TÉCNICO LISBOA

A secure data dissemination scheme supporting conjunctive keyword searchable encryption for connected vehicle networks

Afonso Maria Outeiro Teixeira Pimentel

Master Thesis in

Electrical and Computers Engineering Engineering

Supervisor(s): Prof. Paulo Rogério Barreiros D'Almeida Pereira
Prof. Naercio David Pedro Magaia

Examination Committee:

Chairperson: Prof. João Luís Da Costa Campos Gonçalves Sobrinho
Member of the Committee: Miguel Filipe Leitão Pardal
Member of the Committee: Paulo Rogério Barreiros D'Almeida Pereira

October 2022

Declaration

I declare that this document is an original work of my own authorship and that it fulfils all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

Gostaria de aproveitar este espaço para agradecer a todos os professores que me acompanharam nesta jornada académica, por me transmitirem não só respostas, mas perguntas e desafios que me fizeram avançar nesta longa jornada. Um agradecimento em especial aos Professores Paulo Rogério e Naercio Magaia que me acompanharam na reta final da minha trajetória e me deram uma ajuda inestimável para poder concluir esta dissertação. Este trabalho enquadra-se no âmbito do projecto SEEDS (H2020-MSCA-RISE sob o número 101006411).

Também gostaria de ressaltar o apoio fundamental da minha família que sempre acreditou em mim e foi a fonte inesgotável da minha confiança. São exemplos que me fazem aspirar ser mais e melhor e são a minha grande motivação para atingir (ou pelo menos tentar) a excelência.

Agradeço também aos meus amigos e colegas que me acompanharam nesta jornada e me ajudaram, não só neste projeto, mas ao longo de toda a vida académica.

Por fim fica a minha gratidão aos pesquisadores da área de IoV e Criptografia, que disponibilizaram os resultados do seu trabalho e alavancam a comunidade cientifica como um todo. Espero que está tese possa contribuir de alguma forma para o desenvolvimento dessas áreas.

Resumo

Existe a necessidade crescente de garantir a segurança das comunicações e as soluções modernas para esse problema passam pelo uso da criptografia. Hoje, os esquemas utilizados encriptam os dados que passam a ser inutilizáveis para todos os utilizadores que não tenham a chave.

O carácter dinâmico das redes veiculares exige que as comunicações sejam feitas com velocidade e a sobrecarga que a criptografia clássica impõe pode ser suficiente para inviabilizar a troca de mensagens. Pensado nisso, este trabalho foca-se em explorar esquemas que geram dados encriptados com capacidade de busca para desenvolver protocolos eficientes em redes veiculares. A ideia é que ao adicionar mais funcionalidade aos dados encriptados, novos protocolos possam ser criados.

Neste trabalho estudámos os trabalhos desenvolvidos em *Searchable Encryption* e analisámos as implementações de alguns esquemas mais promissores. Nessa análise percebemos que os esquemas disjuntivos (*queries* com OR) baseados em Linear Secret Sharing eram equivalentes do ponto de vista prático a esquemas conjuntivos (*queries* com AND). Modificámos então um esquema conjuntivo para fazer *queries* disjuntivas e verificámos melhora na performance de verificação da *query*.

Criámos dois protocolos simples para redes veiculares que usassem *Searchable Encryption* e testámo-los em um simulador de redes, o NS3. No simulador criámos um cenário realista e concluímos que protocolos que enviam *cyphertexts* de SE não são práticos. Terminamos por propor algumas soluções alternativas para esse problema.

Palavras-chave: *Searchable Encryption*, IoV, ABE, NS3, VANET

Abstract

There is a growing need to ensure the security of communications, and the modern solution to this problem is using cryptography. Classic cryptographic schemes obtain this security by encrypting the data, which becomes unusable for all users who do not have the secret key.

The dynamic nature of vehicular networks, however, requires communications to be carried out with speed. The overhead that classical encryption adds can be enough to make the exchange of messages unfeasible. With that in mind, this work focuses on exploring cryptographic schemes that generate searchable encrypted data and leverage that in the development of efficient protocols in vehicular networks. The idea is that by adding more functionality to encrypted data, we can compensate for the overhead in computation.

