchmarks, we used a simple

gender	prename	name	age
male	Frank	Kluge	23
male	Marko	Schmid	4
male	Mike	Herrmann	95
male	Jens	Fenstermacher	54
female	Sabrina	Moeller	33

Figure 7.36.: An excerpt of the data sets used in the optimisations benchmarks.

Query Nr.	Query
S1	SELECT * FROM users WHERE prename = Maximilian
S2	SELECT * FROM users WHERE prename = Maximilian AND gender = male
S3	SELECT * FROM users WHERE name = Moeller AND gender = female

Figure 7.37.: The SELECT queries used in the scaling benchmark. The query S1 is a simple query with one condition. In the data set used, 93% of all tuples fulfil the condition *gender = male*.

SELECT query with one condition and two SELECT queries with two conditions, each. The second condition of the second query is fulfilled by 93% of all tuples, while the second condition of the third query is fulfilled by only 7% of all tuples.

Results According to Figure 7.34, the Compression optimisations behave similar to the standard implementation. Furthermore, the Bucket optimisation behaves similar to the B-Tree optimisation which, in turn, is an extension to the Binary Search optimisation (cf. Section 7.4). Therefore, we compare the standard implementation (List), the Keyed Hash optimisation, and the Bucket optimisation. For the Bucket optimisation, we chose a bucket size of 10.

Figures 7.38 to 7.39 depict the benchmark results. More detailed benchmark results can be found in the appendix in Chapter A.3. The results show that for INSERT queries, the Hash and the Bucket optimisation scale better than the standard implementation. As expected, the Bucket optimisation has the lowest overhead. The Bucket optimisation has a fixed size of buckets and, therefore, each row in the index tables only holds a fixed number

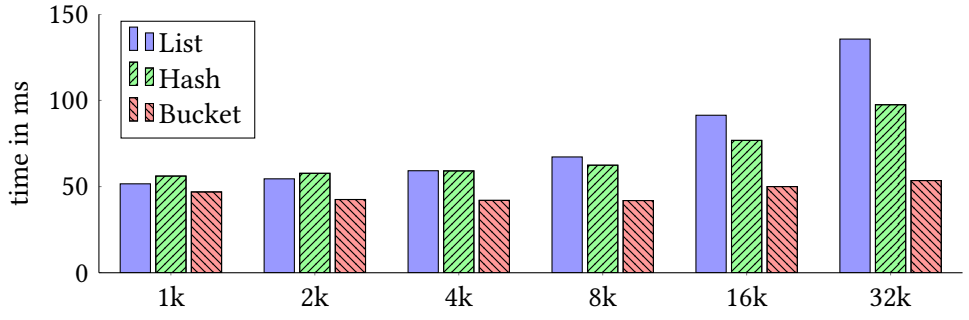


Figure 7.38.: The average execution time in ms of an insert operation depending on the optimisation and the size of the data set.

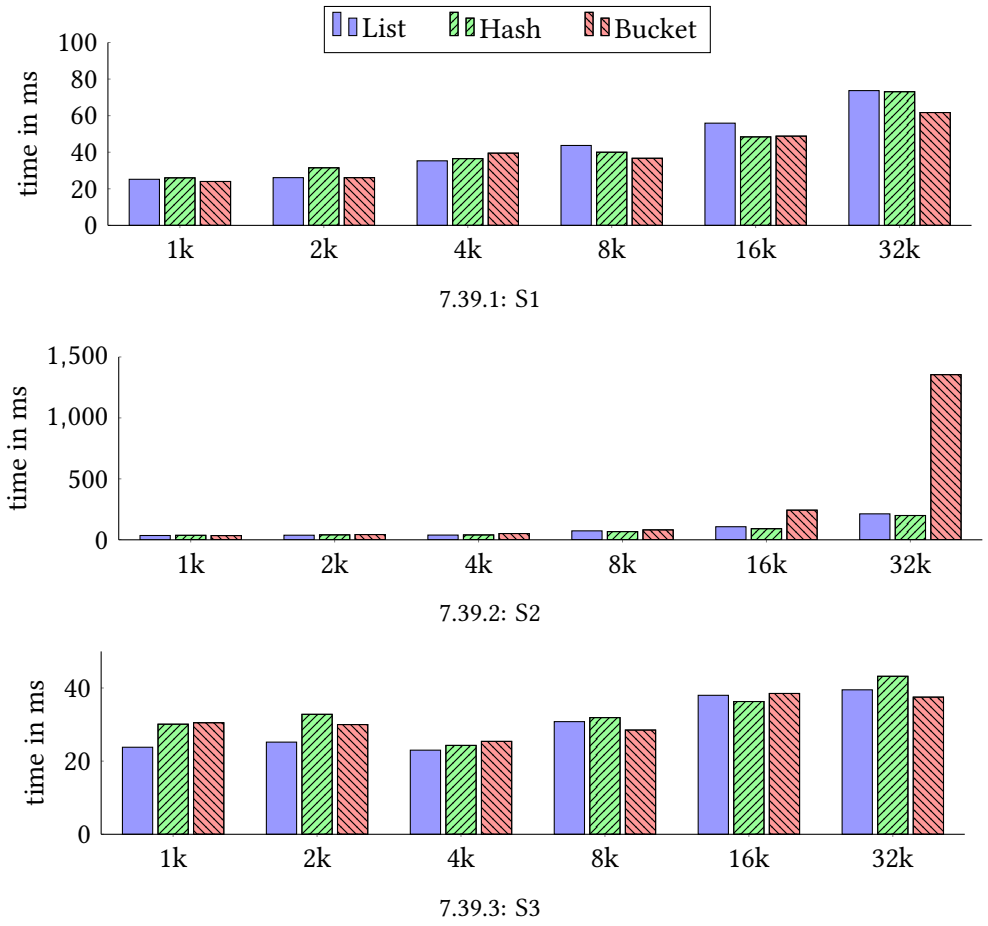


Figure 7.39.: Average execution times in ms of the select queries S1, S2, and S3 depending on the optimisation and the size of the data set.

of row ids. Consequently, the Bucket optimisation leverages the search optimisations of the back end database better than the standard adaptor or the Hash optimisation.

Figure 7.39 depicts the benchmark results for the SELECT queries S1, S2, and S3. The results for S1 suggest, that the Keyed Hash and the Bucket optimisations scale better for attributes with many different and equally distributed values. The results for S2 show, that the bucket optimisation does not scale very well for queries that affect a high number of buckets. Since 93% of all rows are affected by the query S2, the adaptor has to query nearly all buckets. The results for select query S3 show that there is not much difference in the overheads of the optimisations if a very small number of rows is affected.

Overall, the Bucket optimisation seems to be most suited for attributes with many different values. For attributes with a distribution similar to the attribute *gender* in our test data sets, the Bucket optimisation should not be used. Please note, that for each attribute an individual index optimisation can be used.

8. Side Channels in Secure Database Outsourcing

When implementing or using implementations of cryptographic primitives, protocols, or schemes, so-called side channels have to be considered. Side channels are security issues of implementations. Cryptographic models that are used to prove security are

abstractions and, therefore, do not capture every aspect of the real world. For example power consumption is not covered in classic cryptographic security models. There are so called power attacks, that allow an adversary to learn partially or fully the secret input of a cryptographic scheme by observing the power consumption. In this case, observing the power consumption and deducing parts of the secret input is a side channel attack. Other common examples for side channels are electromagnetic emission or timing.

When a provably secure method, protocol, or scheme is implemented and used, from a security point of view, there is a context switch from a formal model to the real world. This context switch can introduce side channels. Note that side channels are not security relevant bugs of an implementation but can occur even if the implementation itself is bug free.

For the security proof of our secure database outsourcing scheme, we also used an abstract model. Consequently, it is important to study possible side channels of implementations of this scheme that exploit aspects of reality that are not covered in the security model. In this section, we will discuss potential security issues of implementations of database outsourcing schemes on the example of the MimoSecco database outsourcing scheme. The side channels discussed in this section, however, are not specific our scheme but have to be considered when implementing database outsourcing schemes in general. We will analyse the side channels presented in this section and discuss how to avoid them.

8.1. Exclusion of Possible Database Contents

In contrast to Bayes Privacy, the model that we used to prove the security of our secure database outsourcing scheme does not consider background specific knowledge of adversaries. In the experiment $\text{IND-ICP}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}$ (cf. Definition 14), the adversary has to choose two databases. The adversary chooses the first database directly and the second one by choosing an independent column permutation of the first one. Therefore, the model implicitly assumes that all possible database have equal probability. We pointed this out in Chapter 4, Section 6.2, where we modelled the security notion IND-ICP as a Computational Bayes Privacy notion.

In reality, an adversary would not choose a database, but is presented with the outsourced database and wants to determine one or several relations of the original database. If the adversary has specific background knowledge about the domain of the content of the database, she can potentially exclude possible original databases. As an example, consider the tables in Figure 7.40.2. These tables depict a database that has been transformed according to Enc_{DB} of the MimoSecco database outsourcing scheme (cf. Definition 69). Due to the low number of rows of the data table (2), there are only two possible original databases. These two candidates for the original database are depicted in Figure 7.40.1. An adversary with the background knowledge that men can not be pregnant, however, can rule out the second database in Figure 7.40.1 as the original database, which gives the adversary a non negligible advantage in the security game $\text{IND-ICP}^{\mathcal{A}}_{(\text{Gen},\text{Enc})}$ and, consequently, breaks IND-ICP.

