


Technische Universität München
Department of Informatics

Master's Thesis in Informatics

Applicability and Performance Analysis of Encrypted Databases for Smart Environments

Elias Hazboun

Technische Universität München

Department of Informatics

Master's Thesis in Informatics

Applicability and Performance Analysis of Encrypted Databases for Smart Environments

Anwendbarkeit und Performanzanalyse von kryptographischen Datenbanken für Smart Environments

Author	Elias Hazboun
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Dr. Heiko Niedermayer, Dr. Holger Kinkelin, Marcel von Maltitz, M. Sc.
Date	December 15, 2016

Informatik VIII
Chair for Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, December 15, 2016

Signature

Abstract

Advances in sensor and networking technologies have finally brought about the era of the Internet of Things. In the near future, people will be living in smart buildings which are a part of smart campuses and smart cities. These smart environments are encompassed by a growing array of sensors that constantly collect information - which is often strongly person related - and interact with the surrounding environment using an array of actors. However, despite all the research efforts, there still exists a lack of consensus over how to tackle the privacy issues that might arise from living in such environments.

In this thesis, we analyze smart environments for their privacy requirements by taking smart buildings as a concrete example. We then propose the use of encrypted databases as one building block to enable privacy in such environments. Encrypted databases are an emerging technology that allows for the storing and processing over encrypted data without exposing the data to the operator of the database server. Despite named advantages, encrypted databases come with the cost of reduced flexibility and performance. Hence, our main research goal is to assess if such a technology can be applied to the given problem domain. For this purpose, we analyzed various existing encrypted databases and selected ZeroDB as a representative technology and our main research subject. We then conducted performance experiments with the goal of assessing ZeroDB's performance and applicability in our proposed solution.

Our analysis shows that ZeroDB's performance could allow it to be deployed in smart environments, given that certain requirements which affect its performance the most - such as server latency - are carefully managed. However, ZeroDB currently lacks multi-user support and data sharing capabilities, which currently limits its applicability to scenarios with only one stakeholder. Nonetheless, we argue that ZeroDB is a promising technology that - if developed and optimized further - has the potential to succeed and be deployed in privacy preserving smart environments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ubiquitous computing and its derivatives like the Internet of things have been a focus of research since as early as 1987, but as of yet, no real wide-deployment has ever been achieved [1]. The consensus today however, is that advancements in wireless communication and electronic manufacturing have finally reached maturity to the point of enabling such concepts [2]. Therefore, we are witnessing an era of smart environments where sensors, displays and actuators are being embedded in everyday objects and are interconnected together through the Internet to what is commonly known as the cloud.

These smart environments work by constantly collecting data from the surrounding physical environment, processing it or directly transmitting it to other surroundings or remote devices through the underlying network infrastructure. Such environments could be large and relatively public like smart cities or campuses, or could be small and relatively intimate and private like smart homes or cars.

However, the pervasive nature of such environments poses a serious question about their impact on people's lives and data especially in terms of security and privacy. Today more than ever, there is a pressing need to gain more insight on smart environments, their architectures and data handling methods. How can a technology that relies on the continuous yet distributed collection of users' private data succeed without risking the exposure or misuse of such data?

Multiple research efforts have attempted to tackle this problem by employing different approaches like thorough access-control and accounting of centralized data processing and storage systems [3], the use of anonymization techniques [4], and the use of trust based approaches [5]. We believe that the concept of a central data processing and storage entity in a smart environment inherently poses privacy concerns. Nevertheless, a completely distributed approach results in an overhead in terms of cost and complexity among other things. In this thesis, we will look into applying the relatively new technology of encrypted databases to preserve privacy in smart environments that use

the cloud to store data.

1.1 Considered Smart Environments

Smart environments could be divided - based on the number of users they serve and the number of physical areas they span - into three categories as shown in Figure 1.1. Physical areas here represent the semi-autonomous administrative locations inside an environment such as buildings in a campus.

1) Single user, single area

Examples of such environments are simple smart homes and smart cars; they are characterized by having data pertaining usually to a single owner or user at a time and confined in a single physical area. Usually data consists of intimate and private information that is deemed sensitive to the owner such as life habits and locations driven to.

2) Multiple users, single area

Examples of such environments are more complex smart homes or smart buildings and smart streets. This category differs from the first by handling data belonging to multiple users at the same time. Usually data ranges from private information like life habits to less intimate data such as preferred room temperature. This category will be the subject of our research.

3) Multiple users, multiple areas

These are large and complex environments that span multiple physical areas such as smart campuses or even smart cities. They usually handle less private data when compared to previous categories, but nevertheless - considering that users in these environments could be complete strangers to one another - data should be handled carefully.

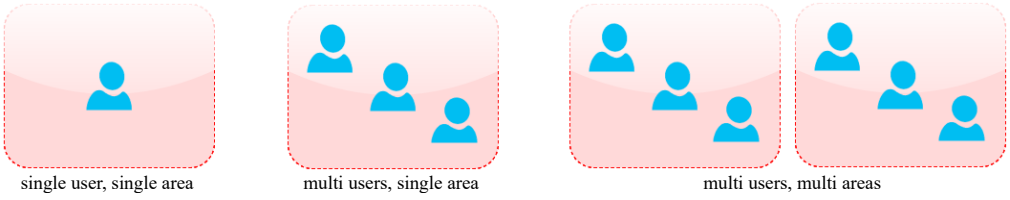

Figure 1.1: Smart environments categorization based on number of users and physical areas

1.2 Encrypted Databases

One of the defining characteristics of ubiquitous computing is cloud computing. A significant number of enterprises today are opting to leverage servers, services and storage facilities in the cloud instead of using locally owned servers. One of the byproducts however of such usage is that enterprises' data is stored and processed on platforms controlled by third parties (cloud hosting companies). This means that private data must be encrypted against these third parties to avoid being exposed. However, encrypted data, due to its nature, cannot be processed before being decrypted first. So a typical data record retrieval from the cloud would involve retrieving a large portion of the data from the cloud, decrypting it on local servers, querying the specific data records required, working on them, and then if needed, storing them back on the server in encrypted form.

Encrypted databases are a relatively new technology that enables applications to not only store encrypted data, but also perform database queries efficiently on them without exposing them to any adversary including the cloud operator. Therefore, enterprises now can utilize encrypted databases on the cloud to skip the initial step of data retrieval and directly perform their needed queries on encrypted data and retrieve the data records required.

As a new technology, encrypted databases are still an active field of research, with no clear consensus over the best method to accomplish them. Currently, research efforts vary significantly in the underlying database technology, the type of queries supported, the encryption technology used, and the level of security guaranteed.

1.3 Research Questions

The goal of this master thesis is to investigate the current technological approaches of encrypted databases and how to best apply them in a smart building scenario. Specifically, we aim to find:

- **Question Q1:** How should a data processing system in a smart building or other smart environments be structured to allow for privacy preserving characteristics?
- **Q2:** What are the limitations of using encrypted databases in smart buildings?
- **Q3:** How fine grained and flexible can access control to encrypted data be performed using encrypted databases?
- **Q4:** What are the factors that affect the performance of encrypted databases and what is their impact?

- **Q5:** What are the performance effects of using encrypted databases compared to non-encrypted ones?

1.4 Methodology

To tackle our research questions, we split our work into three phases.

- **Phase 1:** Study smart buildings and analyzing them from a software engineering and privacy point of view; what are the typical use-cases, users and data in them, what are the requirements needed in a privacy preserving solution for them?
- **Phase 2:** Study encrypted databases, how they work and how they differ from one another and based on it choose one that fits the requirements found in the first phase.
- **Phase 3:** Perform extensive performance analysis tests on that database technology to evaluate it in terms of effectiveness and limitations in smart building scenarios.

1.5 Outline

The rest of this thesis is structured as follows:

Chapter 2 offers background knowledge on smart buildings and encrypted databases, introduces ZeroDB and provides an overview of previous research on the subject of privacy in smart environments.

Chapter 3 provides an in-depth analysis of smart buildings, their software design, typical smart buildings scenarios and the requirements needed to preserve privacy in them. Additionally, it also compares encrypted databases in light of these requirements.

Chapter 4 describes the performance experiments we performed to measure the performance of ZeroDB in smart buildings.

Chapter 5 presents the results we obtained in our experiments. In it we also evaluate the results and conclude some guidelines on encrypted database use in smart buildings.

Chapter 6 serves as the the conclusion of this thesis where we present our contributions to the research questions, discuss the limitations of our study and propose future work.

Chapter 2

Background and Related Work

This chapter provides the background information necessary to better understand the scope of the thesis. It starts by introducing smart buildings and the privacy problems in them, followed by encrypted databases and ZeroDB - an encrypted database example which we will focus on in this thesis. Finally, the chapter ends with a selection of research studies related to tackling the topic of privacy in smart environments and performance analysis of databases.

2.1 Smart Buildings

2.1.1 Building Automation

To better understand smart buildings, we must first understand their origins which lie in the idea of building automation.

Ever since buildings began to incorporate various electronic and electric systems, building management became an increasingly difficult and crucial goal for building owners. When we realize that buildings incur great costs for companies and influence the efficiency of workers inside them, we can comprehend how critical it is for enterprises to control buildings effectively and efficiently.

A typical building today incorporates a vast number of systems and services - most often from different vendors - to perform a variety of goals [6]. These systems are usually independent and controlled through different channels. Moreover, these systems have a varying degree of criticality from fire alarms to lighting controls. Therefore, to manage all of them means that a move to building automation is a must. What we are looking for then is a comprehensive and integrated solution that could efficiently control and automate all of modern buildings' different systems and services.

Analyzing a building automation solution, we can divide it into three main components:

- **Facilities Management:** This includes the typical building services such as lighting, HVAC, access control, security monitoring and safety sensors. This also includes the policies set in place to reduce the running costs of operating a building by efficiently controlling the previously mentioned building services.
- **Information Management:** The controls that allow for the efficient storage of information and more importantly its flow throughout the building. This includes monitoring data, business information and user information.
- **Connectivity:** Buildings must be fitted with all the connectivity tools needed for their systems to function. This not only includes the traditional physical cables but also any modern wireless technologies that might be useful for modern devices.

In building automation research, a usual model to represent the various functions within a building automation solution is a three-level hierarchical model [6]. The three levels from top to bottom are management, automation, and field. The management level is where the majority of operator interface functions reside. Additional functions include communication with controllers, statistical analysis, and centralized energy management functions. Most of the devices at this level are personal computer workstations. The automation level is where the majority of real-time control functions are carried out. The devices tend to be general-purpose, programmable controllers. The field-level contains the devices that connect to sensors and actuators which interact with the physical world. The devices here are usually unitary or application-specific controllers [7].

2.1.2 Smart Buildings

In recent years, building automation became a standard of modern enterprise level buildings. However, not every building that employs a building automation system can be deemed a smart building. For a building to become "smart", it needs: to replace its devices with their "smart device" counterparts, and to include another layer of control which could be seen as the intelligent software that interconnects all of its systems and smart devices together [8]. This intelligence in the software stems from three ideas:

1. It understands the management goals; for example, it can tap into the business schedule to plan the lighting and HVAC settings of offices which in turn can be set based on predetermined user preferred values.
2. It can utilize advanced algorithms to intelligently reduce energy costs of the building without affecting user efficiency and comfort.
3. It provides the management with easy access to control and monitoring interfaces through the web, mobile or any other form needed.

The goals of smart buildings have for the past few years centered around [9]:

1. Cost and energy reduction: The European Union has pledged recently to cut energy consumption by 20% by the year 2020 [10]. When we consider that buildings energy consumption has reached 40% of total consumption in the European Union [11], energy use reduction in buildings becomes a clear candidate for achieving this goal. Moreover, costs of running and maintaining a building are such a large contributor to an enterprise's running expenses that any reduction in them is beneficial.

2. Staff efficiency both for the building administrators and actual tenants. Maintenance employees' or security guards' jobs could become much more efficient by adding the right sensors and actuators which reduce time to resolve problems and help identify issues remotely. Additionally, by analyzing schedules, employees' habits and preferred settings, buildings could intelligently prepare employees' surrounding environments to help them perform their tasks.

3. Users comfort: while this does not directly affect enterprises and building owners running costs, it does increase employees' satisfaction which makes them work more efficiently and ultimately could increase enterprises profits.

2.1.3 Smart Building Protocols

Inspired by the market demand and the goals mentioned previously, different technologies and protocols spawned to perform building automation and more recently control smart buildings and the vast array of smart devices they contain. We will mention here three major such protocols which took different approaches to perform the task at hand, noting that the consensus on the market today is that there is no protocol that holds ultimate advantage over the others; in fact, some buildings employ multiple protocols together.

- The Building Automation and Control Network (BACnet) [12] is a protocol developed by the American Society of Heating, Refrigerating and Air Conditioning Engineers. It was made into a standard in 1995 and has since been continuously updated. It is an object oriented protocol that allows for the communication between and integration of the different systems and devices in a building. BACnet can be used at all levels of building automation, but is particularly suited for management functions.

- LonWorks [13] is a family of protocols for building automation and control networks, where at its core is the LonTalk protocol for transporting messages. LonWorks was developed by the American company Echelon and is used most often at the automation level for the decentralized processing of automation functions.

- Konnex (KNX) [14] is an industry standard for building control communication systems. It was developed by European Installation Bus Association now known as Konnex Association, hence it's dominant in the European market. LonWorks is the main competitor for KNX in Europe, since both have similar roles building automation.

2.1.4 Security and Privacy in Smart Buildings

Unfortunately, as with most early information technology concepts and standards, the protocols we have just mentioned suffer from not paying enough attention to security from the beginning. In other words, they fail to achieve security by design, but regard it as an afterthought [15]. In their early iterations, BACnet and similar protocols did not specify rigorous security controls to the specifications [16]. They decided to leave security as an optional component to be added by building engineers and administrators or applied using other protocols such as TLS or VLANs. In fact, the majority of BACnet devices today do not even support the security features in the standard [16]. To complicate things further, since they were invented, smart buildings took on more privacy critical features and components, which means that security and privacy aspects now have even a greater importance.

The problem becomes even worse when we turn our attention to consumer grade smart devices where these devices are usually not even governed by any smart environment standard and most often include various security bugs in their software. This renders them entry points to people's private spaces and in other cases a part of a botnet army to deploy large DDoS attacks which use insecure IoT devices to amplify their bandwidth [17].