In this work, we searched the literature on Searchable Encryption and analyzed implementations of some schemes. In this analysis, we realized that disjunctive schemes (queries with OR) based on Linear Secret Sharing were, in practice, equivalent to conjunctive schemes (queries with AND). We then modified a conjunctive scheme to make disjunctive queries and verified an improvement in the performance of the query verification speed.

We created two simple protocols for vehicular networks that used Searchable Encryption and tested them in a network simulator, NS3. In the simulator, we created a realistic scenario that made use of traces with the movement of cars and generated traffic between nodes to imitate the behavior of a standard network. We concluded that protocols that send encrypted SE indexes are unfeasible and proposed solutions to it.

Keywords: Searchable Encryption, ABE, IoV, NS3, VANET

Contents

List of Figures

Nomenclature

$\mathbf{pk}$ Public Key

$\mathbf{sk}$ Secret Key

$\mathbf{T_Q}$ Trapdoor for query Q

ABE Atributed-Based Encryption

AF Scheme SE Scheme adapted from [1]

BDH Bilinear Diffie-Hellman

BN Barreto-Naehrig

BSM Basic Safety Message

CLE Certificateless Encryption

CW Scheme SE Scheme developed in [2]

DDH Decisional Diffie-Hellman

DSRC Dedicated Short-Range Communication

FE Functional Encryption

HTTP Hypertext Transfer Protocol

IBE Identity Based Encryption

IoT Internet of Things

IoV Internet of Vehicles

IP Internet Protocol

LSSS Linear Secret Sharing Scheme

LTE Long Term Evolution

LWE Learning with errors

MAC Medium Access Control

MNT Miyaji, Nakabayashi and Takano

NS3 Network Simulator 3

OBU Onboard Units

PE Predicate Encryption

PHY Physical

PKG Public Key Generator

PKI Public Key Infrastructure

RSU Roadside Units

SE Searchable Encryption

SPE Searchable Public-key Encryption

SSE Searchable Symmetric Encryption

SS Stress–Strain

TF Scheme SE Scheme developed in [3]

UDP User Datagram Protocol

V2I Vehicle to infrastructure

V2R Vehicle to road

V2S Vehicle to sensor on-board

V2V Vehicle to vehicle

VANETs Vehicular Ad-Hoc Networks

WAVE Wireless Access in Vehicular Environments

WPA Wi-Fi Protected Access

Chapter 1

Introduction

Modern cars have an increasing number of interconnected sensors and actuators that generate data that can be leveraged in an incredible set of applications. Examples of this are lane-keeping assistance and automatic parking.

The natural extension of this concept is to allow cars to share data amongst themselves and the internet. The goal is to construct a network connecting all cars, road infrastructure, sensors, and cloud servers, forming an Internet of Vehicles (IoV) [4]. An advancement like this can improve mobility, making it safer, faster, and more enjoyable. However, the conventional internet protocols cannot be used in this scenario given the high mobility of nodes. To address this concern researchers have been studying ways of making this Internet of Vehicles possible.

One of the main concerns is how to make the communications of this network secure against malicious users. If attackers can tamper with the data, the consequences can be more catastrophic than when dealing with normal internet communications. This is not a new problem, communicating in secrecy has been a subject of study for years now, and modern cryptography has already very good answers to it. We now have protocols and algorithms that can provide communications as secure as we want. The problem is that we often have a tradeoff between secure and fast communication.

Normally cryptographic protocols use two cryptographic techniques to assure secure communications: asymmetric and symmetric cryptography [5]. In symmetric settings, users share a key and can encrypt and decrypt messages with it. In asymmetric, there are two types of keys, a public and a private. If one encrypts a message with a public key, it can only be decrypted with the private key, and vice versa. This allows for flexible use cases and facilitates key sharing.

Although these systems are very successful, the rise of cloud computing, where users outsource data to third-party servers demands a new paradigm of cryptography. For example, if a user would like to perform some operations on data that is encrypted and stored in a public server, using only standard encryption techniques, the server would need to decrypt the data to perform the operation, needless to say, this is not secure.