This problem is not specific to our scheme. According to the *No-Free-Lunch Theorem* [KM11] this problem is always present in schemes that allow adversaries to learn something about the database, even if the model captures specific background knowledge. As a result, for the security of a database outsourcing scheme it is vital to check if the

name	pregnant
Alice	yes
Bob	no

7.40.1: d_0

name	pregnant
Alice	no
Bob	yes

7.40.1: d_1

values	rows
Alice	$\mathsf{Enc}_I(1, K)$
Bob	$\mathsf{Enc}_I(2, K)$

values	rows
yes	$\mathsf{Enc}_I(1, K)$
no	$\mathsf{Enc}_I(2, K)$

row	data (name, condition)
1	$\mathsf{Enc}_I((\mathit{Alice}, \mathit{yes}), K))$
2	$\mathsf{Enc}_I((\mathit{Bob}, \mathit{no}), K)$

7.40.2: $\mathcal{M}(d_0) = \mathcal{M}(d_1)$

Figure 7.40.: Potential original databases 7.40.1 and the tables of an outsourced database 7.40.2. With domain specific background knowledge, it is possible to determine the original database.

assumptions about the background knowledge still hold given the data to be outsourced.

8.2. Usage of the Database

Static security notions such as IND-ICP do not consider the usage of the outsourced database. In the Experiment $\text{IND-ICP}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}$ (Definition 14), the adversary can not observe the execution of queries or has acces to access statistics of the database. In reality, adversaries that observe how the database is used are possible and have to be considered. They can, for example, observe sequences of queries and the their result.

We examine notions for data outsourcing schemes for adversaries that observe the usage of outsourced databases in Chapter 5, Section 3.1.2 and Chapter 6, Section 3. We present a scheme with provable security against adversaries that observe queries in Section 5. Nevertheless, we discuss these side channels as they may aid designers and programmers in implementations of data outsourcing schemes.

Access Statistics Rows in the index and data tables that are queried in short intervals are more likely to be correlated than other rows. By observing access patterns over time, an adversary can learn the parts of the database that are queried often. Attacks involving access statistics, however, are not specific to our database outsourcing scheme. Even if a database is completely encrypted, an adversary with access to access statistics can gain information about the database [KC05]. There are so-called *private information retrieval* (PIR) schemes [Cho+98] that consider this issue. For databases the notion of PIR implies the notion of Query Privacy (cf. Chapter 5, Section 5).

As mentioned above, countermeasures to this attack involve PIR or mechanisms that prevent an adversary from learning the information a client queries from the server. Such mechanisms, however, do have a very high overhead and are not applicable under the performance constraints of most realistic scenarios.

Another method to counter this attack is to rely on architectural assumptions (as in e. g. [Agg+05]). For example we can distribute each table to a different server. Assuming

they do not collaborate maliciously a single server does not have access to the access statics of the other servers and therefore can not conduct the attack described above.

Database Updates Adversaries can not only observe the execution of queries on an outsourced database. They also can observe how queries change the database. In contrast to message encryption or searchable encryption, a database can change over time. Queries can add, remove, and change tuples. Consider for example Figure 7.41. This Figure depicts

values	rows
Alice	$\mathsf{Enc}_\mathrm{I}(1, K)$
Bob	$\mathsf{Enc}_\mathrm{I}(2, K)$
Eve	$\mathsf{Enc}_\mathrm{I}(3, K)$

values	rows
flu	$\mathsf{Enc}_\mathrm{I}((1, 3), K)$
cancer	$\mathsf{Enc}_\mathrm{I}(2, K)$

row	data (name, condition)
1	$\mathsf{Enc}_\mathrm{I}((\mathit{Alice}, \mathit{flu}), K)$
2	$\mathsf{Enc}_\mathrm{I}(\mathit{Bob}, \mathit{cancer}), K)$
3	$\mathsf{Enc}_\mathrm{I}((\mathit{Eve}, \mathit{flu}), K)$

7.41.1: The tables $\mathit{patients}_{\mathit{index},\mathit{name}}$, $\mathit{patients}_{\mathit{index},\mathit{condition}}$, and $\mathit{patients}_{\mathit{data}}$ prior to the update

values	rows
Alice	$\mathsf{Enc}_\mathrm{I}(1, K)$
Bob	$\mathsf{Enc}_\mathrm{I}(2, K)$
Eve	$\mathsf{Enc}_\mathrm{I}(3, K)$
Carol	$\mathsf{Enc}_\mathrm{I}(4, K)$

values	rows
flu	$\mathsf{Enc}_\mathrm{I}((1, 3), K)$
cancer	$\mathsf{Enc}_\mathrm{I}(2, K)$
lupus	$\mathsf{Enc}_\mathrm{I}(4, K)$

row	data(name, condition)
1	$\mathsf{Enc}_\mathrm{I}((\mathit{Alice}, \mathit{flu}), K)$
2	$\mathsf{Enc}_\mathrm{I}((\mathit{Bob}, \mathit{cancer}), K)$
3	$\mathsf{Enc}_\mathrm{I}((\mathit{Eve}, \mathit{flu}), K)$
4	$\mathsf{Enc}_\mathrm{I}((\mathit{Carol}, \mathit{lupus}), K)$

7.41.2: example tables after the update

Figure 7.41.: Example for an outsourced database before 7.41.1 and after 7.41.2 execution of Query 7.3. An adversary observing both states can determine the new tuple.

the states of a database before (Figure 7.41.1) and after (Figure 7.41.2) the execution of the following update query:

$$\text{INSERT INTO } \mathit{patients} \text{ VALUES (Carol, lupus)} \tag{7.3}$$

Given the tables in Figure 7.41.1, this query will be transformed into the following INSERT queries:

$$\begin{aligned}
&\text{INSERT INTO } \mathit{patients}_{\mathit{data}} && \text{VALUES } \mathsf{Enc}_I \textit{ ((Carol, lupus), K)}\\
&\text{INSERT INTO } \mathit{patients}_{\mathit{index},\mathit{name}} && \text{VALUES } \textit{(Carol, } \mathsf{Enc}_I \textit{ (4, K))}\\
&\text{INSERT INTO } \mathit{patients}_{\mathit{index},\mathit{condition}} && \text{VALUES } \textit{(lupus, } \mathsf{Enc}_I \textit{ (4, K))}
\end{aligned}$$

If an adversary observes the state before and after execution of Query 7.3, she can determine the new tuple *(Carol, lupus)* inserted into the database and therefore get information she should not get according to the static security notion. This is possible since the security model does not consider database updates.

This attack can be prevented, for example, by methods like distributing the tables to different severs, caching updates, or encryption of the index tables.

Our framework for notions for data outsourcing in Chapter 5, Section 3 as well as the extension of our outsourcing scheme in Chapter 7, Section 5 already consider this side channel, since the database is in the view (cf. Definition 41) of the server. The MimoSecco

and Cumulus4j implementations, however, do not since they only provide IND-ICP which is a Static Security notion.

For future implementations of database outsourcing schemes, this side channel should be considered. In contrast to usage scenarios for message encryption, updates are an important aspect of databases.

8.3. Order of Values on Physical Storage

Using sets or multisets is a common approach in order to model databases (e. g. [Hac+02; NC11; Agg+05] and in this work in Chapter 2, Definition 6). Sets and mutisets do not have a specific order of their elements. In some cases, however, for example if we want to address single tuples of the database, we assume it to be ordered.

The elements of real databases, always do have an order. Databases are stored on physical devices that are organised in storage cells with addresses. Additionally some storage devices use file systems that also introduce an order of stored entities. Consequently, elements of databases stored on physical devices do have an order implied by the storage device or the file system. This order can potentially be different to the order used in the abstract model.

An adversary can exploit this order and potentially break the security of the data outsourcing scheme. This problem, is not specific to our outsourcing scheme. Any scheme, that stores data in plain text on the back end (e. g. [NC11; De +10]) is potentially vulnerable to this kind of attack.

In realistic scenarios, application data is stored in the database over time by using the application. Consequently, the order of data on the physical device can reflect the order it has been stored in the database.

For our database outsourcing schemes, this means that in such a scenario, the adversary may learn correlations of attribute values by simply reconstructing them from the physical order of the entries of the index table on the storage device. This attack is similar to the update attack described in Section 8.2. Now, however, the adversary does not need to observe different states of the database. She can reconstruct different states simply by the order of the tuples stored on the physical storage device. Consider for example Figure 7.42. If an adversary knows by the order of the data on the physical storage, as indicated by the additional column *storage order* that the tuples (Eve, $\mathsf{Enc}_\mathrm{I}(1, K)$) and (yes, $\mathsf{Enc}_\mathrm{I}(1, K)$) were added after the other tuples in their respective tables, she learns that the original database has to be the second table in Figure 7.42.1.

Please note, that the row ids (the attribute values of the *row* columns) do not necessarily reflect the temporal order of their creation. In this work, for the sake of readability, the row ids are subsequent numbers. In implementations, there can be missing row ids due to DELETE queries or due to mechanisms that generate new row ids completely different to using subsequent numbers.

This side channel can be prevented by either enforcing the order of the database and the model to be the same or by enforcing a random order on the physical storage device or file system. This, however, induces a substantial performance overhead. Another solution could be caching changes until there are l cached and independent queries and then executing them in random order. The overall system would then provide IND-ICP only for blocks of size l in the database (cf. Definition 57).