2.2 Encrypted Databases

Encryption is a well-established technology for protecting sensitive data, especially that which is handled or stored by third parties, a recent prominent example of which is cloud storage. Unfortunately, the use of standard encryption schemes for databases in the cloud renders them completely inefficient [18]. This is due to the fact that performing database queries like aggregation or equality, on an encrypted column for example, requires the decryption of the said column first on the client side, since data in standard encrypted form does not reveal any useful information to perform the requested queries.

Therefore, the concept of encrypted databases stems from the need to efficiently perform queries over encrypted data in the cloud without exposing any significant information to third parties or even the cloud service provider [18].

2.2.1 Encryption Schemes Enabling Encrypted Databases

Before we can delve deeper into the different techniques used to achieve encrypted databases, we should first discuss the different encryption schemes that lie at their foundation. In the following we present some of the most prominent of these encryption schemes.

Homomorphic Encryption—First introduced by Rivest et al in 1978 [], homomorphic encryption is a method in which some computation (addition, multiplication) could be done on encrypted data and the result is obtained in encrypted form, thus never needing to expose the original data or the result by decryption.

Any encryption scheme that can only support a subset of operations (usually one) such as addition only or multiplication only is called partial Homomorphic Encryption (PHE) [19]. Some traditional schemes fall into this category; for example, ElGamal and unpadded RSA are PHE schemes supporting multiplication.

Schemes that support arbitrary operations are called Fully Homomorphic encryption (FHE), that is they support any computation to be performed over ciphertexts. Traditionally, FHE was considered only a theoretical scheme. However, in 2009, in his PhD thesis Gentry provided a practical proof of its existence and what is regarded today as a foundation for FHE research [20]. Unfortunately, even though it is achievable, FHE is still very prohibitively slow; almost 105 orders of magnitude slower than normal computations which renders it far from being used today.

Searchable Encryption—This schemee supports the ability to perform a word search over encrypted pieces of text without exposing the original text or the words being searched for. Most efficient approaches to achieve this involves creating some encrypted index of keywords and storing them alongside the encrypted documents [19]. Most searchable encryption schemes rely on client – server model, hence some research focus in this area has been around the interactions of having multiple clients storing encrypted documents in a shared store and allowing for the searching and sharing of documents.

Deterministic Encryption—This scheme is a one that outputs the same ciphertext given the same pair of plaintext and key no matter how many times the encryption is repeated [21]. Deterministic encryption can be used to achieve efficient searchable encryption among other things such as avoiding the storage of duplicates of encrypted files in the cloud [21]. However, from a security point of view, its strength is usually weaker than probabilistic encryption [22].

Functional Encryption—In this scheme, an authority that holds a master key can generate a key to compute a function over encrypted data. That is, functional encryption allows for a client to learn the output of a function of the ciphertext, but nothing about the actual plaintext data [23].

Property Preserving Encryption—The scheme allows a group of ciphertexts to be tested whether or not they satisfy a certain property [24]. This scheme could be seen as a generalization of order preserving encryption (not discussed here) where the property tested is "ordering of elements", or keyword searchable encryption where the property is "equality of keywords".

The previously discussed encryption schemes vary between each other in three trade-off categories: (1) efficiency, (2) security, and (3) query expressiveness [25]. These categories ultimately decide which area an encryption scheme is best suited for and how it used in encrypted databases. In addition, these encryption schemes cannot be individually seen in a vacuum since they are all related and in some cases they could be regarded as special cases or generalizations of one another. Finally, it is also possible to leverage a combination of these schemes to work together and create a more robust and well-rounded scheme.

2.2.2 Categories of Encrypted Databases

Now that we have reviewed the different schemes of encrypting data in the cloud, we look into a few practical examples of achieving encrypted databases. There are three major approaches for achieving encrypted databases [26]. We discuss them below.

Encryption in the cloud—This approach involves using one or a combination of the encryption schemes we have previously discussed. The most prominent example of this category is CryptDB [27] which uses a combination of encryption schemes to create layers of ciphertexts known as "onion-encryption" based on the query to be performed.

Trusted hardware—This approach utilizes trusted hardware modules usually on the server side to perform the parts of the queries which are deemed too sensitive to expose to the cloud operator [26].

Client Centric—This approach usually relies on performing as much computation as possible on the server side, and then relegating the parts that are confidential to the client side to perform [18]. One example of this category is ZeroDB [28] which is discussed in the next section.

2.2.3 ZeroDB

In this thesis, we take ZeroDB as a representative example of client centric encrypted databases in order to assess their performance. Therefore, we deem it necessary to include an introduction on ZeroDB at this stage of the thesis.

ZeroDB is an open-source encrypted database project from a startup called by the same name. It is built upon ZODB which is a non-encrypted object-oriented database providing persistence storage for python objects. The threat model of ZeroDB assumes a trusted client, a hostile network and an honest but curious server or cloud operator [28]. ZeroDB belongs to the family of encrypted databases that puts the trusted client as their approach to security; its main concept is its query protocol where the client is responsible for traversing and maintaining an encrypted B-Tree index on the server side.

The server in ZeroDB stores an encrypted index as a B-Tree where the nodes are buckets and the leaf nodes point to actual objects stored. Therefore, to store data on the server, the client needs to first encrypt the object, send it to the server and update the tree index to accommodate the new object. Throughout the process the key material never leaves the client and hence the server cannot determine any information regarding the plaintext of the object or the overall structure and contents of the index tree.

Let us assume that the client wishes to get all records where an attribute "temperature" is equal to 23. The client requests the root node of the index tree of that attribute from the server which in turn sends it back, the client then proceeds to decrypt it to figure out which of its branches could contain the value 23, then requests from the server to return the root of that branch. For the server, this request involves a random encrypted object in memory and cannot infer any more information other than the depth of that object in the index tree. This step repeats, with each roundtrip of messages allowing the client to reach a lower depth in the tree until it either finds the objects or none. An example of this query can be seen in Figure 2.1

In terms of query expressiveness, ZeroDB supports traditional SQL queries like equality, range, greater and smaller than, etc. Furthermore, it supports search over encrypted text which is performed by ordering results by a net rank. Net ranks are determined by the text's relevance to the word to be searched and include the use of additively homomorphic encryption.

ZeroDB hence is agnostic to the encryption algorithm used, since the query processing does not occur over encrypted text on the server, but is rather done by the client which traverses the remote index. In the actual implementation of ZeroDB, the developers use AES256 in GCM mode as their underlying encryption scheme. Additionally, they use SSL protocol to secure the traffic over the network and perform authentication of client and server.

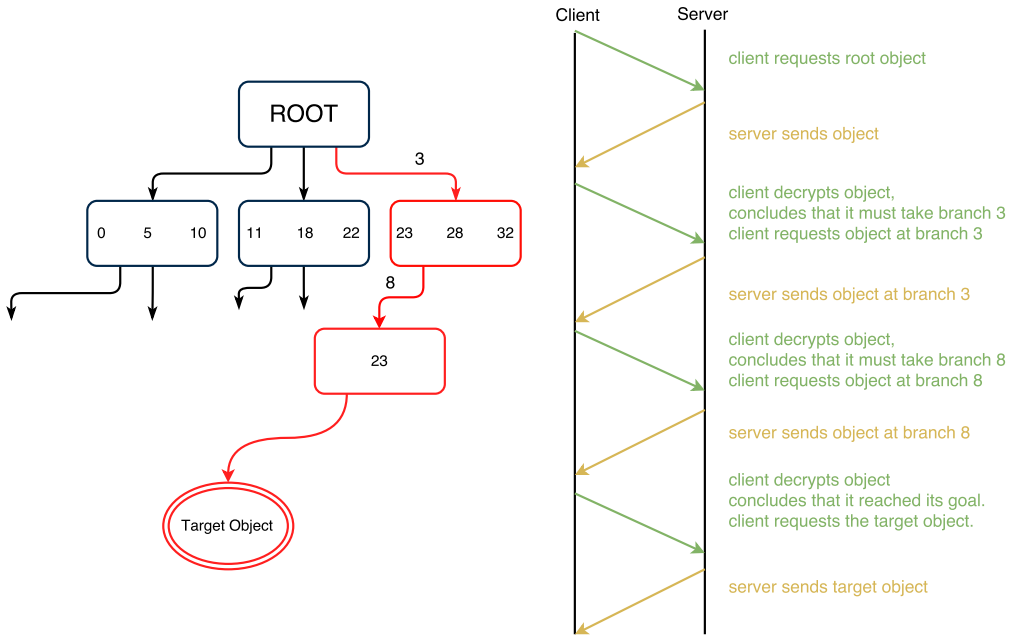

Figure 2.1: An example query sequence where the client remotely traverses through the index tree to find an object with value 23.

ZeroDB's approach to encrypted databases creates a large number of message exchanges and traffic to be sent over the connection between the server and client. As a result, the developers added some optimizations to mitigate the problems of slow connections. The optimizations involve bulk-fetching small subtrees where if a subtree is small enough, the client would simply decrypt the root node and fetch all of its children. The client repeats this process for all the subtree. Additionally, another optimization is to perform parallel tree traversal where multiple requests are sent simultaneously.

For sharing data between users of the same database, ZeroDB uses of Proxy re-encryption which in simple terms allows for the server to act as a proxy and alter the encrypted text of one user to be encrypted in a way that can be decrypted by another user's secret key. This method requires the user to send to the server the re-encryption key and an optional revocation time.

2.3 Related Work

There is a considerable amount of research on the topic of privacy and security in smart environments. However, so far there is no consensus on how to best tackle this issue especially in terms of privacy. Research efforts varied in their proposed solutions but the majority of earlier works shared the same threat model and goal. Namely, they tackled first and foremost the issue of outside actors gaining access to private data

stored and collected by the smart environment. Only recently that a number of papers investigated the issue of malicious or simply curious insider actors accessing private data or exploiting their access to spy on users' private information.

In their paper [29], Chen et al. designed a context broker for Smart meeting rooms. To protect users' privacy in it, they created a policy language for users to define their own privacy rules. The language allows users take direct control over who gets access to their data. Moreover, their engine that infers access right based on these policies exploits the concept of locality. For example, if an actor requests the location of a user from the system the system will respond with the location only if it is within a value that could be defined by the user.

Armac et al. [30] on the other hand took a different approach to protect the privacy of users in smart homes. Their approach involves the use of an identity management system where users have different identities which - based on the smart environment - they could select to present themselves as to a smart home. Moreover, they use the concept of anonymous credentials which benefits their system by minimizing traceability and linkability of users' identities between smart homes.

Finally, Neisse et al. [31] created a Model-based security Toolkit named SecKit to tackle privacy and security in smart environments. An interesting inclusion in their toolkit is the support for trust management. In it, trust relationships could be precisely defined between users and smart devices. For example, a user trusts device A to collect their location information but does not trust it to recommend directions to the nearest hospital in a city. This is a valid concept that we believe could be used in conjunction with other systems that tackles privacy.

Chapter 3

Analysis

This chapter focuses on analyzing smart buildings in terms of software design in its first half and current encrypted database research in its second half. An in-depth analysis of typical scenarios and the data flowing in smart buildings is presented, followed by an analysis of how to ensure proper privacy of data in smart buildings. Moreover, a comparison of encrypted database offerings is performed, and we conclude the chapter with an analysis of ZeroDB as an example encrypted database.

3.1 Smart Buildings

To properly assess the software design properties of smart buildings, we must first define what a smart building is. We envision a smart building as a multi-floor building owned and maintained by a single company which either occupies it or rents it to multiple tenants. In the case where the building is rented, most of the tenants are companies renting offices, meeting rooms, server rooms, and communal rooms. Some offices and parts of the building are dynamically rented and occupied; they don not have a fixed occupant during the day but can be booked on demand if they are empty.

The building draws its energy from the smart electrical grid but also has solar panels on the roof for local electricity generation. It is also connected to hot and cold water supply, and internally has an extra water tank for storing and heating water locally as necessary. Additionally, a sophisticated system of heating, ventilation and air-conditioning (HVAC) is installed throughout the building.

In terms of information technology infrastructure, the building has an extensive computer network cabling reaching all rooms and locations. In addition, cameras, smoke detectors and other safety and security sensors are installed throughout the building.

Most physical objects in the building are considered smart devices; they are fitted with sensors, actuators and displays to enhance their functionality. For example, doors,

windows, blinds and electronic equipment can be controlled automatically or remotely. Moreover, their energy consumption is monitored on a per device and room basis.

3.1.1 Software Architecture Design

Sensors and actuators in smart devices usually collect and transmit data to interact with the physical world around them. These smart devices most often belong to a single physical location like an office, but send data to and receive commands from a central server residing in the building. We believe this centralization approach is an inherent source of risk to users' privacy in smart buildings. A malicious attacker that gains access to the central server can achieve complete control over the entire network of smart devices in the building rendering the whole building and employees' data exposed. Moreover, an honest but curious building operator could exploit their "big brother" view of the building to intrude on employees' privacy. Even further, when we analyze the functions of smart devices, it is clear that a majority of their goals can be achieved without the need of such central entity to coordinate their work.

Therefore, it could be theorized that smart devices within a single location could be regarded as part of an autonomous region – a locality, thus confining most of the users' data within said locality. In this solution, an attacker needs to gain physical access to each locality to achieve complete coverage. However, this complete decentralized approach results in a dramatic increase in complexity and costs due to the installation of servers within each locality to govern smart devices and store their data. If we assume each office or confined space to be a locality, then we have a cost that no building operator would be willing to pay.

Hence, we propose an architecture that could be seen as a hybrid of the two approaches we have mentioned. Its basis lies within the following three concepts:

1. Most of the data within a locality should remain inside it, and only a selected portion is needed to be shared with other localities or with central administration.

2. Centralization of processing and storage is essential to reduce complexity and costs, and keep an administrative oversight over the building.

3. Building administrators' access to users' and devices' data should be limited to the proportion needed for them to perform their administrative duties. For example, a curious administrator should not have access to the whereabouts of a user all day long.

One way to reconcile these three points is to secure each locality's data on the central server in a manner such that even though the data is stored on one server, only the locality owning the data will actually be able to access it and share it when necessary.

Using encrypted databases on a central server allows us to achieve this concept. If we assume each locality is a user in the database, we can centralize the data on one database while protecting it from unauthorized access. Figure 3.1 illustrates a high level overview of smart building using encrypted databases.