In recent years, new solutions are being developed to make it possible to operate on encrypted data. The most known is fully homomorphic encryption [6] which allow users to perform all sort of operations

on data, such as multiplications, additions, logical operations, and so on. This is already the best we could hope for. However, the current homomorphic encryption solutions are still very slow and it is still impractical for most applications [7]. To work around this problem, we accept a reduction in functionality, hoping that it would increase speed.

Searchable Encryption [8] goes in that direction, by only performing searches on encrypted data and without leaking the users' query and the plaintext data. In these schemes, we can achieve faster solutions and since search is an important component in a lot of applications, this could already be practical in some scenarios. There is already plenty of searchable encryption schemes developed but because it is a recent field, some uncertainty still exists about how secure they are. It is not uncommon for a researcher to invent a new type of attack that breaks some schemes that were considered secure. As of today, there is no security standard defined so it is important to understand what are the security definitions, attacks, and limitations of a given scheme.

1.1 Problem

There is right now, a big concern for security in vehicular networks. The classical cryptographic protocols impose too big of a time and size overhead in the communication between nodes. This makes them unfeasible given the real-time nature of a lot of vehicular network applications.

New cryptographic protocols need to be developed for this setting, and new cryptographic techniques such as homomorphic encryption and searchable encryption can be used in them. The problem is that these paradigms are very new, and a lot of the schemes have not been tested in a vehicular network scenario. There are also a lot of security concerns about these technologies and they need to be carefully studied in that regard so that the developer understands the compromises they need to take.

1.2 Objectives

In this thesis, we will evaluate the current state of the art in searchable encryption. We intend to investigate some of the security definitions, what attacks can be made, and how expressive their queries can be. With that information, we will evaluate three Searchable Encryption schemes [1–3] and select the most suited for vehicular networks.

Then, we will evaluate said schemes in a vehicular network using a network simulator assess the overhead the scheme imposes on the network, and give an in-depth analysis of them. The objective is to provide a clear explanation of the schemes and point out their weakness and strengths so that the research community can have a better understanding of them.

1.3 Report Outline

In chapter 2 of this thesis, we will present the current state of the art in Searchable encryption, vehicular networks technologies, and simulators. In chapter 3 we will put explore current SE implementations

benchmarking them and finding functions that describe the time and space complexity of it. We will leverage that knowledge to simulate SE VANET protocols in chapter 4. In Chapter 5 we will present the conclusions and possible direction for future work.

Chapter 2

State of the Art

2.1 Searchable Encryption

Most modern applications store some kind of user's data in cloud servers. Because, normally, it is important to perform operations on that data (like search or delete) this kind of information is usually kept unencrypted in the servers. This makes the servers a very desirable place for attackers and has been one of the reasons that so many data leaks have happened. If data were kept encrypted, even if an attacker could get access to the files he would still need to decrypt the information, adding a new layer of security.

The challenge lies in how to perform operations on encrypted data. Many approaches are being explored such as homomorphic encryption and functional encryption. In this subsection, we will explore Searchable Encryption, a method that enables searches on encrypted data while aiming to reduce the leakage of information about the query to the server.

The general idea of all the schemes is the following: the client stores encrypted data in an untrusted server and for every entry generates an encrypted index S, containing all attributes of that entry. To perform searches, the client needs to generate a trapdoor T_Q with the attributes to be matched encrypted. The server can now perform for every data entry a test function that receives as input S, T_Q, and a key and return 1 if the document has the attributes being searched and 0 otherwise.

2.1.1 Cryptographic Definitions

We define the following mathematical ideas that are used in this work.

Multiplicative Cyclic Group

A $\mathbb{G}$ is a multiplicative cyclic group of order q and generator g if we can construct all his elements as g^i, with $i < q$ and $g, q, i \in \mathbb{Z}$.

Bilinear Map

Given two cyclic groups of the same order $\mathbb{G}$ and $\mathbb{G}_{\mathbb{T}}$ we define a function $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_{\mathbb{T}}$ that satisfies:

- (Bilinear) $\forall u, v \in \mathbb{G}, \forall a, b \in \mathbb{Z}, e(u^a, v^a) = e(u, v)^{ab}$
- (Non-degenerate) $\exists g \in \mathbb{G}$ such that $e(g, g)$ has order n in $\mathbb{G}_{\mathbb{T}}$

2.1.2 Public Key vs Private Key Schemes

Many Searchable Encryption schemes have been developed through the years and can be divided into two classes, Searchable Symmetric Encryption (SSE) first defined by Song in [8] and Searchable Public-key Encryption (SPE) defined by Boneh [9]. The difference between them lies in the way the data is searched. In SSE schemes the client and server share the same key, while in public key settings the server only knows the client's public key.