Another method that prevents this side channel is the encryption of the index tables.

name	pregnant
Alice	yes
Eve	no

name	pregnant
Alice	no
Eve	yes

7.42.1: Potential original *patients* tables for the Tables in Figures 7.42.2-7.42.4

storage order	values	rows
1	Alice	$\mathsf{Enc}_{\mathrm{I}}(2, K)$
2	Eve	$\mathsf{Enc}_{\mathrm{I}}(1, K)$

7.42.2: $patients_{index,name}$

storage order	values	rows
1	no	$\mathsf{Enc}_{\mathrm{I}}(2, K)$
2	yes	$\mathsf{Enc}_{\mathrm{I}}(1, K)$

7.42.3: $patients_{index,condition}$

storage order	row	data (name, condition)
1	2	$\mathsf{Enc}_{\mathrm{I}}((Alice, no), K)$
2	1	$\mathsf{Enc}_{\mathrm{I}}((Eve, yes), K)$

7.42.4: $patients_{data}$

Figure 7.42.: An example for *patients* database (Figures 7.42.2-7.42.4) oursourced with $\mathsf{Enc}_{\mathrm{DB}}$ of the MimoSecco database outsourcing scheme with an additional column storage order that indicates the order of the tuples on the physical storage device. Potential original databases are shown in Figure 7.42.1. If the adversary knows that the tuples in the database were added subsequently and learns the order of the tuples on the storage device, she learns that the original database is the second table of Figure 7.42.1.

This either restricts the number of queries that can be executed efficiently or introduces an additional overhead. For example in order to efficiently support substring queries, searchable encryption has to be used for the attribute values. This introduces additional index structures and the need for additional queries as well additional decryption steps.

8.4. Active Adversaries

In our security models, we only consider honest but curious or passive adversaries. Therefore, the goal of our scheme is not to improve the security of a database system against active adversaries. For example, an actively malicious database server could manipulate results before sending them to the client or even suppress answers. Active adversaries are out of the scope of this work. However, for the sake of completeness, and in order to show potential future research directions, we discuss these attacks in the remainder of this section.

8.4.1. Manipulation of Results

Instead of returning the correct answer for a query issued to the database, an active adversary can exchange encrypted parts of a result. Consider the example an index table with two rows, and a query with the result of the *rows* cell of the first row of this

index table. The adversary can intercept this result and replace it with *rows* cell of the second row of this index table. This leads to an inconsistent view of the database and can compromise the functionality of the client application. It can, however, be detected. One method to detect this specific behaviour is to add the value of the *values* attribute of each row to its encrypted list of row ids. Please note, that this measure, however, does not prevent all possible manipulations of results. Other methods can involve cryptographic signature schemes, timestamps, distribution and cryptographic proofs.

8.4.2. Attacks on Availability

Instead of manipulating results prior to sending them to the client, an actively malicious adversary also can delay or completely suppress results. Similar to manipulation, such behaviour can compromise the functionality of the client application. Such behaviour, also, can be easily detected by the client. A straightforward solution to a database service that does not fulfil a certain quality of service is to switch the service provider.

8. Conclusion and Outlook

We will conclude this thesis by a short summary of Chapters 3 to 7 and with a discussion of problems left open and potential directions of future research.

Data Privacy and the Bayes Privacy Framework In Chapters 3 and 4, we provided an introduction to database privacy and presented the Bayes Privacy framework. The Bayes Privacy framework, however, does not fulfil the privacy axioms postulated by Kifer and Lin in [KL10] that allow for a thorough mathematical examination of notions fulfilling these axioms. An examination of the subset of the Bayes Privacy framework that fulfils the privacy axioms and its properties could provide more insights into the theoretical aspects of data privacy. On the other hand, support for the design of mechanisms that fulfil notions defined in a privacy framework would be of great benefit for practical applications and contribute to closing the gap that exists between theoretical research of data privacy and application of data privacy methods in practice. Another potentially interesting direction of research is the extension of the Bayes Privacy framework, for example, by the inclusion of the expected number of individuals affected by a breach into the framework. This can lead to notions that represent more detailed trade-off decisions. Furthermore a more detailed examination of composability properties of formal privacy notions would enable profound reasoning about the security of composed mechanisms.

Privacy Notions for Data and Database Outsourcing In Chapters 5 and 6, we presented a model for data outsourcing, meta notions for the privacy goals of data outsourcing, and instantiations of these meta notions for database outsourcing. We established relations between the basic notions Static Security, Data Privacy, Query Privacy, and Result Privacy. An interesting question is if and how these relations hold for the generalised notions for data outsourcing. We showed on the example of database outsourcing, that provable security with a meaningful guarantee can be achieved under practicability constraints. There are, however, database outsourcing schemes without a formal security guarantee (cf. Section 2 in Chater 7 and Section 3 in Chapter 6) that intuitively offer some form of security. Capturing their security properties with formal notions would help evaluate and compare the security of such schemes. Furthermore, on a broader scope, this will be an additional contribution bridging the gap between the fields of provable security and practical security mechanisms.

Mechanisms for Database Outsourcing In Chapter 7, we presented schemes and implementations for secure database outsourcing. We argued that the overhead of a security mechanism for data outsourcing should, in order to be used, not cancel out the benefits of outsourcing. This may not be true for all scenarios. Furthermore, since the formalisation of the benefits of outsourcing is out of the scope of this work, we could not prove that implementations of our candidate, the MimoSecco scheme, can fulfil such economical constraints. We, however, presented benchmarks that point into a promising direction.

Furthermore, the implementations provided leave room for additional optimisations such as an automated selection of the index optimisation based on the data and the queries, an extension to provide security even if the adversary observes query executions for example through the scheme provided in Chapter 7, Section 5 or through the introduction of shuffles and caching such as in [Vim+15]. Additionally, efficient support for more complex operations that allow for secure outsourcing of data analysis steps would be of benefit for privacy preserving data mining applications.

Author's Publications

[Ach+16] Dirk Achenbach et al. "Closing the Gap: A Universal Privacy Framework for Outsourced Data". English. In: *Cryptography and Information Security in the Balkans*. Ed. by Enes Pasalic and Lars R. Knudsen. Vol. 9540. Lecture Notes in Computer Science. Springer International Publishing, 2016, pp. 134–151. ISBN: 978-3-319-29171-0. URL: http://dx.doi.org/10.1007/978-3-319-29172-7_9.

[AGH11] Dirk Achenbach, Matthias Gabel, and Matthias Huber. "Improving Complex Systems Today: Proceedings of the 18th ISPE International Conference on Concurrent Engineering". In: ed. by D. Daniel Frey, Shuichi Fukuda, and Georg Rock. London: Springer London, 2011. Chap. MimoSecco: A Middleware for Secure Cloud Storage, pp. 175–181. ISBN: 978-0-85729-799-0. URL: http://dx.doi.org/10.1007/978-0-85729-799-0_20.

[Hei+10] Clemens Heidinger et al. "Privacy-aware Folksonomies". In: *Proceedings of the 14th European Conference on Research and Advanced Technology for Digital Libraries*. ECDL'10. Glasgow, UK: Springer-Verlag, 2010, pp. 156–167. ISBN: 3-642-15463-8, 978-3-642-15463-8. URL: http://dl.acm.org/citation.cfm?id=1887759.1887783.

[HH14] Matthias Huber and Gunnar Hartung. "Trusted Cloud Computing". In: ed. by Helmut Krcmar, Ralf Reussner, and Bernhard Rumpe. Cham: Springer International Publishing, 2014. Chap. Side Channels in Secure Database Outsourcing on the Example of the MimoSecco Scheme, pp. 35–48. ISBN: 978-3-319-12718-7. URL: http://dx.doi.org/10.1007/978-3-319-12718-7_3.

[HM11] Matthias Huber and Jörn Müller-Quade. "Methods to Secure Services in an Untrusted Environment". In: *Software Engineering 2011: Fachtagung des GI-Fachbereichs Softwaretechnik, 21.-25. Februar 2011 in Karlsruhe*. Ed. by Ralf Reussner et al. Vol. 183. LNI. GI, 2011, pp. 159–170. ISBN: 978-3-88579-277-2.

[HMN13] Matthias Huber, Jörn Müller-Quade, and Tobias Nilges. "Number Theory and Cryptography: Papers in Honor of Johannes Buchmann on the Occasion of His 60th Birthday". In: ed. by Marc Fischlin and Stefan Katzenbeisser. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Chap. Defining Privacy Based on Distributions of Privacy Breaches, pp. 211–225. ISBN: 978-3-642-42001-6. URL: http://dx.doi.org/10.1007/978-3-642-42001-6_15.

[HMN14] Matthias Huber, Jörn Müller-Quade, and Tobias Nilges. "Structural Composition Attacks on Anonymized Data". In: *GI Informatik 2014 - Sicherheit, Schutz und Zuverlässigkeit*. 2014, pp. 443–460.

[Hub+11] Matthias Huber et al. "Towards Secure Cloud Computing through a Separation of Duties". In: *Informatik 2011: Informatik schafft Communities, Beiträge der 41. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 4.-7.10.2011, Berlin (Abstract Proceedings)*. Ed. by Hans-Ulrich Heißand Peter Pepper, Holger Schlingloff, and Jörg Schneider. Vol. 192. LNI. GI, 2011. ISBN: 978-88579-286-4-7.

[Hub+13] Matthias Huber et al. "Security Engineering and Intelligence Informatics: CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Germany, September 2-6, 2013. Proceedings". In: ed. by Alfredo Cuzzocrea et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Chap. Cumulus4j: A Provably Secure Database Abstraction Layer, pp. 180–193. ISBN: 978-3-642-40588-4. URL: http://dx.doi.org/10.1007/978-3-642-40588-4_13.

[Hub10] Matthias Huber. "Towards Secure Services in an Untrusted Environment". In: *Proceedings of the Fifteenth International Workshop on Component-Oriented Programming (WCOP) 2010*. Ed. by Barbora Bühnová et al. Vol. 2010-14. Interne Berichte. Karlsruhe, Germany: Karlsruhe Institue of Technology, Faculty of Informatics, June 2010, pp. 39–46. ISBN: ISSN 1432 - 7864. URL: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000018464.