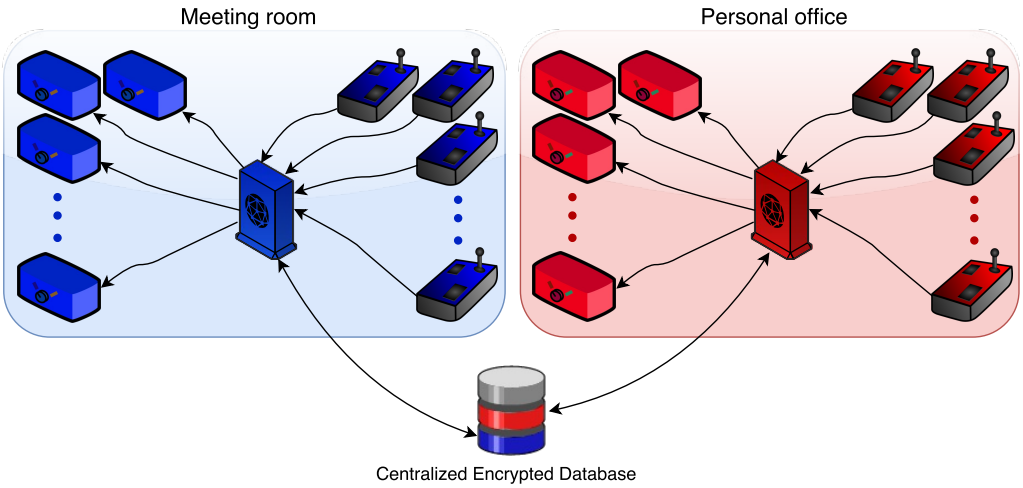

Figure 3.1: Two adjacent rooms with small computing devices in their center which are connected to an array of sensors and actuators. Each room is also connected to a central encrypted database.

The big light red and light blue rectangles represent two localities e.g. a meeting room and an office inside a building. They both contain a set of sensors and actuators that are attached to smart devices. In the center of each locality is a logic unit - a small computing device - capable of communicating with the sensors and actuators, and also with with the centralized database at the bottom of the figure. An actual implementation of such device could be a raspberry pi []. Each logic unit is a unique client on the central encrypted database whose data can be decrypted by a key known only to said the client.

The advantage of including a logic unit in each locality is twofold. One, it acts as a coordinator between the various smart devices within each locality, and also between two localities that want to communicate with each other. Second, it acts as a client to the encrypted database sever on behalf of all the sensors and actuators in the locality. This is of great importance since usually these devices are low powered and lack the computing capabilities necessary to communicate with an encrypted database. Moreover, this also removes the need to change any software on the smart devices installed in the building.

3.1.2 Typical Scenarios

In order to get a better understanding of the requirements needed in an encrypted database to function properly in a smart building, we need to analyze the typical scenarios and use cases in which it will be utilized. In this section we present a number of

selected scenarios we deemed representative. Moreover, in appendix A, we also present use-cases derived from these scenarios.

Typical Scenarios:

1. Employee books a room on demand

   Description: An employee swipes his badge by sensor near the door of an empty office, thus opening the door and by using either the app or a small screen mounted on the office wall, books the room for the next 6 hours.

   Analysis:

   - Sensor reads user badge, sends data to logic unit pertaining to company (if the room is rented by one tenant) or to a logical unit belonging to the building (if the room is shared by multiple tenants).
   - Logic unit returns back ACK/NACK.
   - Command to actuator is sent to open the door.
   - User preferences from database belonging to company are fetched.
   - HVAC and lighting systems are informed of user preferences.
   - HVAC and lighting systems calculate, using current room and neighboring rooms' sensor data, the best possible temperatures.

2. Energy consumption

   Description: A company has a fixed amount of electricity consumed per month, if exceeded, cost of electricity goes up substantially. Moreover, the electric grid periodically informs building electric system of cost changes. End users are informed of this when they try to use a non-work critical device like a kitchen device.

   Analysis:

   - Electric grid informs the building system of an increase in the cost in the next two hours due to high demand.
   - An employee in that period turns on an electric device..
   - The logic unit in the room is informed of the matter, and fetches historical data of energy consumption of device when this specific employee uses it.
   - The logic unit calculates that it will cost money more than what is expected for an employee per day.
   - The logic unit sends command to device to warn the employee of this.
   - If the employee agrees, device continues to work, otherwise it is stopped.

3. Energy consumption optimization

   Description: The system is able to precisely optimize the parameter states of multiple adjacent rooms in terms of HVAC, lighting etc. to save energy consumption

without greatly interfering with users' optimal settings and preferences.

Analysis:

- A change is triggered in the parameters of a room's HVAC, lighting etc.
- The central logic unit of that area (floor, department) calculates how this change can be achieved within the context of the surrounding rooms (or the rooms communicate between each other to determine this).
- Each room affected can change its optimal parameters slightly to accommodate for the overall change where the overall energy consumption is minimized while user comfort remains intact.
- For example, before the change trigger, Room 1 had the AC: 21°, Room 2: 23° and Room 3: 20°. When Room 4 is occupied and a user wants it to be 25°. The end result could become: Room 1: 21.5°, Room 2: 23.5°, Room 3: 20.8° and Room 4: 24.2°.

4. Fire Emergency

   Description: Smoke detectors, detect an increase in CO_2 levels in a room on some floor and initiate fire emergency case.

   Analysis:

   - Smoke sensors detect an increase in CO_2.
   - Logic unit inside the room calculates that this amount is high enough to indicate a fire.
   - Logic unit inside the room contacts the neighboring rooms to inform them of this and to query their CO_2 sensor readings to check if the fire is spread.
   - Logic unit also informs the central building emergency system of the fire in the room and possibly the readings of neighboring rooms.
   - Logic unit disables HVAC systems and if the room occupancy is empty, it will close all windows and doors to prevent the spread of fire.
   - Central logic unit now initiates building wide emergency protocol (including shutting down empty elevators and fast evacuation of used ones).

5. Periodical calculation of energy and water consumption

   Description: For saving purposes, periodical calculation of consumption per floor, per room, per building, per tenant is done to better understand it and try to minimize it as appropriate.

   Analysis:

   - Energy consumption is logged in the logic unit of a room at the end of each day.
   - The same is done for a floor, and for a tenant.

- At the end of the month, data is carefully shared to calculate the consumption.
- Consumption analysis is shared with room occupants, tenants and building owners and operators as appropriate.

6. Room occupancy

   Description: sensors at room doors calculate number of occupants in each room and their identities. This data can be used in a variety of ways:

   - Calculate the best HVAC and lighting settings to suit most of the employees in a meeting room. For example, lowering the light intensity in the room or increasing the fan speed.
   - Calculate the number of employees in the building at any point to better adjust cost savings. For example, based on their location, elevators in almost empty sections can be turned off.
   - In case of a grievous incident, the system can inform authorities of the whereabouts of an employee or a room occupant at a specific date in the past (A specific set of pre-defined and agreed data can only be accessed).

3.1.3 Users and Data in Smart Buildings

Analyzing the previous scenarios, we can divide the users of a smart building into the following groups:

1. Employees.
2. Guests.
3. Building Operators.
4. Third party service personnel.
5. Intruders (anyone not belonging to any of the previous categories).

In a similar vein, we can divide the data flowing through the smart building into different categories, which can help us in understanding the privacy requirements needed for each of them. The categories are:

1. Basic Data

   a Sensor Data (pertaining to a room).
   b Sensor Data (pertaining to a user).
   c Arbitrary Data (File, announcement, software update, etc.)
   d Maintenance Data (Device and system status, reporting of system intercommunication).
   e Commands to an Actuator.
   f Preferences of a User.

   g Emergency data (usually commands).

2. Processed Data

   a Aggregated Data which usually decreases privacy criticality.
   b Enrichment of data by complex functions which usually increases privacy criticality.

Finally, we can also summarize how data is flowing inside a smart building:

1. Between a sensor or an actuator and a logic unit and vice-versa.
2. Between physically close localities
3. Between physically separated localities.
4. Between a locality and the server.
5. Between the building and an outside entity or entities.

3.1.4 Privacy Requirements

From the previous analysis we can infer a set of requirements that must be in a technology to be considered as a viable candidate to power the end solution. These are discussed below:

- **Requirement R1: Strong encryption** - Data must be stored using a strong encryption scheme with keys known only by the data owner that even the administrator of the system cannot access arbitrary data in plaintext.
- **R2: Multi-user support** - The system must be able to handle multiple users or tenants and appropriately handle their data.
- **R3: Secure Sharing** - The Owner of the data might be willing to grant access rights to its data to the system, a user in the system or a third party. By doing so, the owner will not risk exposing the plaintext to any party other than the intended, even the database itself.
- **R4: Timed or revocable access rights** - Access rights to data elements can be granted temporarily or revoked on demand.
- **R5: Non-trust based** - No trust in an entity should be needed to operate the system securely. All system guarantees must be made solely on the basis of mathematical proofs of security.
- **R6: Secure, efficient aggregation** - Database queries and aggregation functions like sum, average etc. should be achievable by the system in a reasonable time on encrypted data belonging to multiple users.

3.2 Encrypted Databases

Having analyzed the software architecture of smart buildings using encrypted databases and summarized the requirements needed in an encrypted database to make it usable in such an environment, we now offer a survey of the available research on encrypted databases and compare their viability for our smart building architecture.

We analyzed numerous research projects that dealt with encrypted databases. They ranged in their purpose and functionality from completely theoretical studies to commercial products. In the following we offer a brief overview of the most prominent encrypted databases we analyzed.

CryptDB

Published in 2011, CryptDB [27] is one of the most prominent research projects on the topic. It is often cited by other authors and used as a comparison reference. It relies on SQL aware encryption schemes to protect the data. CryptDB addresses two types of attackers, a curious database administrator and an attacker who gains access to the database server. For the latter type, CryptDB doesn't guarantee the confidentiality for users already logged in the system, but ensures it only for logged-out users.

CrypDB uses three concepts to accomplish its goals. First, it uses well defined SQL queries that rely on equality checks, aggregates, and joins which in turn can be used to adapt encryption schemes to work on encrypted data. Second, it uses onions of encryption to store multiple ciphertexts within each other, each layer of encryption uses an encryption scheme designed specifically to accommodate a different type of query. Finally, it chains encryption keys to user passwords, so that even if the database is compromised, the attacker must know the user's password to gain access to their data [27].

Arx

Arx [32] is a encrypted database partly developed by the same authors of CryptDB. It uses AES to guarantee strong encryption properties. However, since AES cannot be computed upon, the authors introduced two database indices (for equality and range queries) which are built on top of AES. In addition, it uses one-time obfuscation in the index tree, which means that partial index rebuilding must be done after each query.

The architecture of Arx involves a client proxy and a server proxy which act between the unmodified standard client and database serve. The client proxy handles sensitive data and has access to encryption keys, whereas the server proxy is part of the server

which is under attack. The database server itself could be of any type, however Arx authors implemented it on top of MongoDB a NoSQL database.

SEEED

Search over Encrypted Data (SEEED) [33] is a framework developed by the company SAP for tackling the problem of encrypted databases. SEEED relies on a modified version of the onion encryption concept developed in CryptDB. A SEEED database driver was developed to translate and encrypt SQL statements to be sent to the server. As for the threat model, it comprises a trusted client, an untrusted network and an honest but curious server operator.

TrustedDB

TrustedDB [34] is a trusted hardware based relational database. Its developers argue that using secure server-side hardware like the IBM cryptographic co-processors for secure database processing is highly efficient than using common hardware. Therefore, TrustedDB runs two database management servers, a modified SQLite server on the secured hardware, and a normal MySQL server on the normal hardware.

Cipherbase

Cipherbase [35] is the result of work at Microsoft Research where they tried to leverage in-server trusted hardware to run a small part of the encrypted database that deals with confidential data. They boast that by limiting the use of trusted hardware to a few thousand lines of Verilog code they achieve higher performance compared to databases that run a larger part of the database in trusted hardware.

In terms of functionality, Cipherbase supports most of SQL database features and even some of it without any change in the underlying original code. Finally, for secure hardware, the authors chose a PCIe-based FPGA board to run their critical sections of code.

MuteDB

Multi-User relaTional Encrypted Databse (MuteDB) [36] is an architecture for secure SQL operations over encrypted data in the cloud. It is unique among the rest in that its threat model involves having legitimate users colluding with cloud operators to compromise other user's stored data. It relies on key management and access control policy enforcement to guarantee data confidentiality.

ZeroDB

We have already introduced in the previous chapter ZeroDB; an encrypted database that relies on the client to do most of the heavy-lifting in terms of computation to secure data on the cloud. However, when we delved deeper into its most recent implementation which is published on GitHub, we noticed a few discrepancies between what is claimed in the white paper which we based our introduction upon and what is actually implemented.

The first major discrepancy is multi-user management support, even though we see its seeds in the code, in its current shape it is not fully supported and it is recommended to only use one user per database. The second discrepancy lies in the possibility of data sharing between users. In the paper, they mention two different possible methods to achieve this: proxy re-encryption and delta-keys, none of which is currently implemented. In the published code, some proxy re-encryption tests are visible in the published code. When asked about it, the developers pointed out that it possible to perform proxy re-encryption in ZeroDB, but they don't have actual implementation developed for it yet. Finally, since data sharing is not possible, timed or revocable access control to shared data is consequently not possible as well.

3.2.1 A Comparison of Encrypted Databases

Given the requirements that we have compiled for a scheme to work in a smart building, we now offer a comparison of the different databases we have researched in terms of these requirements. In table 3.1, we show our results where we included a column to address the ZeroDB discrepancies we found. Additionally, we also added a column which states whether or not a publicly available API is there for that database, since without one designed for third parties to use, it would be too difficult for us to utilize it in our analysis tests.

From the table, it is clear that ZeroDB, although misses a few requirements because of its current limitations, is still the right candidate for us since it not only has an API available for us to use, it is also in active development. Moreover, the discrepancies can be ultimately reconciled in the future so that it fully supports most of our requirements.

Nevertheless, we see that moving forward we no longer can perform a complete prototype of our proposed solution of a smart building using an encrypted database (ZeroDB). Therefore, in the next two chapters, we focus instead on planning, performing and evaluating the results of performance analysis experiments that are designed to assess the performance, applicability and efficiency of ZeroDB in a smart building.

Table 3.1: Comparison of encrypted databases in terms of privacy requirements

	ZeroDB	ZeroDB*	CryptDB	Arx	Cipherbase	SEEED	MuteDB	TrustedDB
Strong Encryption	++	++	+	++	++	++	+++	+*
Multi-user Support	++	-	++	-	+**	++	++	-
Secure Sharing	+	-	+	-	-	-	+	-
Timed or Revocable	+	-	?	-	-	-	++	-
Secure, efficient aggregation	-	-	++	++	+	++	++	++
API available	++	++	+	-	-	-	+	-

++ strongly meets the requirement, + partially meets the requirement

- does not meet the requirement, ? no data

* requires trust. ** potentially possible.