In this work, we will be focused on SPE schemes since they allow greater security mainly because the client does not need to share its key with an untrusted server.

2.1.3 General Algorithms

In subsection 1.1 we talked about the general idea behind the searchable encryption scheme. Now we will expand it by specifying the 4 algorithms that an SPE scheme has to perform.

- Key Generation: takes as input the system parameters and outputs the keys (sk, pk) for the client
- Keyword build: takes as input a keyword set $W = w_1, \ldots, w_n$ and pk and outputs K
- Trapdoor: takes as input a query $Q = q_1, \ldots, q_m$ and the secret key sK and produces a trapdoor T_Q
- Verification: takes as input the keyword for a given data entry K and for T_Q and returns true if a capability matches the keywords of a given document

2.1.4 Conjunctive Search vs Disjunctive Search

The first schemes developed such as [8], [9] did not allow for multiple keyword search, however in practical systems one must be able to search for multiple attributes and combine them. New schemes were developed to cope with that, and allow the attributes to be combined in two ways: Conjunctive and Disjunctive.

Conjunctive Search was achieved early on by Park in [1], which allows the user to search data entries that contain a specific combination of attributes.

Disjunctive search is a way of searching for data that contains at least one of the attributes searched for. It was introduced by Katz in [10] to achieve it the authors developed a new paradigm in public key

encryption called predicated encryption. With it, the user can associate cyphertexts with attributes, and combine them to perform a query.

2.1.5 Functional Encryption Schemes (FE)

Functional Encryption is a new paradigm in modern cryptography. In these types of schemes, the key holders do not recover the original message of the cyphertext, but a specific function of the plaintext and the secret key.

FE was first formally studied by Boneh, Sahai and Waters in [11]. They start by defining the concept of a functionality F, which is the set of functions that can be learned from a cyphertext. It is defined over (K, X) such that $F : K \times X \to \{0,1\}^*$ where K is the key space that contains the empty key ϵ and X is the plaintext space. They also defined the 4 algorithms that FE schemes must perform:

1. setup$(1^\lambda) \to (pk, msk)$
2. keygen$(msk, k) \to sk_k$
3. enc$(pk, x) \to c_x$
4. dec$(sk_k, c_x) \to F(k, x)$

The *setup* algorithm simply generates the public key, pk, and master secret key, msk. The *keygen* takes as input the msk and some element k in the key space and returns the secret key sk_k that is related to k. The encryption of a message x is performed by the *enc* algorithm using pk. The resulting cyphertext is defined as c_x. Finally, *dec* returns a function F of x and k and takes as input sk_k and c.

The information that cyphertext leaks (not secure) can be expressed as $F(\epsilon, c)$.

It is clear that we can use these types of schemes to construct searchable encryption, by correctly defining the functionality F and the plaintext x. We start by building x as (keywords, m) and a predicate P that evaluates a logical expression, such that $P : K \times I \to \{0,1\}$, where I is the keyword space.

In that way, if the keywords of a document match the query expressed by P, we are able to retrieve the message, otherwise, we do not. We have:

$$F(k,x) = \begin{cases} m, & \text{if } P(k, keywords) = \text{true} \\ \perp, & \text{if } P(k, keywords) = \text{false} \end{cases} \tag{2.1}$$

From functional encryption, we can define a subset of other schemes. We proceed to present the ones that are most common in SE schemes.

Identity Based Encryption (IBE):

IBE schemes were first defined by Shamir in [12] with the intention of avoiding key exchange between parties. In it, each user has a string as public identity id and a secret key sk_{id} that corresponds to it. The sender encrypts a message to that particular id using pk and only the sk_{id} can decrypt it. However, this type of scheme still needs the use of a central trusted authority, the Public Key Generator (PKG).