Students' Theses

[Bar14] Maximilian Baritz. *Evaluation von Optimierungen des Index eines Schemas zur sicheren Auslagerung von Datenbanken*. Bachelorarbeit. 2014.

[Boe13] Jonas Boehler. *Optimierung des MimiSecco-Datenbankadapters*. Studienarbeit. 2013.

[Har11] Gunnar Hartung. *Sicherheitsanalyse eines Verfahrens zur transparenten Datenbankverschlüsselung*. Bachelorarbeit. 2011.

[Mar13] Jan-Frederic Markert. *Differentially Private One Way Trapdoor Functions for Database Outsourcing*. Bachelorarbeit. 2013.

References

[ABO07] Georgios Amanatidis, Alexandra Boldyreva, and Adam O'Neill. "Provably-secure Schemes for Basic Query Support in Outsourced Databases". In: *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*. Redondo Beach, CA, USA: Springer-Verlag, 2007, pp. 14–30. ISBN: 978-3-540-73533-5. URL: http://dl.acm.org/citation.cfm?id=1770560.1770563.

[Ado] Adobe Systems Inc. *Adobe Creatice Cloud*. URL: http://www.adobe.com/creativecloud.html (visited on 15/10/2015).

[Agg+05] Gagan Aggarwal et al. "Two Can Keep a Secret: A Distributed Architecture for Secure Database Services". In: *The Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*. 2005. URL: http://ilpubs.stanford.edu:8090/659/.

[Ama] Amazon Web Services Inc. *Amazon Workspaces*. URL: http:/aws.amazon.com/workspaces/ (visited on 15/10/2015).

[Aut] Autodesk Inc. *Pixlr*. URL: https://pixlr.com/editor/ (visited on 15/10/2015).

[AX] AX Business Solutions AG. *Ax Easy*. URL: http://www.ax-easy.de/ (visited on 04/03/2016).

[BA05] Roberto J. Bayardo and Rakesh Agrawal. "Data Privacy Through Optimal k-Anonymization". In: *Proceedings of the 21st International Conference on Data Engineering*. ICDE '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 217–228. ISBN: 0-7695-2285-8. URL: http://dx.doi.org/10.1109/ICDE.2005.42.

[BDG13] Johannes Buchmann, Denise Demirel, and Jeroen van de Graaf. "Towards a Publicly-Verifiable Mix-Net Providing Everlasting Privacy". In: *Financial Cryptography*. 2013, pp. 197–204.

[Bha+11] Raghav Bhaskar et al. "Noiseless Database Privacy". In: *Proceedings of the 17th International Conference on The Theory and Application of Cryptology and Information Security*. ASIACRYPT'11. Seoul, South Korea: Springer-Verlag, 2011, pp. 215–232. ISBN: 978-3-642-25384-3. URL: http://dx.doi.org/10.1007/978-3-642-25385-0_12.

[BLR08] Avrim Blum, Katrina Ligett, and Aaron Roth. "A Learning Theory Approach to Non-interactive Database Privacy". In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. STOC '08. Victoria, British Columbia, Canada: ACM, 2008, pp. 609–618. ISBN: 978-1-60558-047-0. URL: http://doi.acm.org/10.1145/1374376.1374464.

[Blu+05] Avrim Blum et al. “Practical Privacy: The SuLQ Framework”. In: *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS ’05. Baltimore, Maryland: ACM, 2005, pp. 128–138. ISBN: 1-59593-062-0. URL: http://doi.acm.org/10.1145/1065167.1065184.

[Bun78] Deutscher Bundestag. *Bundesdatenschutzgesetz*. Jan. 1978. URL: http://www.gesetze-im-internet.de/bdsg_1990/ (visited on 04/12/2014).

[Can01] R. Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*. FOCS ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 136–. ISBN: 0-7695-1390-5. URL: http://dl.acm.org/citation.cfm?id=874063.875553.

[CAS] CAS Software AG. *CAS Software AG*. URL: http://www.cas.de/ (visited on 04/03/2016).

[Cas+13] David Cash et al. “Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I”. In: ed. by Ran Canetti and Juan A. Garay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Chap. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries, pp. 353–373. ISBN: 978-3-642-40041-4. URL: http://dx.doi.org/10.1007/978-3-642-40041-4_20.

[Cas+14] David Cash et al. “Dynamic searchable encryption in very-large databases: Data structures and implementation”. In: *Network and Distributed System Security Symposium, NDSS*. Vol. 14. 2014.

[Ces+05] Alberto Ceselli et al. “Modeling and Assessing Inference Exposure in Encrypted Databases”. In: *ACM Trans. Inf. Syst. Secur.* 8.1 (Feb. 2005), pp. 119–152. ISSN: 1094-9224. URL: http://doi.acm.org/10.1145/1053283.1053289.

[Cho+98] Benny Chor et al. “Private Information Retrieval”. In: *J. ACM* 45.6 (Nov. 1998), pp. 965–981. ISSN: 0004-5411. URL: http://doi.acm.org/10.1145/293347.293350.

[Cir+10] Valentina Ciriani et al. “Combining Fragmentation and Encryption to Protect Privacy in Data Storage”. In: *ACM Trans. Inf. Syst. Secur.* 13.3 (July 2010), 22:1–22:33. ISSN: 1094-9224. URL: http://doi.acm.org/10.1145/1805974.1805978.

[Cit] Citrix Systems Inc. *Citrix Daas*. URL: http://www.citrix.com/products/daas/overview.html (visited on 15/10/2015).

[CM06] Kamalika Chaudhuri and Nina Mishra. “When Random Sampling Preserves Privacy”. In: *Proceedings of the 26th Annual International Conference on Advances in Cryptology*. CRYPTO’06. Santa Barbara, California: Springer-Verlag, 2006, pp. 198–213. ISBN: 3-540-37432-9, 978-3-540-37432-9. URL: http://dx.doi.org/10.1007/11818175_12.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. "Computationally Private Information Retrieval with Polylogarithmic Communication". English. In: *Advances in Cryptology — EUROCRYPT '99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1999, pp. 402–414. ISBN: 978-3-540-65889-4. URL: http://dx.doi.org/10.1007/3-540-48910-X_28.

[Cod] Codenvy Inc. *Codeenvy*. URL: https://codenvy.com/ (visited on 15/10/2015).

[Cor] Corban Works, LLC. *Fake Name Generator*. URL: http://fakenamegenerator.com/ (visited on 12/23/2015).

[CS14] Melissa Chase and Emily Shen. *Pattern Matching Encryption*. Cryptology ePrint Archive, Report 2014/638. http://eprint.iacr.org/. 2014.

[Cur+06] Reza Curtmola et al. "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS '06. Alexandria, Virginia, USA: ACM, 2006, pp. 79–88. ISBN: 1-59593-518-5. URL: http://doi.acm.org/10.1145/1180405.1180417.

[Dam+03] Ernesto Damiani et al. "Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs". In: *Proceedings of the 10th ACM Conference on Computer and Communications Security*. CCS '03. Washington D.C., USA: ACM, 2003, pp. 93–102. ISBN: 1-58113-738-9. URL: http://doi.acm.org/10.1145/948109.948124.

[Dat] DataNucleus. *DataNucleus AccessPlattform*. URL: http://datanucleus.org/ (visited on 12/26/2015).

[De +10] Sabrina De Capitani di Vimercati et al. "Fragments and loose associations: respecting privacy in data publishing". In: *Proc. VLDB Endow.* 3.1-2 (Sept. 2010), pp. 1370–1381. ISSN: 2150-8097. URL: http://dl.acm.org/citation.cfm?id=1920841.1921009.

[De +12] Sabrina De Capitani di Vimercati et al. "Data Privacy: Definitions and Techniques". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 20.6 (Dec. 2012), pp. 793–817.

[De +13] Sabrina De Capitani di Vimercati et al. "Extending Loose Associations to Multiple Fragments". English. In: *Data and Applications Security and Privacy XXVII*. Ed. by Lingyu Wang and Basit Shafiq. Vol. 7964. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 1–16. ISBN: 978-3-642-39255-9. URL: http://dx.doi.org/10.1007/978-3-642-39256-6_1.

[Dem+13] Denise Demirel et al. "Prêt à Voter Providing Everlasting Privacy". In: *VOTE-ID*. 2013, pp. 156–175.

[DG15] Giovanni Di Crescenzo and Abhrajit Ghosh. "Privacy-Preserving Range Queries from Keyword Queries". English. In: *Data and Applications Security and Privacy XXIX*. Ed. by Pierangela Samarati. Vol. 9149. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 35–50. ISBN: 978-3-319-20809-1. URL: http://dx.doi.org/10.1007/978-3-319-20810-7_3.

[DMN11] Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. "Perfectly Secure Oblivious RAM Without Random Oracles". In: *Proceedings of the 8th Conference on Theory of Cryptography*. TCC'11. Providence, RI: Springer-Verlag, 2011, pp. 144–163. ISBN: 978-3-642-19570-9. URL: http://dl.acm.org/citation.cfm?id=1987260.1987274.

[Dua09] Yitao Duan. "Privacy Without Noise". In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. CIKM '09. Hong Kong, China: ACM, 2009, pp. 1517–1520. ISBN: 978-1-60558-512-3. URL: http://doi.acm.org/10.1145/1645953.1646160.

[Dwo+06a] Cynthia Dwork et al. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *Proceedings of the Third Conference on Theory of Cryptography*. TCC'06. New York, NY: Springer-Verlag, 2006, pp. 265–284. ISBN: 3-540-32731-2, 978-3-540-32731-8. URL: http://dx.doi.org/10.1007/11681878_14.