Chapter 4

Performance Evaluation Methodology

In this chapter, we describe the performance experiments we performed to measure the performance of ZeroDB in smart buildings. We start by detailing our methodology including the experiment setup and the databases with which we chose to compare our results with, followed by listing each experiment, what it measures, how it is performed and what its goals are.

4.1 Experiment methodology

In order to understand the overall performance and applicability of ZeroDB in a smart building, we chose to perform experiments that assess its performance given various operating conditions. We analyzed the different parameters that might affect the overall performance of ZeroDB, and for each of them designed an experiment that measures that parameter's effect on the performance. These parameters are: bandwidth and latency of the link, size and number of records in the database, distribution of data in the database, query result size and the use of multiple-condition queries.

4.1.1 Compared Databases

To assess ZeroDB's performance, it does not suffice to test it in various conditions, we also need to test other databases in the same conditions and compare its performance to theirs, which will result in better conclusions on ZeroDB.

The first database we will compare ZeroDB to is MySQL [37]. MySQL is a prominent open-source relational database. Its performance, reliability and features made it one of the most used database in the world, powering projects from the US Navy to Uber and Youtube [38]. We chose MySQL as the first comparison database, because it represents an industry standard and will most likely represent the expected performance out of

relational databases. We expect of MySQL to especially outperform ZeroDB when querying large amounts of data or when the latency between the client and the server is large.

The second database is ZODB, which is an open-source object-oriented database [39]. In a sense, it is a persistent storage facility for objects in python. The reason we choose ZODB as the second comparison database is that ZeroDB is based on it, so by comparing their performances, we can measure two things: the performance penalty of the algorithm used by ZeroDB to achieve the security and privacy of an encrypted database, and also the effectiveness – if any – of the optimization algorithms inserted by ZeroDB into ZODB. However, we do not have sufficient historical or background information to quantify how the performance of ZODB will measure compared to ZeroDB.

4.1.2 Experiment Setup

To perform the various experiments, we setup a lab environment where two computers are setup to act as a server and a client. The computers were desktop computers with specifications listed in Table 4.1:

Table 4.1: Hardware specifications

Processor	Intel Xeon CPU E3-1275L v3
Memory	Kingston 8 GB, DDR3-1600
Secondary Storage	Crucial SSD BX100, 120 GB
Network Card	Intel ET I340 Server Adapter

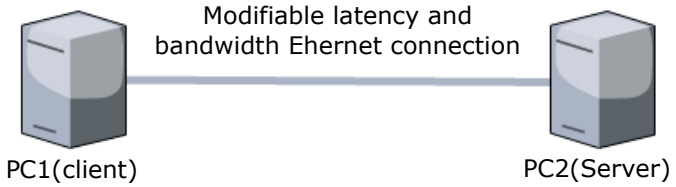

Figure 4.1: Setup of experiment environment

In the setup as seen in Figure 4.1, one computer, namely PC1, acts as the server holding the database, while PC2 acts as a client querying records from the database. The two computers are connected directly by an Ethernet cable providing 1000 Mbps bandwidth. Debian Jessie is the operating system installed on both computers in addition to all the required libraries and packages used in the experiments.

For a more in depth procedure on how the software is setup for each of the three different databases, please refer to Appendix B.

4.1.3 Experiment Flow

The premise of all the experiments is as follows: start network traffic capture on the link between client and server, instruct the client to send an equality SELECT query (or its equivalent for ZeroDB and ZODB) to the server, the server responds back with the query results, stop the traffic capture. This process is repeated a certain number of times to increase the statistical significance of its results and try to remove the effect of any irregularities that might exist. Unless otherwise noted, the number of repetitions for each experiment is set to 10.

To perform this, we wrote multiple Bash scripts that together orchestrate a given experiment from the setup of the database server and the generation of records, to the execution of the actual queries and recording them in capture files.

An experiment starts by running a script called `automate.sh` and providing it with the name of the file that contains the parameters for the experiment. The most important parameters recognized by the script are:

- `Repetitions:` number of times the experiment will be repeated.
- `Technology:` Database to be used (ZeroDB, MySQL, ZODB).
- `Record:` name of object (for ZeroDB and ZODB) or table (for MySQL) to be queried.
- `Dbsize:` number of records stored in the database.
- `Recordsize:` size of added bytes to the record size.
- `Bandwidth:` bandwidth in Mbps on the client side.
- `Latency:` latency in milliseconds between client and server.
- `Querysize:` number of records queried.
- `Distribution:` distribution of the randomized values of stored records.

After reading the parameters, `automate.sh` script repeatedly calls `analyze.sh` (according to the number of repetitions) before finally calling another script to create the relevant results graphs.

The script `analyze.sh` is the main script that performs the experiment itself. Its basic flow is described in Figure 4.2 which we can summarize in the following points:

1. Read the parameters.
2. Setup and run a server instance of the needed database.
3. Populate the database with randomized data generated according to the required distribution.
4. Configure the latency and the bandwidth of the link as required.
5. Start traffic capture on the server.

6. Run the client program to query the data from the server.
7. End traffic capture once the client is done.
8. Analyze the capture file and save the important values to an XML file.
9. Shutdown the server instance.

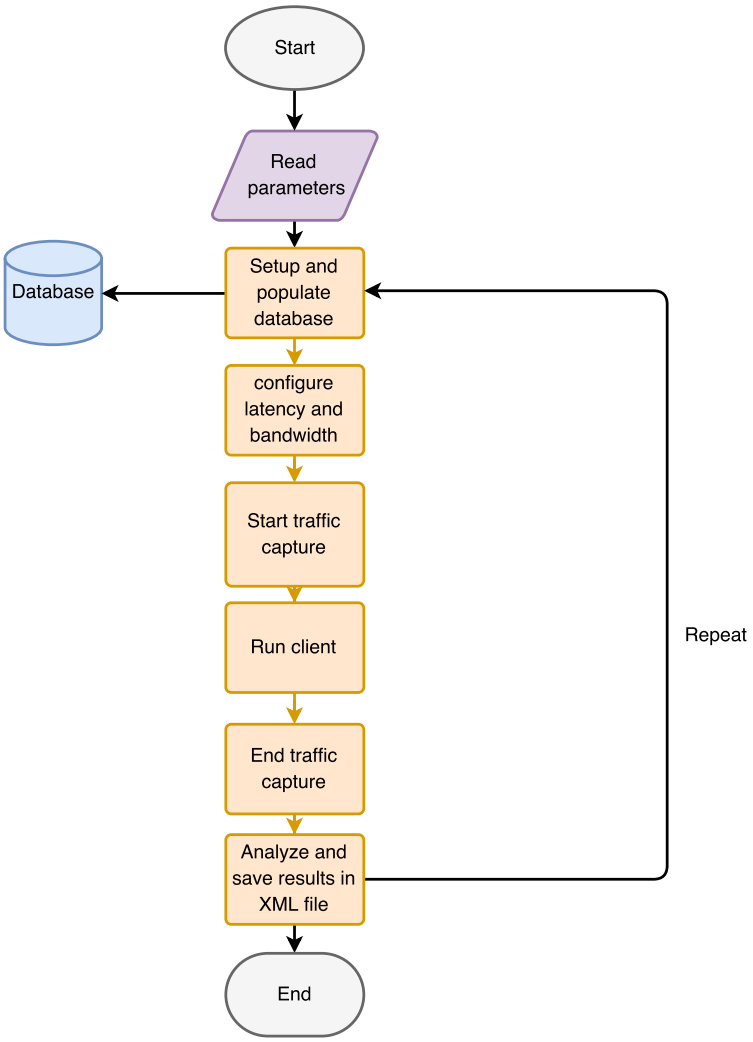

Figure 4.2: Flowchart of an experiment run

Data Population

To perform the experiments, we need to create data records and populate the server with them. For smart buildings, it is logical that we create records that represent measurements taken from sensors in the building. In the case of ZeroDB and ZODB, these records are objects of class `Measurement` stored in the database. While in the case

of MySQL, the records are stored as rows in a table called `Measurement`. Listing 4.1 shows the `Measurement` class attributes which directly translate to columns in `Measurement` table.

Listing 4.1: Measurement class definition

```
class Measurement(Model):
        roomID = Field()
        nodeID = Field()
        value = Field()
        date = Field()
        desc = Text()
        state = Field()
```

Three important keywords to note from Listing 4.1 are `Model`, `Field()` and `Text()`. `Model` is a class provided by ZeroDB and should be inherited when trying to create objects to be stored in ZeroDB. `Field()` denotes an attribute that should be indexed by ZeroDB, while `Text()` informs ZeroDB server that this attribute should support string searches.

The attributes of the class include the room and node IDs for the measurement, the value of the measurement and the date and time it was taken on. The remaining two are the `desc` and `state` which represent a text description and the state of the node creating the measurement respectively.

For MySQL, the table creation statement is shown in Listing 4.2 where ID column is added to be used as the primary key of the table.

Listing 4.2: Measurement table creation statement

```
"CREATE TABLE measurement ("
"ID INT UNSIGNED NOT NULL AUTO_INCREMENT,"
"roomID INT(4),"
"nodeID INT(5),"
"value INT,"
"date BIGINT,"
"description VARCHAR("+str(elementsize+10)+"),"
"state INT UNSIGNED,"
"PRIMARY KEY (ID))"
```

It is worth mentioning that in MySQL, the sizes of each column are set proportionally so as not to waste storage space. However, in ZeroDB and ZODB, this is not possible because there is no direct control over it, since it is handled by the underlying layer of object creation of the database itself.

The values to be stored in each attribute/column is randomly generated. For each

experiment and for each repetition for an experiment, new values are generated. This is done to remove any effect that the random generation of data might have on the performance. However, the values are still controlled to an extent so as to keep the experiments consistent. In the following, we give a description of the values stored in each of the attributes and columns.

- `roomID`: any of 150 random integers between 1111 and 9999 inclusive.
- `nodeID`: any of 900 random integers between 11111 and 99999 inclusive.
- `value`: any random integer between 1 and 9999 inclusive.
- `date`: actual seconds since epoch time (non randomly generated).
- `desc`: the string "door" concatenated with a randomly generated string of characters the size of which is determined in the experiments.
- `state`: any random integer between 0 and 3 inclusive, where each number represents a state of the node, for example 1 could represent in-use while 2 represents turned off.

Traffic Capture and Analysis

Before starting the client to query the database in each experiment, traffic capture has to be setup on the server network card. It is done by `dumpcap` which is a command-line tool to capture network traffic in Linux and is part of Wireshark which is the de-facto standard for traffic capture. The traffic capture is stopped after the client receives all its results back from the server.

The traffic for each run is saved in pcapng formatted files for subsequent analysis and named in a meaningful manner to denote exactly which experiment and run it belongs to. After that, the file is read by a python script to extract useful statistics from it. These include but are not limited to: the minimum and maximum time for the client and server to respond, the total bytes, the number of roundtrips, etc. The statistics for each file are finally saved as XML files.

These statistics are made after filtering out traffic not related to the experiment. It is worth mentioning that for ZeroDB, we include the SSL handshake used by the protocol in the statistics collecting since we deem it an integral part of the query process of ZeroDB.

4.2 Experiments

In essence, each experiment we will now discuss works by measuring the performance of varying the value of a single parameter out of the seven we mentioned at the beginning of this chapter (with the exception of multiple-condition queries), while preserving the

others in tact. The parameters and their default values in the experiments are shown in Table 4.2.

Table 4.2: Experiment parameters and their default values

Latency	10 milliseconds
Bandwidth	1000 Mbps
Database Size	50000 records
Query Size	1000 records
Data Distribution	uniform
Record Size	10 bytes

4.2.1 Experiment Ex1: Latency

Even though cloud usage is on the rise, some enterprises are still reluctant to use it, especially for time-critical operations, since the latency to the cloud could be a barrier to entry for them. That is why, we would like to investigate in this experiment the effects of latency on a chatty protocol like ZeroDB.

To manipulate the latency of the link between the client and the server we use the `TC` command in the `iproute2` package. `TC` allows for the configuration of traffic control in the Linux kernel. We enforce the latency to our desired limit by creating a queuing discipline `qdisc` on the client's interface, since typically clients are the parties with limited latency. The following Listing 4.3 shows the commands used.

Listing 4.3: Modification of latency using TC command

```
tc qdisc add dev $interface handle 1: root htb default 11
tc class add dev $interface parent 1: classid 1:1 htb rate 1000Mbps
tc qdisc add dev $interface parent 1:11 handle 10: netem delay $latency
```

`$interface` and `$latency` are variables representing the network interface and the latency of the link respectively. Values for latency to be tested are 10, 50, 100 and 1000 milliseconds. Note that we define latency here by the total time needed for a message to perform one roundtrip from client to server and back.

Due to the nature of ZeroDB protocol, we expect latency to greatly affect performance. For each query to the database, the client needs to send and receive a large number of messages in order to traverse the encrypted b-tree index on the server.

4.2.2 Ex2: Bandwidth

A parameter of concern for smart environments is the bandwidth between conversing nodes. A wireless node in a smart building such as a smartphone which uses cellular

communication can be limited in bandwidth compared to regular computers. This means that an important test is to measure the effects of low bandwidth on the performance of ZeroDB.

Similar to latency, we use the `TC` command to change the bandwidth of the link between the client and server. Listing 4.4, which should be viewed as a part following Listing 4.3 that dealt with latency change, shows the command used.

Listing 4.4: Modification of bandwidth using TC command

```
tc class add dev $interface parent 1:1 classid 1:11 htb rate $bandwidth
```

`$bandwidth` is a variable representing the bandwidth of the link. Values to be tested for it in the experiment are: 0.5, 1, 10, 50, 100 and 1000 Mbps. To verify the results of the command we perform bandwidth measurement tests using `iperf`.

It is expected that due to the encryption and protocol overheads, ZeroDB performance will be negatively affected by low bandwidth. To be exact, we expect that up to a certain point, low bandwidth will increase the overall time needed for a query to be executed and its results to be returned to the client. After that point, the amount of traffic transmitted at any given time would not exceed the bandwidth of the link.

4.2.3 Ex3: Database Size

Smart buildings generate a large amount of data that needs to be stored for current and future use. For example, energy consumption measurements are sent every number of seconds from an array of points in the building to the servers. Hence, a problem might arise when the size of the databases becomes too large. This could be the case when the indexing and searching algorithm cannot deal well with such database sizes. In this experiment, we want to measure the effect of the increase in database size to the performance of ZeroDB.

To change the database size, we simply increase the amount of unique random records with which we populate the database. The values for the number of records to be tested are: 5000, 50000, 150000 and 250000 records.