[Dwo+06b] Cynthia Dwork et al. "Our Data, Ourselves: Privacy Via Distributed Noise Generation". English. In: *Advances in Cryptology - EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 486–503. ISBN: 978-3-540-34546-6. URL: http://dx.doi.org/10.1007/11761679_29.

[Dwo+06c] Cynthia Dwork et al. "Our data, ourselves: privacy via distributed noise generation". In: *Proceedings of the 24th annual international conference on The Theory and Applications of Cryptographic Techniques*. EUROCRYPT'06. St. Petersburg, Russia: Springer-Verlag, 2006, pp. 486–503. ISBN: 3-540-34546-9, 978-3-540-34546-6. URL: http://dx.doi.org/10.1007/11761679_29.

[Dwo08a] Cynthia Dwork. "Differential Privacy: A Survey of Results". In: *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*. TAMC'08. Xi'an, China: Springer-Verlag, 2008, pp. 1–19. ISBN: 3-540-79227-9, 978-3-540-79227-7. URL: http://dl.acm.org/citation.cfm?id=1791834.1791836.

[Dwo08b] Cynthia Dwork. "Differential Privacy: A Survey of Results". English. In: *Theory and Applications of Models of Computation*. Ed. by Manindra Agrawal et al. Vol. 4978. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 1–19. ISBN: 978-3-540-79227-7. URL: http://dx.doi.org/10.1007/978-3-540-79228-4_1.

[EFG10] Sergei Evdokimov, Matthias Fischmann, and Oliver Günther. "Provable Security for Outsourcing Database Operations". In: *Int. J. Inf. Sec. Priv.* 4.1 (Jan. 2010), pp. 1–17. ISSN: 1930-1650. URL: http://dx.doi.org/10.4018/jisp.2010010101.

[EFW10] Alexandre Evfimievski, Ronald Fagin, and David Woodruff. "Epistemic Privacy". In: *J. ACM* 58.1 (Dec. 2010), 2:1–2:45. ISSN: 0004-5411. URL: http://doi.acm.org/10.1145/1870103.1870105.

[Elg85] T. Elgamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *Information Theory, IEEE Transactions on* 31.4 (July 1985), pp. 469–472. ISSN: 0018-9448.

[Geh+12] Johannes Gehrke et al. "Advances in Cryptology – CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings". In: ed. by Reihaneh Safavi-Naini and Ran Canetti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Chap. Crowd-Blending Privacy, pp. 479–496. ISBN: 978-3-642-32009-5. URL: http://dx.doi.org/10.1007/978-3-642-32009-5_28.

[Gen09] Craig Gentry. "A Fully Homomorphic Encryption Scheme". AAI3382729. PhD thesis. Stanford, CA, USA, 2009. ISBN: 978-1-109-44450-6.

[GKS08] Srivatsava Ranjit Ganta, Shiva Prasad Kasiviswanathan, and Adam Smith. "Composition Attacks and Auxiliary Information in Data Privacy". In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '08. Las Vegas, Nevada, USA: ACM, 2008, pp. 265–273. ISBN: 978-1-60558-193-4. URL: http://doi.acm.org/10.1145/1401890.1401926.

[GLP11] Johannes Gehrke, Edward Lui, and Rafael Pass. "Towards Privacy for Social Networks: A Zero-knowledge Based Definition of Privacy". In: *Proceedings of the 8th Conference on Theory of Cryptography*. TCC'11. Providence, RI: Springer-Verlag, 2011, pp. 432–449. ISBN: 978-3-642-19570-9. URL: http://dl.acm.org/citation.cfm?id=1987260.1987294.

[GM82] Shafi Goldwasser and Silvio Micali. "Probabilistic Encryption &Amp; How to Play Mental Poker Keeping Secret All Partial Information". In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. STOC '82. San Francisco, California, USA: ACM, 1982, pp. 365–377. ISBN: 0-89791-070-2. URL: http://doi.acm.org/10.1145/800070.802212.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. "How to Play ANY Mental Game". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: ACM, 1987, pp. 218–229. ISBN: 0-89791-221-7. URL: http://doi.acm.org/10.1145/28395.28420.

[GO96] Oded Goldreich and Rafail Ostrovsky. "Software Protection and Simulation on Oblivious RAMs". In: *J. ACM* 43.3 (May 1996), pp. 431–473. ISSN: 0004-5411. URL: http://doi.acm.org/10.1145/233551.233553.

[Goh03] Eu-Jin Goh. *Secure Indexes*. Cryptology ePrint Archive, Report 2003/216. http://eprint.iacr.org/2003/216/. 2003.

[Goo] Google Inc. *Google Docs*. URL: https://www.google.com/docs/about/ (visited on 15/10/2015).

[Goo+11] Michael T. Goodrich et al. “Oblivious RAM Simulation with Efficient Worst-case Access Overhead”. In: *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*. CCSW ’11. Chicago, Illinois, USA: ACM, 2011, pp. 95–100. ISBN: 978-1-4503-1004-8. URL: http://doi.acm.org/10.1145/2046660.2046680.

[GR05] Craig Gentry and Zulfikar Ramzan. “Single-Database Private Information Retrieval with Constant Communication Rate”. English. In: *Automata, Languages and Programming*. Ed. by Luís Caires et al. Vol. 3580. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 803–815. ISBN: 978-3-540-27580-0. URL: http://dx.doi.org/10.1007/11523468_65.

[Hac+02] Hakan Hacigümüş et al. “Executing SQL over Encrypted Data in the Database-service-provider Model”. In: *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’02. Madison, Wisconsin: ACM, 2002, pp. 216–227. ISBN: 1-58113-497-5. URL: http://doi.acm.org/10.1145/564691.564717.

[Hay+13] R. Haynberg et al. “Symmetric searchable encryption for exact pattern matching using directed Acyclic Word Graphs”. In: *Security and Cryptography (SECRYPT), 2013 International Conference on*. July 2013, pp. 1–8.

[HIM02] Hakan Hacigümüs, Bala Iyer, and Sharad Mehrotra. “Providing Database as a Service”. In: *ICDE ’02: Proceedings of the 18th International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2002, p. 29.

[HK14] Florian Hahn and Florian Kerschbaum. “Searchable Encryption with Secure and Efficient Updates”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’14. Scottsdale, Arizona, USA: ACM, 2014, pp. 310–320. ISBN: 978-1-4503-2957-6. URL: http://doi.acm.org/10.1145/2660267.2660297.

[HMT04] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. “A Privacy-preserving Index for Range Queries”. In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*. VLDB ’04. Toronto, Canada: VLDB Endowment, 2004, pp. 720–731. ISBN: 0-12-088469-0. URL: http://dl.acm.org/citation.cfm?id=1316689.1316752.

[IBM] IBM. *IBM Bluemix DevOps Services*. URL: https://hub.jazz.net/ (visited on 15/10/2015).

[KC05] Murat Kantarcioglu and Chris Clifton. “Security Issues in Querying Encrypted Data”. English. In: *Data and Applications Security XIX*. Ed. by Sushil Jajodia and Duminda Wijesekera. Vol. 3654. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 325–337. ISBN: 978-3-540-28138-2. URL: http://dx.doi.org/10.1007/11535706_24.

[KJ14] Jens Köhler and Konrad Jünemann. “Privacy and Identity Management for Emerging Services and Technologies: 8th IFIP WG 9.2, 9.5, 9.6/11.7, 11.4, 11.6 International Summer School, Nijmegen, The Netherlands, June 17-21, 2013, Revised Selected Papers”. In: ed. by Marit Hansen et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. Chap. Securus: From Confidentiality and

Access Requirements to Data Outsourcing Solutions, pp. 139–149. ISBN: 978-3-642-55137-6. URL: http://dx.doi.org/10.1007/978-3-642-55137-6_11.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007. ISBN: 1584885513.

[KL10] Daniel Kifer and Bing-Rong Lin. "Towards an axiomatization of statistical privacy and utility". In: *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. PODS '10. Indianapolis, Indiana, USA: ACM, 2010, pp. 147–158. ISBN: 978-1-4503-0033-9. URL: http://doi.acm.org/10.1145/1807085.1807106.

[KL12] Daniel Kifer and Bing-Rong Lin. "An Axiomatic View of Statistical Privacy and Utility". In: *Journal of Privacy and Confidentiality: Vol. 4: Iss. 1, Article 2*. 2012. URL: http://repository.cmu.edu/jpc/vol4/iss1/2.

[KM11] Daniel Kifer and Ashwin Machanavajjhala. "No Free Lunch in Data Privacy". In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. SIGMOD '11. Athens, Greece: ACM, 2011, pp. 193–204. ISBN: 978-1-4503-0661-4. URL: http://doi.acm.org/10.1145/1989323.1989345.

[KM12] Daniel Kifer and Ashwin Machanavajjhala. "A Rigorous and Customizable Framework for Privacy". In: *Proceedings of the 31st Symposium on Principles of Database Systems*. PODS '12. Scottsdale, Arizona, USA: ACM, 2012, pp. 77–88. ISBN: 978-1-4503-1248-6. URL: http://doi.acm.org/10.1145/2213556.2213571.

[KM14] Daniel Kifer and Ashwin Machanavajjhala. "Pufferfish: A Framework for Mathematical Privacy Definitions". In: *ACM Trans. Database Syst.* 39.1 (Jan. 2014), 3:1–3:36. ISSN: 0362-5915. URL: http://doi.acm.org/10.1145/2514689.

[KO15] Kaoru Kurosawa and Yasuhiro Ohtaki. *How to Construct UC-Secure Searchable Symmetric Encryption Scheme*. Cryptology ePrint Archive, Report 2015/251. http://eprint.iacr.org/. 2015.