We expect that with the increase in the number of records in the database, comes the increase of the size of the b-tree index, which ultimately means more messages are sent and received to traverse it.

4.2.4 Ex4: Record Size

Another method to change actual database size, without changing the number of records like in the previous experiment, is to change the size of each record stored. A special

concern about ZeroDB is that unlike traditional databases, it stores actual objects and their attributes. Moreover, it also stores an additional overhead layer of encryption per object. This not only affects the size of data at rest, but also affects it while in transit over the network. Thus, it is interesting to quantify the different dimensions that storing and querying records with larger sizes pertain.

To change the size of objects, we increase the size of the attribute `desc`; since it is a text attribute, we can create increasingly larger values by concatenating more characters. It is worth noting that we do not simply concatenate the same character multiple times since this size increase could easily be mitigated by the database's compression algorithm. We chose to generate and concatenate random characters from the pool of printable characters of the String class in Python. In our experiments we test adding 1, 10, 25, 50 and 75 random characters to the string "door" as the value of `desc` attribute.

We expect that the size of records will increase the total bytes sent over the link and the time needed to complete the queries.

4.2.5 Ex5: Data Distribution

Typically, when storing measurement records regularly, a majority of records will have the same value for an attribute. For example, the majority of values for an attribute about the state of a node, would be "normal", so one challenge could be to quickly find, among the thousands of records, the few records with state "malfunctioned". Given that ZeroDB indexes attributes and stores them encrypted in a tree on the server, one could argue that the time needed to find a record with a specific attribute value could be affected by the overall distribution of the values for that attribute in the database. With this experiment, we set out to find the effects of querying records with a certain value for an attribute when that attribute's values are uniformly distributed versus when they are biased towards a given value.

We accomplish this in this experiment by changing the values of `state` attribute. In the uniform distribution scenario, we randomly generate records with equally likely state values between 0 and 3. While in the biased distribution scenario, we generate records where values 0, 1, 2 and 3 are %17.5, %65, %15 and %2.5 likely to occur respectively.

We expect that in the case of a biased distribution, the time needed to query the same number of records with a value that is most likely to occur, is lower than that when a value is least likely to occur. This also holds true in regard to the number of messages sent and received within that exchange.

4.2.6 Ex6: Query Result Size

As seen in Chapter 3, scenarios in smart buildings range from querying a single record for the HVAC settings of a user to querying large number of records to create accounting reports of the previous week. This variation in query result size is an interesting effect to quantify; since by measuring its effect on performance, we can justifiably state whether or not ZeroDB is suitable for each scenario.

To change the number of records returned in the query result, we use the limit keyword. The values of query result size to be tested are 1, 100, 1000 and 10000 records.

We expect that ZeroDB would have a comparable performance to the other databases for small query results. Nonetheless, we expect ZeroDB to become much slower and generate more traffic once the required number of records increases.

4.2.7 Ex7: Multiple-Condition Queries

Database queries often involve performing a statement that contains two or more conditions at once. In all the previous experiments only one condition is used. In this experiment however, we would like to assess the capabilities of ZeroDB to find records that satisfy two conditions at the same time.

The first condition is the one that is used in the other experiments, which is based equality query of one attribute. The other condition is a range query of the attribute timestamps. Furthermore, to remove the influence of the actual range values chosen on the position in the index tree, we vary the timestamp values to be queried while still taking into consideration that the returned query result size is always equal to 1000 records. This is done to enable us to compare the performance of this experiment with the query result experiment and conclude the performance penalty of adding a range query.

We expect that ZeroDB would have a significant performance decrease since adding the range statement means it has to fetch all the nodes in the subtree between the upper and the lower limit of the range, which means more traffic sent and more time needed to complete the query.

Chapter 5

Results Evaluation

In this chapter we present and evaluate the results of our performance experiments for encrypted databases. We start by detailing the results of each experiment, then we move on to discuss the various factors that affected the performance parameters during the experiments. Finally, we finish by concluding some guidelines on the applicability of encrypted databases in smart environments.

5.1 Experiments Results

In this section, we give an overview of the results of each experiment, discussing briefly the interesting effects that varying each parameter had on performance.

Variance in Results

Before discussing the results, we see it fit to first explain an issue we noted with how ZeroDB currently works. This issue affects mainly the total traffic sent by the server; we found that when comparing between the repetitions of the same experiment, the amount of traffic sent by the server most often was divided around two values.

When investigating further through the capture files, we noticed that in the tests that had higher amount of traffic, the client was sending one or more TCP reset packets to the server near the end of the test. However the server kept sending traffic before finally realizing that the client asked for the connection to be over. While in the tests that had lower amount of traffic, the client was also sometimes sending TCP reset packets, but the server seems to have realized in time and did not send any extra packets after that.

Comparing the ACK in the first TCP reset packet sent, we can trace back the packet that triggered it, and in all cases it was a packet sent by the server directly after a client had sent a packet itself. More interestingly, if we subtract all the traffic that was sent by

the server including and after that particular packet, we end up with almost the same value as the tests with the lower amount of traffic.

We believe that this behavior could be attributed to either tree pre-fetching or due to the client asking the server for multiple branches of the index tree, but ultimately reaching the amount of records it needs in the first few. To illustrate the second case, consider if a query was requested with "limit 1000" keyword, the client will remotely search in the index tree and at one point calculates that in three different branches there exists records that satisfy the query. The client asks the server to send these three branches, but by the time the client has received the first branch, it had already gotten the 1000 records it needs. So the server ends up sending extra traffic and the client sending TCP reset packets to inform the server that it had closed the connection.

Figure 5.1 shows a manifestation of this issue in experiment Four. We measure the total traffic sent by the server versus the size of records stored in the database. Without accounting for the issue of extra packets, we see on the left graph that no matter the record size, the various values for traffic sent by the server center around two different values an upper one equaling to around 257 KB and a lower one around 180 KB. Meanwhile on the left graph, we show that when we remove the extra packets sent by the server after the reset packet, all values now are around the lower average of 180 KB.

This issue is clearly visible in the amount of packets and traffic sent among the various metrics we analyze. However it also marginally affects the execution time and the amount of roundtrips (The exact definition of how we measure them both can be seen in the next section).

Note that in this chapter we present the data without removing the extra traffic sent by the server. We deem this extra packet sending as part of the current state of ZeroDB and by sanitizing the data we are actually misrepresenting the results. Therefore, while reading the remainder of this chapter, it is important to note that while the conclusions we make on the correlations between the various parameters and performance are accurate, the concrete values could have a skew to higher numbers due to this issue. However, this is sufficient enough for this thesis, since our goal here is to present the overall performance of encrypted databases such as ZeroDB.

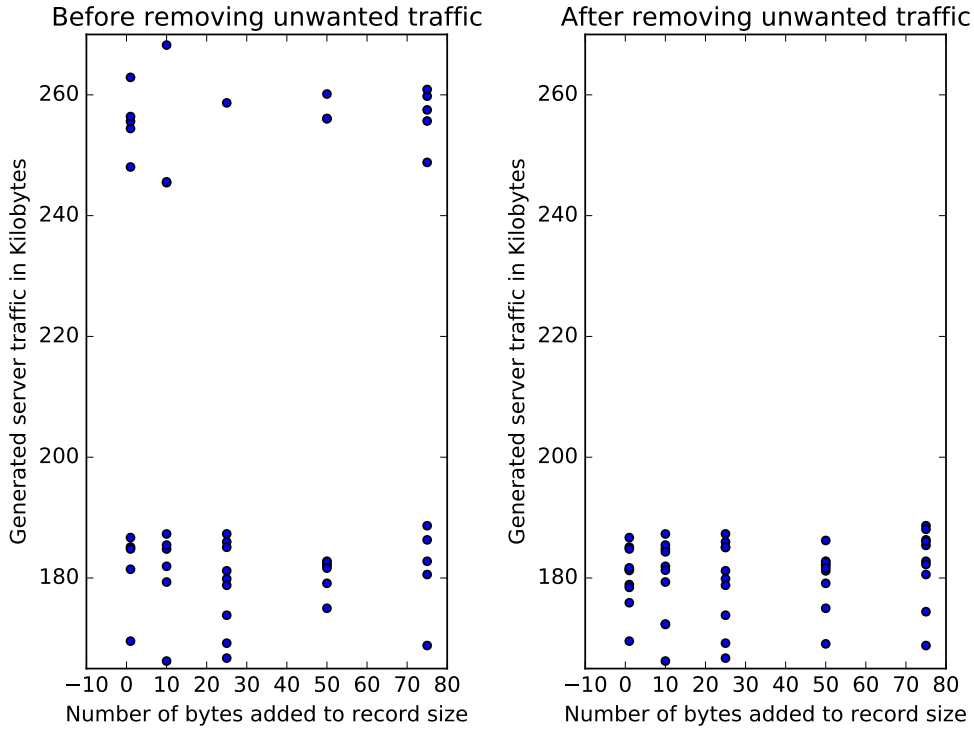

Figure 5.1: Difference in the amount of traffic sent by ZeroDB server in Ex4 (record size) when accounting for extra packets after client TCP reset packets

5.1.1 Ex1: Latency

Varying the latency in the experiments yielded a pronounced effect on the time needed for a query to be sent and its results to be returned to the client – henceforth called execution time. We calculate execution time by subtracting the time of the first packet after the TCP three way handshake and the last packet carrying data corresponding to the query.

In our experiments ZeroDB's execution time had almost a linear correlation with the latency, as did the other databases as well. However, ZeroDB had the most pronounced increase between the databases when we increased the latency; this is due to the number of roundtrips that ZeroDB needs to complete a query. Roundtrips are message exchanges between the client and server; that is the number of times one or more messages were sent in one direction and one or more messages were returned as a response. The number of roundtrips amplify the effect of latency, since for each roundtrip, the latency value must be added to the execution time. In our experiments, ZeroDB had between 32 and 44 roundtrips, so for any change in the latency we made, the effect was greatly multiplied.

Figure 5.2 depicts the increase in execution time versus the increase of latency for the three databases, notice the linear relationship between the two variables.

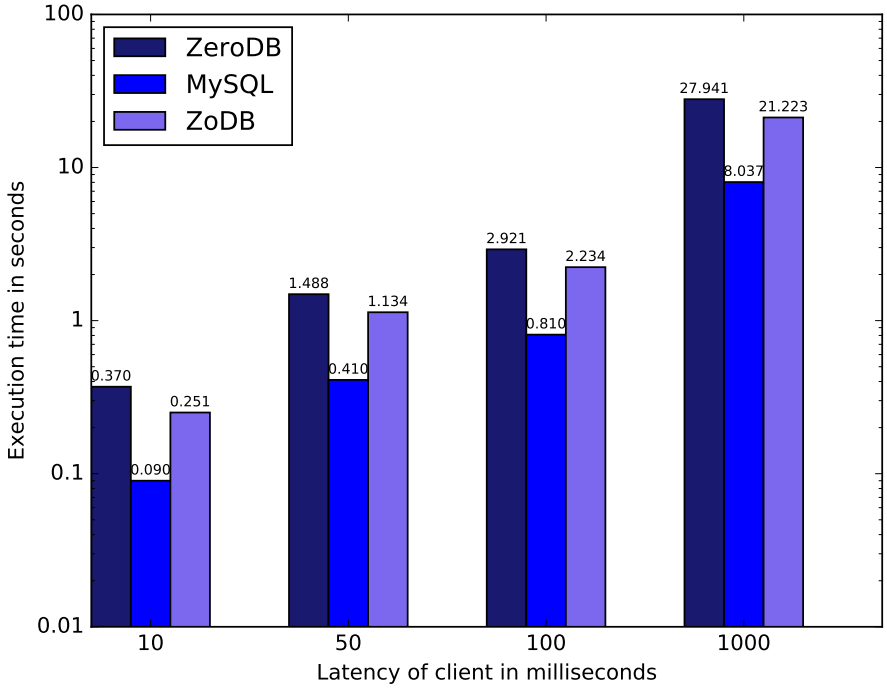

Figure 5.2: The correlation between the client latency and the execution time

Note that due to time constraints we only performed tests on ZODB for 10, 50 and 100 milliseconds, the results shown on the graph for 1000 milliseconds are calculated by linear interpolation.

5.1.2 Ex2: Bandwidth

Changing the bandwidth of the link between the server and client had a direct consequence on the execution time. This effect can be clearly seen in Figure 5.3 which depicts the execution time versus the bandwidth of the link. The execution time decreases from an average of 0.963 seconds for a 0.5 Mbps link to 0.684 seconds for a 1 Mbps link, the decrease continues also for a 10 Mbps link where the time reaches 0.390 seconds. However, any increase beyond this point results in no net gain, since at this bandwidth range, the latency of the link becomes the only dominant factor affecting time especially since ZeroDB is a chatty protocol.

To realize the point in which the bandwidth stops being a bottleneck for ZeroDB execution time for this type of query, we calculated the combined average throughput of

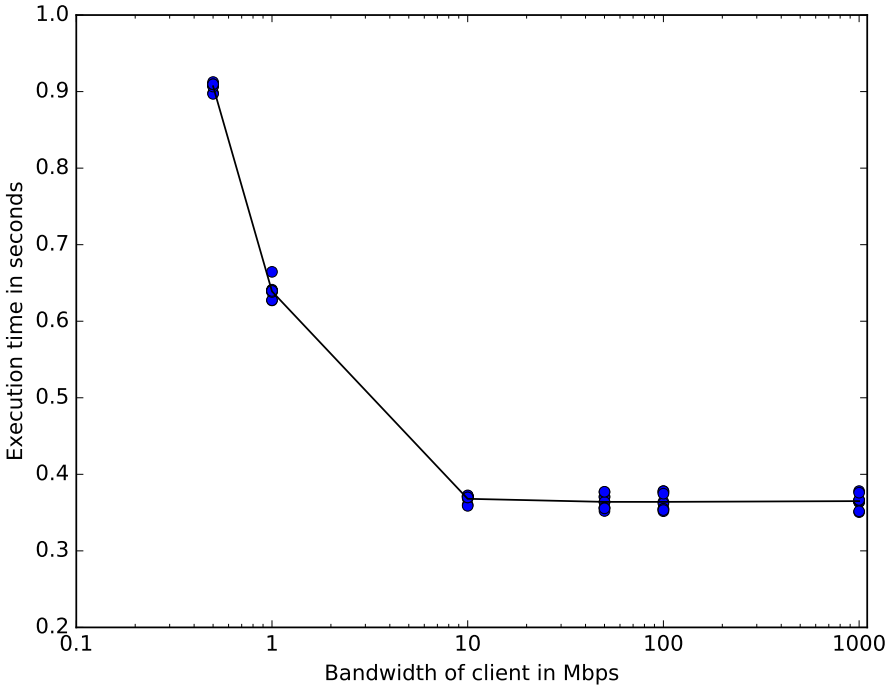

Figure 5.3: The correlation between ZeroDB client bandwidth and the execution time

ZeroDB client and server for links with 10, 50, 100 and 1000 Mbps. The average value was around 5.8 Mbps, which means at this point the execution time does not decrease with bandwidth increase.

For this size of queries, MySQL does not require a large bandwidth to operate since it sends raw data; that's why in our experiments MySQL execution time did not get affected by any change of bandwidth with a stable average value of 0.111 seconds throughout all the trials.

ZODB follows the same pattern as MySQL in regards to bandwidth, with the execution time being a stable value throughout the different bandwidth changes equal to 0.315 seconds.

5.1.3 Ex3: Database Size

In the third experiment, we varied the number of records stored in the database from 5000 to 250,000 records. However, the number of records to be queried remained the same at 1000. We noticed that with the increase of number of records, it took successively more roundtrips to complete the query which translated into more execution time, more packets exchanged and more overall bytes sent over the link as seen in Figure 5.4.

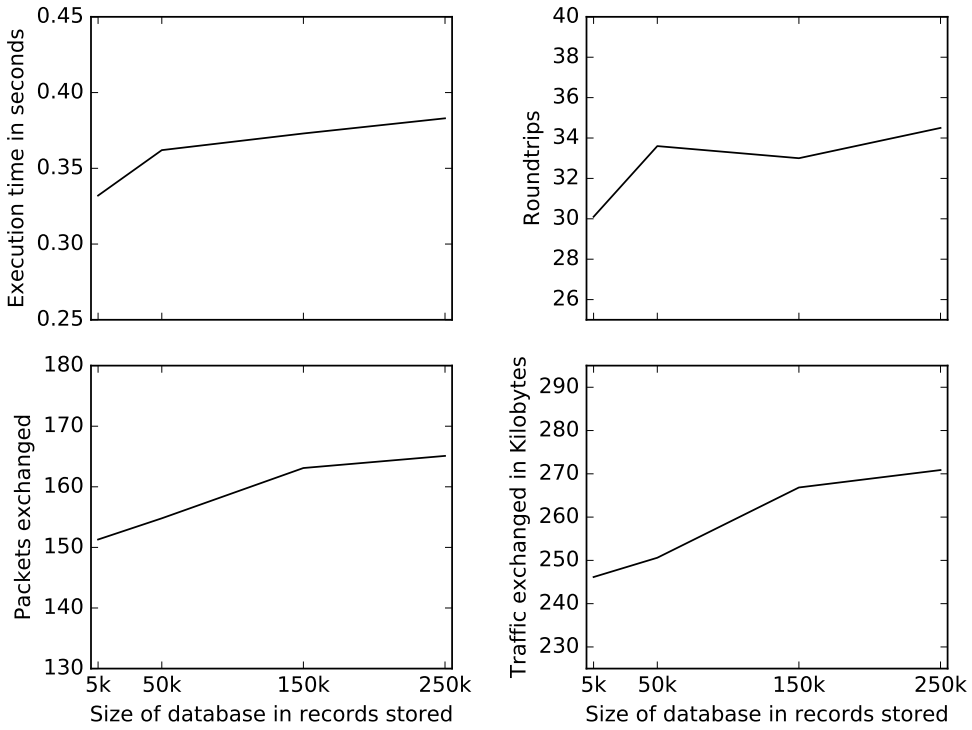

Figure 5.4: The correlation between ZeroDB database size and roundtrips, execution time and total packets and traffic exchanged

In ZeroDB, the client remotely traverses the index tree in roundtrips searching for the required record. Hence, this increase can be attributed to the fact that with the increase in the number of records, the size of the index tree is increased which in turn lowers the chance for the wanted record to be found in a branch of the tree.

For MySQL, the increase in the number of records did not have any effect on the performance. With the execution time and total bytes exchanged remaining stable around 0.111 seconds and 57533 bytes respectively no matter the database size. Notice how these values are much smaller than their respective counterparts in ZeroDB. As for ZODB, it behaved similarly in this experiment to ZeroDB; the increase in number of records directly increased the roundtrips, execution time and bytes exchanged over the wire.

Note that due to time constraints, ZODB tests were restricted to database sizes of 5000, 50000 and 150000 records and only repeated 5 times each.

5.1.4 Ex4: Record Size

The results of this experiment stood out from the rest in terms of how they change in ZeroDB compared to MySQL and ZODB. In this experiment the change in record size actually affected the performance of MySQL but did not have any significant effect on ZeroDB and ZODB. Note that - as we have previously explained - the record size in our experiment is the amount of random characters added to the end of a string in the record.

We have already discussed this partially towards the beginning of this chapter. As part of our discussion of the problem of extra traffic in ZeroDB, we have shown in Figure 5.1 how after removing superfluous traffic, the amount of traffic sent by the server is steady for all record sizes around an average of 180 KB.

Moreover, if we look into the roundtrips and execution time metrics, record size does not seem to affect them significantly with roundtrips value remaining steady at 33 roundtrips and the execution time slightly increasing by 0.01 seconds from 0.367 for record sizes of 1, 10 and 25 to around 0.377 seconds for record sizes of 50 and 75.

ZODB results also follow the same pattern with roundtrips, execution time and exchanged traffic staying equal to 21, 0.247 seconds and 140 KB respectively throughout the experiment. Note that for ZODB we did not test 75 as a record size value and we only performed 5 repetitions due to time constraints.

As for MySQL, the increase in the size of the record meant an increase in the amount of traffic exchanged which ultimately meant an increase in execution time. Figure 5.5 depicts the increase in traffic sent by the server. Note how the increase between each test is roughly equal to the difference in bytes appended to the record times 1000 (number of records queried).

We believe this can be justified by comparing how MySQL stores its records versus how ZeroDB and ZODB store theirs. In MySQL strings are stored as raw bytes where each increase means one additional byte to be sent over the network. On the other hand, the other databases store objects and properties of those objects in complex structures that are also compressed to save space.

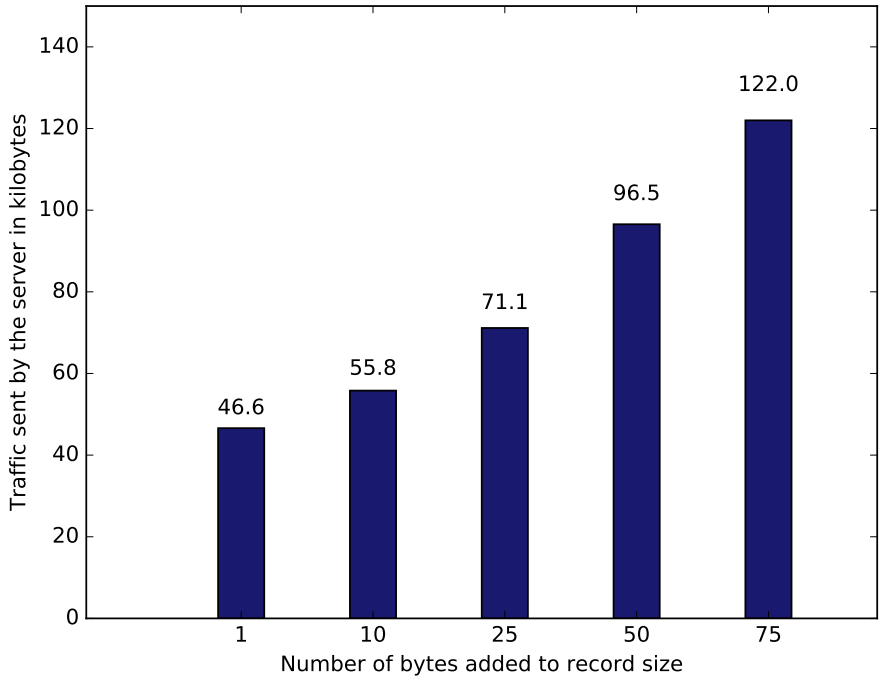

Figure 5.5: The correlation between record size and traffic sent by MySQL server

5.1.5 Ex5: Data Distribution

The results of the Ex5 are shown in Figures 5.6 and 5.7.

First, let us focus our attention on ZeroDB's execution time in Figure 5.6 and more specifically on the comparison between querying minority versus majority records. It is clear that ZeroDB suffers considerably when querying records which are the minority in the database. The case where the wanted records constitute only a minority of 2.5% of the records in the database needs on average 365% more time to be completed than that when they constitute a majority of 65%. As for the case where all data is uniformly distributed, we notice that its results are almost on par with the majority case.

Second, Figure 5.7 shows the amount of traffic exchanged between the server and client in the three distribution cases. We can see how even though we are querying the same amount of records in all three cases, ZeroDB sends considerably more data over the wire for querying records which are the minority.

For MySQL, querying minority records or the majority barely affected barely affected performance whether in terms of execution time or amount of traffic exchanged.

ZODB behaved similar to ZeroDB with the same trend of increase in execution time and

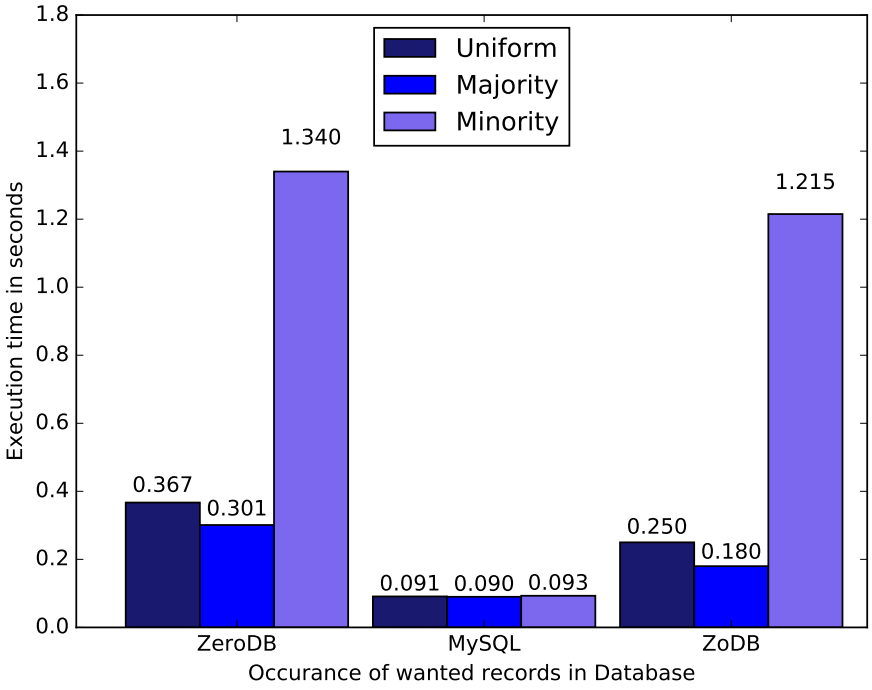

Figure 5.6: The correlation between distribution of data and execution time

total bytes exchanged with the change of data distribution from uniform to minority.

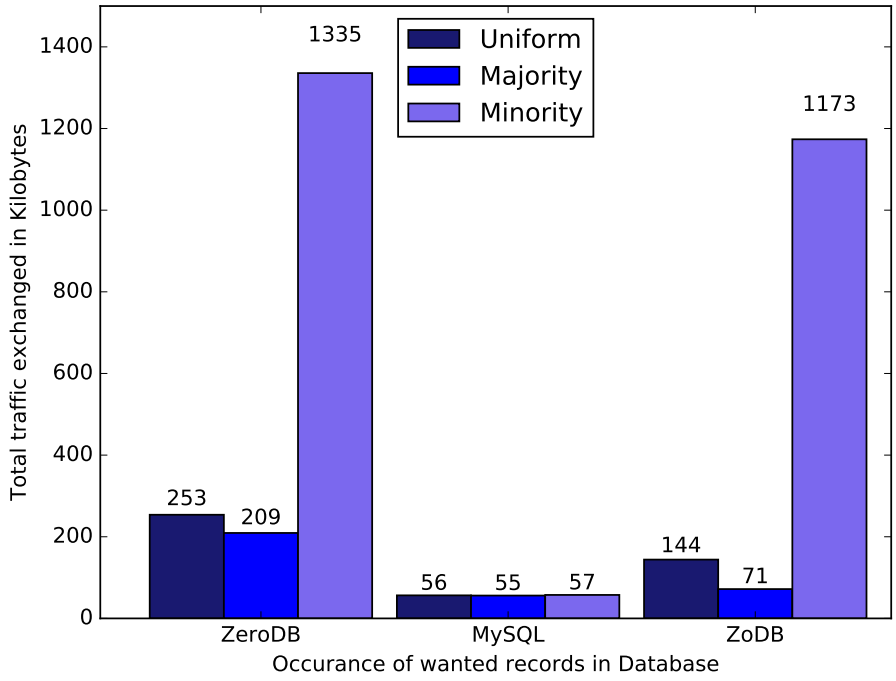

Figure 5.7: The correlation between distribution of data and total traffic exchanged

5.1.6 Ex6: Query Result Size

In this experiment, we changed the number of records returned by using the limit keyword. Its results for all three databases are summarized in Table 5.1. We can draw multiple interesting conclusions from analyzing the data in the table.

First, we note that the difference between querying 1 record and 100 records is barely noticeable in terms of roundtrips and consequently execution time. This is due to how ZeroDB fetches multiple nodes in parallel that we discussed previously. Moreover, we can also conclude that given the number of records in our database, the minimum number of roundtrips to find any value is 19 since it is directly dictated by the depth of the index tree of the server. Additionally, by dividing the results by the amount of records queried, we get a rough estimate of the efficiency of ZeroDB per record sent (these are the values between brackets in the table). For example, increasing the records queried greatly increases the efficiency of traffic sent by the server from 0.624 KB per record for 100 records to 0.136 KB per record for 1000 records. Note also how with the increase in query result size, the server is sending more records simultaneously. This is deduced by the roundtrips per record and execution time per record values shown between brackets.