[KO97] E. Kushilevitz and R. Ostrovsky. "Replication is Not Needed: Single Database, Computationally-private Information Retrieval". In: *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*. FOCS '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 364–. ISBN: 0-8186-8197-7. URL: http://dl.acm.org/citation.cfm?id=795663.796363.

[KP13] Seny Kamara and Charalampos Papamanthou. "Parallel and dynamic searchable symmetric encryption". In: *Financial Cryptography and Data Security*. Springer, 2013, pp. 258–274.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. "Dynamic Searchable Symmetric Encryption". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA: ACM, 2012, pp. 965–976. ISBN: 978-1-4503-1651-4. URL: http://doi.acm.org/10.1145/2382196.2382298.

[Lan+09] Lucie Langer et al. “Classifying Privacy and Verifiability Requirements for Electronic Voting”. In: *GI Jahrestagung*. 2009, pp. 1837–1846.

[LDR05] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. “Incognito: Efficient Full-domain K-anonymity”. In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’05. Baltimore, Maryland: ACM, 2005, pp. 49–60. ISBN: 1-59593-060-4. URL: http://doi.acm.org/10.1145/1066157.1066164.

[LK12] Bing-Rong Lin and Daniel Kifer. “A Framework for Extracting Semantic Guarantees from Privacy”. In: *CoRR* abs/1208.5443 (2012). URL: http://arxiv.org/abs/1208.5443.

[LLV07] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. “t-Closeness: Privacy Beyond k-Anonymity and l-Diversity.” In: *ICDE*. Ed. by Rada Chirkova et al. IEEE Computer Society, 2007, pp. 106–115. ISBN: 1-4244-0802-4. URL: http://dblp.uni-trier.de/db/conf/icde/icde2007.html#LiLV07.

[Mac+07] Ashwin Machanavajjhala et al. “L-diversity: Privacy Beyond K-anonymity”. In: *ACM Trans. Knowl. Discov. Data* 1.1 (Mar. 2007). ISSN: 1556-4681. URL: http://doi.acm.org/10.1145/1217299.1217302.

[McS09] Frank D. McSherry. “Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis”. In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’09. Providence, Rhode Island, USA: ACM, 2009, pp. 19–30. ISBN: 978-1-60558-551-2. URL: http://doi.acm.org/10.1145/1559845.1559850.

[Mic] Microsoft Corporation. *Microsoft Office 365*. URL: https://products.office.com (visited on 15/10/2015).

[Mir+09] Ilya Mironov et al. “Advances in Cryptology - CRYPTO 2009: 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings”. In: ed. by Shai Halevi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Chap. Computational Differential Privacy, pp. 126–142. ISBN: 978-3-642-03356-8. URL: http://dx.doi.org/10.1007/978-3-642-03356-8_8.

[MT07] Frank McSherry and Kunal Talwar. “Mechanism Design via Differential Privacy”. In: *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*. FOCS ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 94–103. ISBN: 0-7695-3010-9. URL: http://dx.doi.org/10.1109/FOCS.2007.41.

[MW04] Adam Meyerson and Ryan Williams. “On the Complexity of Optimal K-anonymity”. In: *Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS ’04. Paris, France: ACM, 2004, pp. 223–228. ISBN: 158113858X. URL: http://doi.acm.org/10.1145/1055558.1055591.

[Nao03] Moni Naor. "Advances in Cryptology - CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings". In: ed. by Dan Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. Chap. On Cryptographic Assumptions and Challenges, pp. 96–109. ISBN: 978-3-540-45146-4. URL: http://dx.doi.org/10.1007/978-3-540-45146-4_6.

[NC11] AhmetErhan Nergiz and Chris Clifton. "Query Processing in Private Data Outsourcing Using Anonymization". English. In: *Data and Applications Security and Privacy XXV*. Ed. by Yingjiu Li. Vol. 6818. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 138–153. ISBN: 978-3-642-22347-1. URL: http://dx.doi.org/10.1007/978-3-642-22348-8_12.

[Nig] NightLabs Consulting GmbH. *Cumulus4j*. URL: http://cumulus4j.org/ (visited on 12/23/2015).

[Pap+14] Vasilis Pappas et al. "Blind Seer: A Scalable Private DBMS". In: *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. SP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 359–374. ISBN: 978-1-4799-4686-0. URL: http://dx.doi.org/10.1109/SP.2014.30.

[Pas11] Rafael Pass. "Limits of Provable Security from Standard Assumptions". In: *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*. STOC '11. San Jose, California, USA: ACM, 2011, pp. 109–118. ISBN: 978-1-4503-0691-1. URL: http://doi.acm.org/10.1145/1993636.1993652.

[Pola] PolePosition. *PolePosition Circuits*. URL: http://polepos.sourceforge.net/circuits.html (visited on 05/06/2016).

[Polb] PolePosition. *PolePosition - The Open Source Database Benchmark*. URL: http://polepos.sourceforge.net (visited on 12/26/2015).

[Pop+11] Raluca Ada Popa et al. "CryptDB: Protecting Confidentiality with Encrypted Query Processing". In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. SOSP '11. Cascais, Portugal: ACM, 2011, pp. 85–100. ISBN: 978-1-4503-0977-6. URL: http://doi.acm.org/10.1145/2043556.2043566.

[PR10] Benny Pinkas and Tzachy Reinman. "Oblivious RAM Revisited". In: *Proceedings of the 30th Annual Conference on Advances in Cryptology*. CRYPTO'10. Santa Barbara, CA, USA: Springer-Verlag, 2010, pp. 502–519. ISBN: 3-642-14622-8, 978-3-642-14622-0. URL: http://dl.acm.org/citation.cfm?id=1881412.1881447.

[SC07] Radu Sion and Bogdan Carbunar. "On the Computational Practicality of Private Information Retrieval". In: *In Proceedings of the Network and Distributed Systems Security Symposium, 2007. Stony Brook Network Security and Applied Cryptography Lab Tech Report*. 2007.

[Shi+11] Elaine Shi et al. "Oblivious RAM with O((Logn)3) Worst-case Cost". In: *Proceedings of the 17th International Conference on The Theory and Application of Cryptology and Information Security*. ASIACRYPT'11. Seoul, South Korea: Springer-Verlag, 2011, pp. 197–214. ISBN: 978-3-642-25384-3. URL: http://dx.doi.org/10.1007/978-3-642-25385-0_11.

[SHJ12] Abbas Taheri Soodejani, Mohammad Ali Hadavi, and Rasool Jalili. "k-Anonymity-Based Horizontal Fragmentation to Preserve Privacy in Data Outsourcing." In: *DBSec*. Ed. by Nora Cuppens-Boulahia, Frédéric Cuppens, and Joaquín García-Alfaro. Vol. 7371. Lecture Notes in Computer Science. Springer, 2012, pp. 263–273. ISBN: 978-3-642-31539-8. URL: http://dblp.uni-trier.de/db/conf/dbsec/dbsec2012.html#SoodejaniHJ12.

[SPS13] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. "Practical Dynamic Searchable Encryption with Small Leakage." In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 832.

[SS98] Pierangela Samarati and Latanya Sweeney. *Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression*. Tech. rep. CMU SRI, 1998.

[Swe02] Latanya Sweeney. "K-anonymity: A Model for Protecting Privacy". In: *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 10.5 (Oct. 2002), pp. 557–570. ISSN: 0218-4885. URL: http://dx.doi.org/10.1142/S0218488502001648.

[Tao+10] Yufei Tao et al. "Correlation hiding by independence masking". In: *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*. Mar. 2010, pp. 964–967.

[Ter+12] Manolis Terrovitis et al. "Privacy Preservation by Disassociation". In: *Proc. VLDB Endow.* 5.10 (June 2012), pp. 944–955. ISSN: 2150-8097. URL: http://dx.doi.org/10.14778/2336664.2336668.

[TOB89] C. Turbyfill, C. Orji, and D. Bitton. "AS/sup 3/AP-a comparative relational database benchmark". In: *COMPCON Spring '89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers*. Feb. 1989, pp. 560–564.

[TZ11] Hongwei Tian and Weining Zhang. "Extending L-diversity to Generalize Sensitive Data". In: *Data Knowl. Eng.* 70.1 (Jan. 2011), pp. 101–126. ISSN: 0169-023X. URL: http://dx.doi.org/10.1016/j.datak.2010.09.001.

[Vim+15] Sabrina De Capitani Di Vimercati et al. "Shuffle Index: Efficient and Private Access to Outsourced Data". In: *Trans. Storage* 11.4 (Oct. 2015), 19:1–19:55. ISSN: 1553-3077. URL: http://doi.acm.org/10.1145/2747878.

[VJ10] Marten Van Dijk and Ari Juels. "On the Impossibility of Cryptography Alone for Privacy-preserving Cloud Computing". In: *Proceedings of the 5th USENIX Conference on Hot Topics in Security*. HotSec'10. Washinton, DC: USENIX Association, 2010, pp. 1–8. URL: http://dl.acm.org/citation.cfm?id=1924931.1924934.

[VMw] VMware Inc. *VMWare Desktop*. URL: http://www.vmware.com/cloud-services/desktop/horizon-air-desktop (visited on 15/10/2015).

[War65] Stanley L. Warner. "Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias". In: *Journal of the American Statistical Association* 60.309 (1965), pp. 63–69. URL: http://amstat.tandfonline.com/doi/abs/10.1080/01621459.1965.10480775.