Table 5.1: Performance analysis of changing query result size

Query size	1	100	1000	10000
	ZeroDB			
Roundtrips	19	20.6 (0.206)	33.8 (0.034)	128.7 (0.013)
Execution time (s)	0.199	0.223 (0.0022)	0.367 (0.00037)	1.66 (0.00017)
Total traffic (KB)	35.9	77.4 (0.773)	260.8 (0.261)	1432.7 (0.143)
Server traffic (KB)	31.2	62.4 (0.624)	203.2 (0.203)	1359.7 (0.136)
	MySQL			
Roundtrips	6	6 (0.06)	6 (0.006)	6 (0.0006)
Execution time (s)	0.062	0.064 (0.0006)	0.091 (0.00009)	0.26 (0.00002)
Total traffic (KB)	1.69	6.9 (0.069)	56.4 (0.056)	554.1 (0.055)
Server traffic (KB)	1.1	6.3 (0.063)	55.8 (0.0558)	553.5 (0.055)
	ZODB			
Roundtrips	12	12.3 (0.123)	21.6 (0.0216)	117.3 (0.0117)
Execution time (s)	0.130	0.1333 (0.0013)	0.247 (0.00247)	1.439 (0.000144)
Total traffic (KB)	29.3	33.0 (0.33)	140.5 (0.141)	1222.7 (0.1222)
Server traffic (KB)	27.7	31.4 (0.314)	137.6 (0.138)	1207.1 (0.1207)

It is interesting how the increase from 1 to 100 records barely affected the execution time needed for the query, but still the server sent more than double the amount of traffic which means for querying 1 record, the protocol and the connection itself are barely saturated. On the other hand, even though the percent increase from from 100 to 1000 is less than that of 1 to 100, it resulted in a more pronounced change with 164% and 165% increase in roundtrips and execution time respectively.

MySQL results follow the same trends discussed about ZeroDB with one major exception that is the roundtrips value. MySQL requires 6 roundtrips to complete any query no matter the number of records returned. Moreover, it suffices to say that MySQL is a magnitude more efficient than ZeroDB in terms of traffic exchanged and total time in all tested query sizes.

Finally, ZODB - being the underlying database of ZeroDB - follows the same patterns albeit with more efficiency. That is it needs on average slightly less roundtrips, execution time and traffic to perform the same queries.

5.1.7 Ex7: Multiple-Condition Queries

In this experiment, we added an extra condition for the queries, namely a range query based on the timestamp of records. The results of this experiment are listed in Table 5.2, we added to them the results of experiment Ex6 (query result size) for comparison and easy referencing.

Note that for ZeroDB, the addition of a range query significantly affected all aspects

Table 5.2: Performance results of returning 1000 records with and without the addition of range query

Query type	without range query	with range query
	ZeroDB	
Roundtrips	33.8	357.2
Execution time (s)	0.223	1.173
Total traffic (KB)	260.8	759.8
	MySQL	
Roundtrips	6	6
Execution time (s)	0.091	0.096
Total traffic (KB)	56.4	57.3
	ZODB	
Roundtrips	21.6	614.18
Execution time (s)	0.247	1.638
Total traffic (KB)	140.5	1359.1

of performance. The execution time for example increased by 520% even though the query result size stayed the same at 1000 records.

Meanwhile, MySQL performance barely changed by the addition of the extra condition with execution time and total traffic increasing by only 5% and 1% respectively.

ZODB, on the other hand, follows the same pattern as ZeroDB. However, it is interesting to note that the increase in execution time and total traffic was higher than that of ZeroDB, which could speak to the optimizations done by ZeroDB developers to ZODB.

Finally, it is worth mentioning that during the variation of the upper and lower limits in the range queries of this experiment, an increase in the variance of the results was noted in ZeroDB when compared to the previous results. This could be attributed to how the records that satisfy both conditions are spread in the index tree.

5.2 Results Discussion

5.2.1 Roundtrips

Due to their protocol's nature, roundtrips in ZeroDB and ZODB changed significantly in each experiment, while in MySQL the roundtrips remained equal to 6 in all the experiments. Furthermore, when we compare ZeroDB with ZODB, roundtrip numbers were almost always higher in ZeroDB than ZODB. The only experiment where ZODB needed more roundtrips was Ex7. This could be the case because in ZODB the client lazily asks for the set of records that satisfies the first condition and then locally evaluates the second condition on that set.

Furthermore, if we compare between the different experiments, we note that the amount of roundtrips needed for a query in ZeroDB is mainly influenced by the query result size, data distribution and use of multiple conditions in queries. The data distribution case however stands as a peculiar one, since as far as we can deduce from the source-code and white paper, ZeroDB uses a sorted B-Tree as index. Furthermore, we specifically coded our experiment in a way that the attribute in question is indexed. Thus, theoretically, the search algorithm should be able to find the needed records in only slightly more roundtrips.

The minimum roundtrips to perform any experiment in ZeroDB was 19 while the maximum reached upwards of 350 in Ex7. These are significant values for devices installed in locations where interference in wireless communication must be set to a minimum. Moreover, they also indicate that one must be certain that the client has a stable connection to the server, because the client will be actually conversing with the server and if at any point one packet is delayed or lost, the whole query comes to a halt.

5.2.2 Execution Time

Execution time of ZeroDB follows roughly a similar trend like roundtrips. However, factors that affected execution time alone are the bandwidth and latency of the client-server connection. However, we noted that, in the experiments, the influence of bandwidth on a typical query could be removed if the connection had a bandwidth of more than 5.8 Mbps.

When comparing ZeroDB with MySQL and ZODB, ZeroDB had higher latency almost always except for when we used multiple conditions like in the case of roundtrips. On average, ZeroDB had comparable execution time to MySQL when conducting simple queries with large amounts of records. However, difference between them increases when we compare more complex queries or change the latency and bandwidth of the link.

5.2.3 Total Traffic Exchanged

Network traffic exchanged between the client and the server of ZeroDB was significantly higher than that of MySQL. The main reason for this is the nature of ZeroDB's storage. In ZeroDB, data is stored as complex Python objects, while in MySQL only raw data types are stored. Another reason for this extra traffic is due to the ZeroDB remote tree traversal protocol which requires more traffic and extra unneeded nodes to be returned to the client.

When compared with ZODB, ZeroDB generated more traffic which is expected due to the ZeroDB query protocol and the encryption overhead. An exception to this was

noted in Ex 7, where ZODB generated more traffic, the reason for which is mentioned in Subsection 5.2.1.

Additionally, it should be noted that in ZeroDB, the client contributes a significant amount of data to the total traffic generated, unlike the case in MySQL. This could be an issue when the client upload is severely limited compared to its download bandwidth.

5.2.4 Database Size on Disk

As we have alluded before, ZeroDB and ZODB store records as objects, while ZeroDB stores them as raw bytes of data. Therefore, we expect a drastic difference between them when we compare the area occupied by their storage on the server disk. In Table 5.3 we show the area occupied versus the number of records stored in the database.

Table 5.3: Size of Database on Disk

Database	ZeroDB			MySQL			ZODB		
No. of Records	50k	100k	200k	50k	100k	200k	50k	100k	200k
Size on Disk (MB)	553	1402	2912	3.5	7.5	13.5	564	1334	2840

We can see that ZeroDB occupies a large amount on disk even for a moderately sized database of only 50000 records. This can pose an issue when a smart building generates large amounts of records regularly since size on disk could quickly become a bottleneck. A solution to this problem could be the use of an archive. However, care must be taken when archiving data in encrypted form, since key material and users access control policies must be carefully handled to avoid security and privacy issues.

Finally, in all database sizes, ZODB occupies just a little less disk space than that occupied by ZeroDB. This is to be expected, since ZeroDB actually uses ZODB as its underlying database system and adds on top of it some overhead due to encryption and structures that are needed to support the ZeroDB query protocol.

5.2.5 Server and Client Response Times

During our experiments, we analyzed the server and client response time, by measuring the delay for each party to respond. We wanted to evaluate whether or not ZeroDB server or client need more time along the execution of the protocol to compute or retrieve values. In all the experiments, the response time did not change significantly enough for us to report on it in Section 5.1; the change in values remained in the magnitude of 1 in a thousandth of a second, which is not large enough for its effect to manifest in real world scenarios. Moreover, it could be argued that such a minimal value could be simply attributed to the server or client processor being busy servicing other other unrelated processes.

5.3 Conclusions

Based on our results and discussion, we can now make some judgments on the applicability of ZeroDB in privacy preserving smart buildings.

- Care must be taken when placing the database server in the network; the latency and bandwidth of the clients in a smart building to the server must be carefully measured to ensure the performance requirements are met. For example, if the server is expected to hold access rights of employees to perform routine tasks like opening a door or using equipment, the latency should be kept to a minimum to ensure an interactive querying process is made with the server. Meanwhile, a measurement collecting database could be placed in an average latency setting, since interactivity is not highly required. Finally, we believe that ZeroDB performance allows it to be deployed in smart buildings as a cloud service given that what we have just discussed is taken into account.
- The amount of stored records in ZeroDB does not significantly influence its querying performance, which means that ZeroDB could be suitable for intensive information gathering smart building. However, attention must be directed to the size occupied on disk, since we have demonstrated that ZeroDB requires large disk space compared to relational databases like MySQL.
- ZeroDB's lack of full multi-user support and data sharing capabilities hinders it from being deployed in multi-actor smart buildings in the near future. However, given that partial support in this area is visible in the source code of ZeroDB, we believe that if developed in the right direction, ZeroDB could have the capabilities required to be deployed in smart buildings.
- Given the amount of traffic generated and the processing done by the client in ZeroDB. We believe that less capable devices like wireless sensors and small actuators should not directly communicate with ZeroDB server, but instead - as we proposed in Section 3.1.1 - communicate with a mediator that acts as a proxy.
- Based on the comparable performance of ZeroDB and its underlying database system ZODB, we believe that ZeroDB's privacy overhead is a reasonable one. In fact most of the performance difference between it and MySQL stems mainly from the intrinsic nature of its database type and not from the algorithms set in place to preserve data privacy.
- We believe that ZeroDB performance overhead and the nuances - we discussed in this chapter of using it - are a fair price to pay for the privacy it provides. We believe that if its capabilities are expanded, building operators will share our conclusion and see that the benefits of using ZeroDB out-shines the drawbacks and penalties it incurs.

Chapter 6

Conclusion

This chapter offers our concluding remarks for this thesis. We attempt to answer the research questions we posed in Chapter 1 and finish by discussing the limitations of the thesis and propose future work to address them.

6.1 Research Contributions

In this thesis, we presented the idea of using the novel concept of encrypted databases to protect users' privacy in smart environments. We focused our work on smart buildings as an example of smart environments.

We additionally performed a comparison of the encrypted databases currently available, from which we picked ZeroDB as a candidate database to be our research subject. Moreover, to assess the applicability and performance of a technology like ZeroDB, we devised and implemented experiments where we varied one of various factors and noted the effect it had on different performance metrics.

In the following, we present our research contributions by answering the research questions we posed in Chapter 1 based on our analysis and the performance results of ZeroDB as a representative of client centric encrypted databases.

- **Question Q1:** How should a data processing system in a smart building or other smart environments be structured to allow for privacy preserving characteristics?

  In Chapter 3, we demonstrated that to use encrypted databases, a smart building would be divided into localities which are treated as unique clients to the encrypted database. A small computing device in each locality will communicate data to and from the encrypted database and between the sensors and actuators within a locality. In this manner, data within one locality would be private to that

locality and only shared with others when necessary. Moreover, the honest but curious building or server operators cannot violate user's private data.

- **Q2:** What are the limitations of using encrypted databases in smart buildings?

  Encrypted databases' performance suffers from high latency, which means that they are greatly limited in their applicability by the client to server latency. Moreover, multi-user support and data sharing between users are still not mature enough to enable the full deployment of encrypted databases in the architecture we propose. Furthermore, encrypted databases occupy large server-side disk space for storing records with 200 thousand records occupying around 2911 MB of space compared with only 13.5 MB for a database like MySQL. Finally, unlike traditional databases, the client plays a significant role in the process which means that it is required to have more processing and networking capabilities in order to perform adequately.

- **Q3:** How fine grained and flexible can access control to encrypted data be performed using encrypted databases?

  Access control is currently limited to allowing a single user to access their own private data. In terms of multi-users and sharing, access control is non-existent at the moment with no easy way for users to share data with each other. Moreover, users cannot efficiently compute functions on data from multiple users, which is provided by technologies like secure multi-part computation.

- **Q4:** What are the main factors that affect the performance of encrypted databases and what is their impact?

  Our results in Chapter 5 show that the main factors that affect performance are latency and distribution of stored data. By far, latency is the greatest factor, with any increase in latency having a pronounced effect on time needed to complete a query. The distribution of data stored has the potential of affecting the time and amount of traffic exchanged if the record sought after is less likely to occur, such as the case when searching for the set of sensors that are malfunctioning.

- **Q5:** What are the performance effects of using encrypted databases compared to non-encrypted ones?

  In all of our conducted experiments in Chapter 5, ZeroDB required more time and roundtrips, and generated more traffic to complete a query when compared with MySQL. The difference however varied by the conditions and type of the query. Moreover, compared with its underlying database technology ZODB, ZeroDB's performance was comparable and followed the same trends as ZODB and even outperformed it in Ex7 that used range queries. This means that ZeroDB's privacy and encryption overhead is considerably reasonable. Ultimately, ZeroDB's

performance - while is not prohibitively poor - still demands that extra care be taken when deploying it as a solution in smart environments.

In conclusion, ZeroDB currently lacks of full multi-user support and data sharing capabilities which prohibits it from being deployed in multi-stakeholder environments like smart buildings. Nevetheless, we see it as a promising technology that holds the potential to succeed if developed in the right direction. Furthermore, while ZeroDB's performance lags behind that of traditional plain-text databases, we recognize that it could be deemed as an acceptable price to pay to protect privacy, and that future solutions could be deployed in a manner that keeps the performance overhead to a minimum based on our analysis of the main contributing factors to performance.

6.2 Limitations and Future Work

Despite our results and what we have accomplished in this thesis, there is still much more room to achieve in this area of research. We identify the following main limitations that we believe our thesis suffered from and derive from them future work proposals that improve and expand upon our thesis.

- In our experiments, we only tested the performance of using Select type statements as queries to the database. It would be interesting to compare performance of other types like Insert, Update and Delete.
- We only tested one encrypted database as a candidate in our experiments. While ZeroDB was the most logical choice for us to use in the thesis, we believe that in the future it is inevitable for other open-source encrypted databases to be developed. Future work could look into comparing the performance of these databases.
- Although we didn't explicitly turn off client-side caching for ZeroDB, we did restart the client between each experiment and repetition, which means that the client always performed a cold start where the cache is empty. Future work could look into the use of the cache and analyze the performance benefits of using it.
- The dataset used in the experiments was handcrafted by us. We identify that it might not be optimally constructed to compare the general performance of databases. That's why one direction for future work could address this by using standard benchmark data like TPC-C [40].
- A smart building has many users, but due to ZeroDB constraints, we had to perform our experiments for a single user. This limits the applicability of our results onto real world scenarios. A future area to explore is to test multi-user databases and to also explore the performance implications of multi-user data sharing and aggregation.