[WB90] Samuel Warren and Louis Brandeis. "The Right to Privacy". In: *Harvard Law Review* (Dec. 1890). URL: http://groups.csail.mit.edu/mac/classes/6.805/articles/privacy/Privacy_brand_warr2.html.

[XT06] Xiaokui Xiao and Yufei Tao. "Anatomy: simple and effective privacy preservation". In: *Proceedings of the 32nd international conference on Very large data bases*. VLDB '06. Seoul, Korea: VLDB Endowment, 2006, pp. 139–150. URL: http://dl.acm.org/citation.cfm?id=1182635.1164141.

[Yao82] Andrew C. Yao. "Protocols for secure computations". In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science* (1982), pp. 160–164. ISSN: 0272-5428.

Acronyms

PPT probabilistic polynomial time (also see: Definition 10 in Chapter 2)

CBC Cipher Block Chaining (also see: Definition 13 in Chapter 2)

JDBC Java Database Connectivity

JPA Java Persistence API

JDO Java Data Objects

AGPL GNU Afero General Public License

SSHFS Secure SHell FileSystem

fuse Filesystem in Userspace

UC Universal Composability

PIR private information retrieval (also see: Definition 46 in Chapter 5)

ORAM Oblivious RAM

SQL Structured Query Language

icp independent column permutation (also see: Definition 52 in Chapter 6)

Appendix

A. Benchmarks

In this section of the appendix, we present details of the benchmark data, queries, and results for the three benchmarks, we used in this work:

- *AS/sup 3/AP* [TOB89], a benchmark, for SQL databases, which we used to benchmark the MimoSecco implementation against a plain PostgreSQL database and against locally mounted and encrypted remote file system with a local database
- PolePosition [Polb], an open source database benchmark, which we used to benchmark DataNucleus with Cumulus4j against Datanucleus without Cumulus4j
- the scaling benchmark, we used to benchmark different optimisations of the MimoSecco implementation

A.1. MimoSecco: *AS/sup 3/AP* Benchmark

A.1.1. Data Sets and Queries

Since the current MimoSecco implementation does not fully support SQL, we chose a subset of the queries of the AS/sup 3/AP Benchmark. We used the SELECT, INSERT, UPDATE, DELETE, ALTER TABLE, and DROP TABLE queries.

For the benchmarks, we used two different data sets of the *AS/sup 3/AP* benchmark, each with 10,000 tuples: the data set *uniques* and the data set *hundreds*. In the data set uniques, all attributes have unique values. In the data set hundreds, most of the attributes have exactly 100 different values. Furthermore, the attribute values are correlated. The data set hundreds data set has a selectiveness of 100. Each data set has 10 attributes comprised of attributes of the types *Int*, *Long*, *Float*, *Date*, and *String* of size 10, 20, and variable size. The queries used for the data set hundreds are depicted in Figure A.1 and the queries used for the data set uniques are depicted in Figure A.2.

Nr.	Query
H1	INSERT INTO hundreds (testint, testlong, testint2, testfloat, testint3, testint4, testdate, teststring10, teststring20, testvariable) VALUES ('2674', '949894990', '191', '-227272727.00', '-858585859', '-858585859', '12/16/1947', 'mE20QkdN38', ':FwWAJf7xCKHGzSTs3zR', 't JNabKQPcNE')
H2	SELECT min(testint) FROM hundreds GROUP BY testint
H3	SELECT min(testlong) FROM hundreds GROUP BY testlong
H4	SELECT min(testint2) FROM hundreds GROUP BY testint2
H5	SELECT min(testfloat) FROM hundreds GROUP BY testfloat
H6	SELECT min(testint3) FROM hundreds GROUP BY testint3
H7	SELECT min(testint4) FROM hundreds GROUP BY testint4
H8	SELECT min(testdate) FROM hundreds GROUP BY testdate
H9	SELECT min(teststring10) FROM hundreds GROUP BY teststring10
H10	SELECT min(teststring20) FROM hundreds GROUP BY teststring20
H11	SELECT min(testvariable) FROM hundreds GROUP BY testvariable
H12	SELECT count(testint) FROM hundreds WHERE testlong <= 1000000 AND testint3 < 99999999 AND testint3 > 1 AND (testfloat < 0 or testfloat > 450000000)
H13	SELECT avg(testint3), min(testint4), max(testint4), max(testdate), min(testdate), count(distinct teststring10), count(teststring10), teststring10, testint FROM hundreds WHERE testfloat < 9800000 GROUP BY teststring10, testint
H14	UPDATE hundreds SET testint = 5000 WHERE testint > 5000
H15	UPDATE hundreds SET testint2 = 5000, teststring10 = 'abcdefghi', testint4 = 49999 WHERE testint = 5000
H16	UPDATE hundreds SET testint = 4000 WHERE testint2 > 5000
H17	UPDATE hundreds SET testint = 3000 WHERE testint3 > 5000
H18	UPDATE hundreds SET testint = 2000 WHERE testint4 > 5000
H19	UPDATE hundreds SET testint = 1000 WHERE testfloat > 5000.0
H20	DELETE FROM hundreds WHERE testint3 < 0
H21	ALTER TABLE hundreds ADD COLUMN land varchar(10)
H22	ALTER TABLE hundreds RENAME COLUMN land TO newland
H23	ALTER TABLE hundreds DROP COLUMN newland
H24	DROP TABLE IF EXISTS hundreds

Figure A.1.: Queries from the *AS/sup 3/AP* benchmark used for the MimoSecco benchmark with the *hundreds* data set.

Figure A.3 depicts mapping of the queries for the data sets hundreds and uniques to the query types SELECT, INSERT, UPDATE, DELETE, ALTER AND DROP.

Nr.	Query
U1	INSERT INTO uniques (testint, testlong, testint2, testfloat, testint3, testint4, testdate,teststring10,teststring20,testvariable) VALUES ('949894990', '949894990', '130163016', '-277777778.00', '-446615527', '-264126413', '12/16/1947', 'mE20QkdN38', 'IJylrEPltCIBhNn4p:dr', '5CZD')
U2	SELECT min(testint) FROM uniques GROUP BY testint
U3	SELECT min(testlong) FROM uniques GROUP BY testlong
U4	SELECT min(testint2) FROM uniques GROUP BY testint2
U5	SELECT min(testfloat) FROM uniques GROUP BY testfloat
U6	SELECT min(testint3) FROM uniques GROUP BY testint3
U7	SELECT min(testint4) FROM uniques GROUP BY testint4
U8	SELECT min(testdate) FROM uniques GROUP BY testdate
U9	SELECT min(teststring10) FROM uniques GROUP BY teststring10
U10	SELECT min(teststring20) FROM uniques GROUP BY teststring20
U11	SELECT min(testvariable) FROM uniques GROUP BY testvariable
U12	SELECT count(testint) FROM uniques WHERE testlong <= 1000000 AND testint3 < 99999999 AND testint3 > 1 AND (testfloat < 0 or testfloat > 450000000)
U13	SELECT avg(testint3), min(testint4), max(testint4), max(testdate), min(testdate), count(distinct teststring10), count(teststring10), teststring10, testint FROM uniques WHERE testfloat < 9800000 GROUP BY teststring10, testint
U14	UPDATE uniques SET testint = 5000 WHERE testint > 5000
U15	UPDATE uniques SET testint2 = 5000, teststring10 = 'abcdefghi', testint4 = 49999 WHERE testint = 5000
U16	UPDATE uniques SET testint = 4000 WHERE testint2 > 5000
U17	UPDATE uniques SET testint = 3000 WHERE testint3 > 5000
U18	UPDATE uniques SET testint = 2000 WHERE testint4 > 5000
U19	UPDATE uniques SET testint = 1000 WHERE testfloat > 5000.0
U20	DELETE FROM uniques WHERE testint3 < 0
U21	ALTER TABLE uniques ADD COLUMN land varchar(10)
U22	ALTER TABLE hundreds RENAME COLUMN land TO newland
U23	ALTER TABLE uniques DROP COLUMN newland
U24	DROP TABLE IF EXISTS uniques

Figure A.2.: Queries from the *AS/sup 3/AP* benchmark used for the MimoSecco benchmark with the *uniques* data set.

A.1.2. Results

Figures A.4 and A.5 show benchmark results for individual queries, Figure A.6 shows averaged benchmark results for query types.

Query Type	Query Numbers
SELECT	H2, H3, H4, H5, H6, H7, H8, H9, H10, H11, H12, H13, U2, U3, U4, U5, U6, U7, U8, U9, U10, U11, U12, U13
INSERT	H1, U1
UPDATE	H14, H15, H16, H17, H18, H19, U14, U15, U16, U17, U18, U19
DELETE	H20, U20
ALTER	H21, H22, H23, U21, U22, U23
DROP	H24, U24

Figure A.3.: Mapping of the queries from Figure A.1 and Figure A.2 to Query types

Query No.	Adaptor	JDBC	SSHFS	Adaptor Optimised
H1	330.0	8.4	5.3	32.0
H2	1428.2	22.6	28.0	265.0
H3	910.0	8.2	9.7	250.0
H4	771.2	3.8	1.0	250.0
H5	807.2	4.0	2.0	265.0
H6	780.2	3.4	1.0	344.0
H7	760.8	3.6	1.3	312.0
H8	768.2	10.4	11.3	281.0
H9	764.4	7.8	8.7	328.0
H10	786.4	12.6	7.0	219.0
H11	793.0	9.2	7.3	203.0
H12	60.8	1.4	1.0	46.0
H13	270.6	15.2	9.0	172.0
H14	4191.0	11.2	64.0	250.0
H15	17300.0	15.6	50.7	550.0
H16	10.4	8.0	4.7	15.0
H17	4774.8	9.0	31.7	234.0
H18	7198.2	18.0	42.0	297.0
H19	5052.6	15.6	29.7	188.0
H20	4306.2	8.6	11.3	312.0
H21	5222.2	8.4	3.0	780.0
H22	17.0	8.0	3.0	48.0
H23	5619.6	8.6	3.0	624.0
H24	218.0	12.4	26.8	94.0