- Finally, we focused in this thesis on smart buildings as one example of smart environments. However, the future of the Internet of Things (IoT) is creating much larger environments like smart campuses and smart cities. These pose more challenges on the privacy front and more strain on databases which support them. It is interesting to analyze how our solution could be adapted and expanded to accommodate such environments.

Appendix A

Typical Use Cases In Smart Buildings

The following are use cases that stem from the scenarios of smart buildings listed in Chapter 3.

Number	1
Name	Data is shared between localities
Description	A locality requests data from a locality under the same company.
Actors	Logic unit A, Logic unit B.
Post-condition	• A reply is received by Logic unit A. • Data is shared in a way that grants access only to locality A.
Main Course	1. Logic unit in Locality A sends a request to the logic unit in locality B to share specific data. 2. Logic unit B receives the request and checks the access rights of the data requested and decides it can share the data. 3. Logic unit B sends the data with its access rights tagged.
Alternate Course	Data cannot be shared due to access rights and the data request is denied.

Number	2
Name	Data is shared locally between sensors/actors and logic unit of same locality.
Description	The simplest form of data sharing, could be a temperature sensor that senses an increase in heat and informs the fan to run faster accordingly, or could be the RFID sensor reading an employee badge and sending the data to the logic unit of the room.
Actors	Sensors, actors and logic unit within a locality.
Post-condition	• Data never leaves the locality.

	• Data is received by the intended device tagged with the correct access.
Main Course	1. A device sends data belonging to a user or the room tagged with identifying information regarding owner and rights to another device in the same locality. 2. Data is sent encrypted and received by the other device.

Number	3
Name	Arbitrary data is shared between localities.
Description	Arbitrary data is sometimes needed to be shared between users (a private file), between localities (a firmware update), across the entire building (a building-wide announcement). This data could be private or public.
Actors	Users, devices, logic units (any actor in the smart building).
Post-condition	• Privacy policies are not broken. • Data is sent securely and tagged appropriately.
Main Course	1. The actor initiates the sharing process. 2. Intended entities receive the sent data protected as necessary.

Number	4
Name	Energy consumption optimization between physically close localities
Description	To optimize energy consumption, two or more physically close localities can calculate and agree on the optimal setting of an HVAC system for example.
Actors	Logic unit A, Logic unit B, Logic unit C.
Triggers	A locality wishes to change its current setting (for example, due to user entering the room)
Post-condition	• All rooms agree on the optimal setting for each one. • Privacy preserving properties (TBD)
Main Course	1. Triggering logic unit A contacts logic units B and C informing them of the change of setting. 2. Logic units share their current settings as well as their desired ones. 3. Logic units calculate the optimal setting for each locality where the desired setting is as close as possible to the real one while providing optimal energy consumption.

Notes	At the end of this use case, a chain reaction of optimization routines with neighboring rooms might start. Thus it is perhaps of the interest of admins to have an indication of the "stability of the parameters" of the system.

Number	5
Name	User usage pattern or preferences are saved in the database
Description	After each use of a smart device, usage patterns or preferences are stored in a company database in such a way that only the user can access.
Actors	User, Usage pattern/settings database, device.
Pre-Condition	A user finishes using a smart device in the building
Post-condition	• Data is received by the database securely. • Data is stored in the database and can only be accessed with the permission of the user.
Main Course	1. The device collects all relevant information about the user (for example, energy consumption or preferred settings). 2. Device asks the user if they accept that the data that has been collected be sent to the database. 3. User accepts this request and optionally clicks on always send data on this device prompt. 4. Logic unit sends data encrypted and privacy preserved to the database.
Alternate Course	User denies the request and the data is not sent.

Number	6
Name	User preferred settings (usage patterns) is retrieved from database.
Description	When a user enters a locality or access a device, user preferred settings are retrieved from the database to guarantee convenience and comfort.
Actors	User, user preferences database, device or logic unit of locality.
Triggers	A user accesses a device or enters a locality.
Post-condition	Data is received by the requesting party securely.
Main Course	1. The device or logic unit recognizes the user by his RFID or any other means. 2. The device or logic unit queries the database for existing settings for the user.

	3. The device or logic unit triggers setting change to accommodate for user preferences.
Alternate Course	No preferred settings are found in the preferences database.

Number	7
Name	Access rights increase in emergency.
Description	In case of emergency, access rights of actors in the system could be temporarily increased to hasten and guarantee user and equipment safety.
Pre-conditions	• A user has already given consent to this beforehand. • Emergency has been triggered.
Post-condition	• Access rights are reverted to normal after the emergency. • User is notified of the exact date and reason of access and exact data accessed.
Main Course	1. Access rights are increased on a locality basis in the area affected by the emergency. 2. Actors accessing the data in these localities are checked against emergency level access rights and are granted access accordingly.
Notes	• Devices in case of fire emergency could be faulty and thus should be treated with care when granting them rights to issue commands to actuators. • Similar to a "break glass" scenario.

Number	8
Name	Data is shared with external entity.
Description	Data originating from the smart building could be shared to an external entity for example regarding water/energy consumption or building occupancy.
Actors	Smart building central server, external entity.
Triggers	The need to share data is recognized by the central server.
Post-condition	• Data is shared securely with external entity. • If user data is part of the data exchange, the user must be notified of the exact date and reason of sharing.
Main Course	1. Data request is evaluated for privacy criticality. 2. Data is processed to leave as little as privacy critical information as possible. 3. Processed data is evaluated against privacy criticality rules. 4. If it passes, it is shared with external entity.

Alternate Course	Data cannot be shared due to failure in privacy check.

Number	9
Name	Logic unit requests data aggregate.
Description	Periodically - or on demand – a logic unit requests an aggregate of data from multiple other localities (or multiple users).
Actors	Logic units of multiple localities.
Post-condition	Correct data aggregate is received without breaching privacy of any entity and securely.
Main Course	Data is already stored in a central database. 1. Logic unit requests data aggregate from database 2. Database computes data aggregate without revealing any pieces of original data. 3. Data aggregate is sent to requesting entity.
Alternate Course	Data is not stored in a central database, which triggers the central database to query the different localities for their data first.

Number	10
Name	Device Discovery and identification
Description	Devices must be able to discover and most importantly identify each other and their access rights.
Actors	Devices of the same locality.
Triggers	A new device is turned on and connected to the network.
Post-condition	Devices are correctly discovered and identified.
Main Course	1. Neighboring Logic unit receives some communication initiation message and replies to the device. 2. Device presents information regarding its capabilities and most importantly regarding its identity that can be trusted. 3. Data aggregate is sent to requesting entity. 4. Access rights are assigned to device and saved for future need.
Alternate Course	Device fails to present trustworthy identifying credentials and is not accepted in the network.

Appendix B

Setup of Database Servers

In this appendix, we give a brief overview of how each database server was setup, such that readers who want to engage in the thesis can - if they wish to - reproduce the thesis results independently.

B.1 ZeroDB

ZeroDB requires python 3.5 to work. In our lab setup, python3.4 was the default, so we needed to install python3.5 and make it the default python version in the system.

After setting up python, we installed `pip3` the package management system in order to install ZeroDB through it.

The next step involved installing required packages by ZeroDB. The packages are: `build-essential, python3-dev, libffi-dev, libssl-dev`

Finally, the last step to install ZeroDB is to run the relevant `pip3` commands. Note that we used zerodb version 0.2.0b2 for the server and 0.99.0b1 for the client.

```
pip3 install zerodb-server==0.2.0b2
pip3 install zerodb==0.99.0b1
```

At this point, ZeroDB should be runnable by using `zerodb-server` command in a linux shell. However, configuration files must be available for the previous command to work. The configuration files can be generated by using `zerodb-manage init_db`, which creates the configuration files based on interactive input by the user.

The last remaining aspect needed to conduct the experiments was to automate the creation of zerodb configuration files; We needed to do without the interactive element of configuration files creation. For this purpose we used `expect` command that allows

the user to record an interaction in the shell and repeat it for the use in a scripting environment.

B.2 MySQL

For MySQL, the installation is straightforward using apt-get command shown below.

```
sudo apt-get install mysql-server-5.7
```

The configuration files for MySQL were kept in their default values.

B.3 ZoDB

For ZoDB installation, we used the straightforward command provided by `pip3`.

```
pip3 install zodb
```

Bibliography

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610001568

[2] A. Iera, C. Floerkemeier, J. Mitsugi, and G. Morabito, "The internet of things [guest editorial]," *IEEE Wireless Communications*, vol. 17, no. 6, pp. 8–9, December 2010.

[3] E. Welbourne, L. Battle, G. Cole, K. Gould, K. Rector, S. Raymer, M. Balazinska, and G. Borriello, "Building the internet of things using rfid: The rfid ecosystem experience," *IEEE Internet Computing*, vol. 13, no. 3, pp. 48–55, May 2009.

[4] J. A. Stankovic, "Research directions for the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3–9, Feb 2014.

[5] F. Bao and I.-R. Chen, "Dynamic trust management for internet of things applications," in *Proceedings of the 2012 International Workshop on Self-aware Internet of Things*, ser. Self-IoT '12. New York, NY, USA: ACM, 2012, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2378023.2378025

[6] D. Snoonian, "Smart buildings," *IEEE Spectrum*, vol. 40, no. 8, pp. 18–23, Aug 2003.

[7] B. Swam, "Internetworking with bacnet," *Engeenered Systems Magazine*, 1997.

[8] 2013. [Online]. Available: http://advancedcontrolcorp.com/blog/2013/03/intelligent-building-management-systems-in-miami/

[9] L. Wang, Z. Wang, and R. Yang, "Intelligent multiagent control system for energy and comfort management in smart and sustainable buildings," *IEEE transactions on smart grid*, vol. 3, no. 2, pp. 605–617, 2012.

[10] M. da Graça Carvalho, "EU energy and climate change strategy," *Energy*, vol. 40, no. 1, pp. 19–22, 2012.

[11] L. Pérez-Lombard, J. Ortiz, and C. Pout, "A review on buildings energy consumption information," *Energy and buildings*, vol. 40, no. 3, pp. 394–398, 2008.

[12] S. T. Bushby, "Bacnet tm: a standard communication infrastructure for intelligent buildings," *Automation in Construction*, vol. 6, no. 5, pp. 529–540, 1997.

[13] H. Shahnasser and Q. Wang, "Controlling industrial devices over TCP/IP by using LonWorks," in *Global Telecommunications Conference, 1998. GLOBECOM 1998. The Bridge to Global Integration. IEEE*, vol. 2. IEEE, 1998, pp. 1309–1314.

[14] [Online]. Available: https://www.knx.org/knx-en/knx/association/what-is-knx/

[15] C. Steward Jr, L. A. Wahsheh, A. Ahmad, J. M. Graham, C. V. Hinds, A. T. Williams, and S. J. DeLoatch, "Software security: The dangerous afterthought," in *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on*. IEEE, 2012, pp. 815–818.

[16] J. Kaur, J. Tonejc, S. Wendzel, and M. Meier, "Securing bacnet's pitfalls," in *IFIP International Information Security Conference*. Springer, 2015, pp. 616–629.

[17] M. Nogueira, "Anticipating moves to prevent botnet generated DDoS flooding attacks," *arXiv preprint arXiv:1611.09983*, 2016.

[18] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '02. New York, NY, USA: ACM, 2002, pp. 216–227. [Online]. Available: http://doi.acm.org/10.1145/564691.564717

[19] B. Prasanna and C. Akki, "A survey on homomorphic and searchable encryption security algorithms for cloud computing," *Communicated to Journal of Interconnection Networks*, 2014.

[20] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[21] M. Bellare, A. Boldyreva, and A. O'Neill, *Deterministic and Efficiently Searchable Encryption*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 535–552. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74143-5_30

[22] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of computer and system sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[23] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Theory of Cryptography Conference*. Springer, 2011, pp. 253–273.

[24] S. Chatterjee and M. P. L. Das, "Property preserving symmetric encryption revisited," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 658–682.

[25] C. Bösch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, p. 18, 2015.

[26] A. Arasu, K. Eguro, R. Kaushik, and R. Ramamurthy, "Querying encrypted data," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 1259–1261.

[27] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 85–100.

[28] M. Egorov and M. Wilkison, "Zerodb whitepaper," *arXiv preprint arXiv:1602.07168*, 2016.

[29] H. Chen, T. Finin, and A. Joshi, "A context broker for building smart meeting rooms," in *Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium, 2004 AAAI Spring Symposium*, 2004, pp. 53–60.

[30] I. Armac, A. Panchenko, M. Pettau, and D. Retkowitz, "Privacy-friendly smart environments," in *2009 Third International Conference on Next Generation Mobile Applications, Services and Technologies*, Sept 2009, pp. 425–431.

[31] R. Neisse, G. Steri, I. N. Fovino, and G. Baldini, "Seckit: A model-based security toolkit for the internet of things," *Computers and Security*, vol. 54, pp. 60 – 76, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404815000887

[32] R. Poddar, T. Boelter, and R. A. Popa, "Arx: A strongly encrypted database system," Cryptology ePrint Archive, Report 2016/591, 2016, http://eprint.iacr.org/2016/591.

[33] P. Grofig, I. Hang, M. Härterich, F. Kerschbaum, M. Kohler, A. Schaad, A. Schröpfer, and W. Tighzert, "Privacy by encrypted databases," in *Annual Privacy Forum*. Springer, 2014, pp. 56–69.

[34] S. Bajaj and R. Sion, "Trusteddb: A trusted hardware-based database with privacy and data confidentiality," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 752–765, 2014.

[35] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan, "Orthogonal security with cipherbase." in *CIDR*. Citeseer, 2013.

[36] L. Ferretti, F. Pierazzi, M. Colajanni, and M. Marchetti, "Scalable architecture for multi-user encrypted SQL operations on cloud database services," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 448–458, Oct 2014.

[37] "MySQL," 2016. [Online]. Available: https://www.mysql.com/

[38] "MySQL customers," 2016. [Online]. Available: https://www.mysql.com/customers/

[39] “ZODB - a native object database for python - ZODB documentation,” 2016. [Online]. Available: http://www.zodb.org/en/latest/

[40] “TPC-C,” 2016. [Online]. Available: http://www.tpc.org/tpcc/