Figure A.4.: Averaged benchmark results in ms for the hundreds data set

Query No.	Adaptor	JDBC	SSHFS	Adaptor Optimised
U1	187.8	8.0	4.0	32.0
U2	856.0	5.4	4.3	172.0
U3	828.2	5.0	2.0	265.0
U4	827.4	4.8	1.3	172.0
U5	827.6	2.4	1.0	174.0
U6	822.8	5.6	1.3	187.0
U7	845.6	5.4	1.3	172.0
U8	848.6	5.4	1.0	172.0
U9	838.0	5.6	1.3	250.0
U10	822.8	6.8	1.3	171.0
U11	829.2	7.0	1.7	312.0
U12	83.6	2.0	0.3	48.0
U13	580.2	21.8	8.0	172.0
U14	9955.2	12.8	63.7	280.0
U15	30665.0	18.0	51.7	858.0
U16	4.2	8.4	4.7	16.0
U17	5372.6	13.8	26.7	141.0
U18	10581.4	18.2	35.0	250.0
U19	771.8	8.0	6.0	47.0
U20	540.0	9.0	6.7	234.0
U21	5630.8	8.0	3.0	655.0
U22	8.0	8.6	2.7	63.0
U23	4957.0	8.4	3.0	718.0
U24	180.8	8.8	26.0	62.0

Figure A.5.: Averaged benchmark results in ms for the uniques data set

Query Type	Adaptor	JDBC	SSHFS	Adaptor Optimised
SELECT	746.3	7.5	4.7	216.8
INSERT	258.9	8.2	4.5	32.0
UPDATE	7989.8	13.1	34.1	260.5
DELETE	2423.1	8.8	9.3	273.0
ALTER	3575.8	8.3	2.9	481.3
DROP	195.8	10.6	26.3	78.0

Figure A.6.: Averaged benchmark results in ms by query type. (cf. Table A.3)

A.2. Cumulus4j: PolePosition

A.2.1. Data Sets and Queries

The PolePosition benchmark suite is organised in so-called *circuits*. These circuits are different benchmark scenarios. They differ in the object data structures and the queries. The idea of these circuits is to replicate different complex application scenarios. PolePosition features the four circles *Complex*, *Flatobject*, *Inheritancehierarchy*, and *Nestetdlists*:

- Complex uses a deep object graph of different classes with an inheritance hierarchy of five levels.

Query Type	Hundreds	Uniques
SELECT	244.6	188.9
INSERT	32.0	32.0
UPDATE	255.7	265.3
DELETE	312.0	234.0
ALTER	484.0	478.7
DROP	94.0	62.0

Figure A.7.: Average benchmark results in ms for the MimoSecco adaptor by data set

- Flatobject uses simple flat objects with indexed fields.
- Inheritancehierarchy operates on objects of a class hierarchy with a depth of five levels.
- Nestetdlists uses a deep graph of lists for traversing.

The goal of PolePosistion is to mimic behaviour of different applications in different scenarios. Therefore, PolePosistion integrates the following operations on its circuits:

- *Write* stores all objects into an initially empty database.
- *Read* loads all attached objects into memory and traverses them by calculating a checksum over all objects.
- *Query* queries for instances over an indexed field.
- *Update* traverses all objects, updates a field in each object.
- *Delete* traverses all objects and deletes each object individually.
- *QueryIndexedString* simulates querying for a number of flat objects by an indexed string member.
- *QueryIndexedInt* simulates querying for a number of flat objects by an indexed int member.

A.2.2. Results

The results for each circuit are depicted in Figure A.8.

Query Type	Datanucleus	Datanucleus with Cumulus4j	Factor
write	9353	33249	3.55
read	20817	3633	0.17
query	8058	1745	0.22
update	21531	19784	0.92
delete	3097	3800	1.23

A.8.1: Complex

Query Type	Datanucleus	Datanucleus with Cumulus4j	Factor
write	52297	1644934	31.5
queryIndexedString	30831	20056	0.7
queryIndexedInt	31156	10328	0.3
update	31513	28920	0.9
delete	30602	25451	0.8

A.8.2: Flatobject

Query Type	Datanucleus	Datanucleus with Cumulus4j	Factor
write	159012	681954	4.3
read	407	41903	103.0
query	6467	252	0.0
delete	88204	462788	5.2

A.8.3: Inheritancehierarchy

Query Type	Datanucleus	Datanucleus with Cumulus4j	Factor
create	192845	1057807	5.5
read	1806679	169003	0.1
update	1476851	827583	0.6
delete	6324391	1153253	0.2

A.8.4: Nestedlists

Figure A.8.: Results of the PolePosition Benchmarks for each circuit in ms

A.3. MimoSecco: Scaling Benchmark

A.3.1. Data Sets and Queries

The data set of the benchmark we used to test the scaling properties of the different index optimisations for the MimoSecco implementation has the four attributes *name*, *surname*, *gender* and *age*. The values of the attribute gender are distributed unevenly with 93% *male* and 7% *female*. The values of the attributes names and surnames are Strings generated with the Fake Name Generator [Cor]. The values of the attribute age is and Integer chosen randomly from $[1, 99]$.

The queries used for this benchmark are depicted in Figure 7.37 in Chapter 7, Section 7.6.2 which we restate in the following in Figure A.9 In order to benchmark the scaling properties of the optimisations, we generated six data sets with 1,000, 2,000, 4,000, 8,000, 16,000, and 32,000 tuples, each.

Query Nr.	Query
S1	SELECT * FROM users WHERE prename = Maximilian
S2	SELECT * FROM users WHERE prename = Maximilian AND gender = male
S3	SELECT * FROM users WHERE name = Moeller AND gender = female

Figure A.9.: The SELECT queries used in the scaling benchmark. The query S1 is a simple query with one condition. In the data set used, 93% of all tuples fulfil the condition *gender = male*.

A.3.2. Results

For each data set and each optimisation and for the scenarios JDBC and SSHBF, the results of the benchmarks are depicted in Figures A.10-A.15. Please refer to Chapter 7, Section 6.1 for a description of the scenarios.

The times depicted in Figure A.10 are the average times needed to insert a single tuple while filling an initially empty database with the corresponding data set.

INSERT	1k	2k	4k	8k	16k	32k
JDBC	0.2	0.2	0.2	0.2	0.2	0.2
SSHFS	0.2	0.2	0.1	0.1	0.2	0.2
Adapter	51.6	54.5	59.2	67.2	91.4	135.6
Hash	56.1	57.7	59.1	62.4	76.8	97.5
Bucket	46.9	42.5	42.0	41.9	50.0	53.5

Figure A.10.: Average execution times in ms of an insert query while filling initially empty databases with each data set depending on the scenario/optimisation

S1	1k	2k	4k	8k	18k	32k
JDBC	2.0	2.0	3.1	5.3	8.0	10.1
SSHFS	0.6	0.5	0.8	1.4	2.5	4.7
Adapter	25.2	26.1	35.3	43.7	55.9	73.7
Hash	26.0	31.5	36.5	40.0	48.4	73.1
Bucket	24.0	26.1	39.5	36.7	48.8	61.7

Figure A.11.: Average execution times in ms of the query S1 depending on the scenario/optimisation and the size of the data set

S2	1k	2k	4k	8k	16k	32k
JDBC	2.0	2.0	3.9	5.9	8.3	10.2
SSHFS	2.0	2.0	2.9	5.0	7.9	9.3
Adapter	35.3	37.7	38.8	73.4	107.3	212.9
Hash	36.6	40.7	40.0	67.3	91.1	199.3
Bucket	34.7	43.1	51.4	81.5	243.2	1354.7

Figure A.12.: Average execution times in ms of the query S2 depending on the scenario/optimisation and the size of the data set

S3	1k	2k	4k	8k	16k	32k
JDBC	2.0	2.0	2.9	5.0	7.9	9.3
SSHFS	0.4	0.5	0.8	1.4	2.5	4.9
Adapter	23.8	25.2	23.0	30.8	38.0	39.5
Hash	30.1	32.8	24.3	31.9	36.3	43.2
Bucket	30.5	30.0	25.4	28.5	38.5	37.5

Figure A.13.: Average execution times in ms of the query S3 depending on the scenario/optimisation and the size of the data set

AVG SELECT	1k	2k	4k	8k	16k	32k
JDBC	2.0	2.0	3.3	5.4	8.1	9.9
SSHFS	0.5	0.5	0.8	1.4	2.5	4.8
Adapter	28.1	29.7	32.4	49.3	67.1	108.7
Hash	30.9	35.0	33.6	46.4	58.6	105.2
Bucket	29.7	33.1	38.8	48.9	110.2	484.6

Figure A.14.: Averaged execution times in ms for the queries S1, S2, and S3 depending on the scenario/optimisation and the size of the data set

AVG ALL	1k	2k	4k	8k	16k	32k
JDBC	1.5	1.5	2.5	4.1	6.1	7.4
SSHFS	0.4	0.4	0.6	1.1	1.9	3.6
Adapter	34.0	35.9	39.1	53.8	73.1	115.4
Hash	37.2	40.7	40.0	50.4	63.2	103.3
Bucket	34.0	35.4	39.6	47.2	95.1	376.8

Figure A.15.: Averaged execution times in ms for the insert queries and for the queries S1, S2, and S3 depending on the scenario/optimisation and the size of the data set