Through equation 2.1 we can better describe an IBE system by modifying the predicate P to the following:

$$F(k,x) = \begin{cases} m, & \text{if } id = keyword \\ \perp, & \text{if } id \neq keyword \end{cases} \tag{2.2}$$

These types of schemes leak information about the id, we have $F(\epsilon, c) = id$.

The first SPE ever created [9] was constructed through an IBE scheme, where instead of id each message had a set of keywords, encrypted separately. The user could then search for only one keyword.

Attributed-Based Encryption (ABE):

Attributed-Based Encryption was first introduced by Sahai and Waters in [13]. The authors developed a fuzzy IBE scheme where the identity of the user is comprised, not of a string, but of a set of descriptive attributes. The documents were also associated with attributes when encrypted and could only be decrypted if at least k attributes matched with the user.

This notion evolved and Goyal, Pandey, Sahai, and Waters [14] introduced the concept of access structures, logical expressions that can combine the attribute values in a more complex way (using **AND** and **OR** gates) to determine if the decryption can be performed. The authors also formulate the following definitions of ABE schemes:

- KP-ABE: Keys are associated with access control policies and cyphertexts are associated with attribute set
- CP-ABE: Keys are associated with attribute set and cyphertexts are associated with access control policies, it was only implemented by Waters in [15]

The intention of this first schemes was to provide a fine grain access control to the data, where users with different permissions could decrypt different documents in the database. However, we can extend ABE to a Searchable Encryption scenario where instead of matching access control policies and attributes we match a query to a set of keywords following the same equation in 2.1. The big problem in ABE schemes is that the query information is exposed to the server.

Predicate Encryption (PE):

Predicate Encryption (PE) was first introduced by Katz in [10]. PE was created to solve the big problem of the ABE schemes, the attributes are not hidden from the server.

Similar to ABE schemes, we associate secret keys with predicates and cyphertexts with attributes. The decryption can only be successful if the predicates of the secret key match the attributes of the cyphertext. The main advantage of this setup is that the attributes are hidden from the server if the query is evaluated as False.

The general flow of an ABE and PE system being used in a Searchable Encryption scenario is shown in the image below:

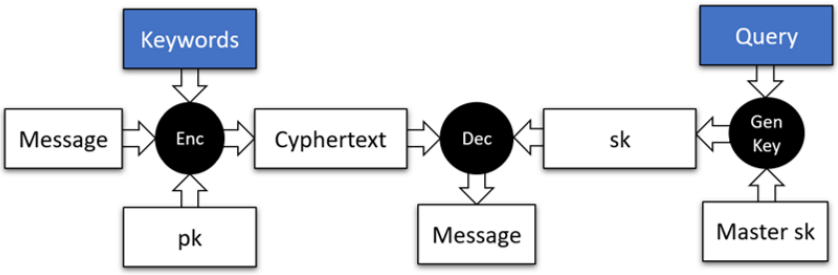


Figure 2.1: Successful decryption in PE or ABE scheme, where the query matches the keywords of the document

2.1.6 Taxonomy of Public Key Schemes

There are several ways of building a public key searchable encryption. Zhou in [16] defined six of them, which are the following:

Public Key Infrastructure (PKI)

Public Key Infrastructure (PKI) is the classical public key structure. The senders generate encrypted data using the receiver's public key and upload it to the server. The authenticity of the key is verified with the use of certificates. The receiver generates trapdoors using the secret key and sends them to the server, which has the ability to check each data entry. These schemes generate heavy burdens in certificate management and key distributions.

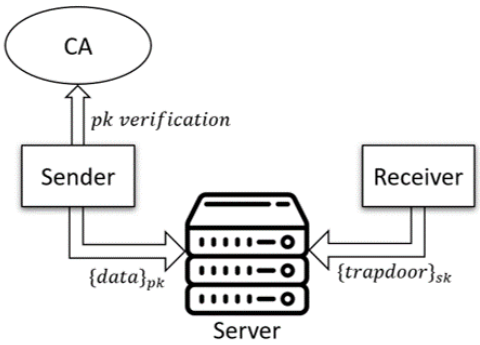


Figure 2.2: General PKI scheme, assuming key distribution has already happened

Functional Schemes

Functional schemes are constructed using a functional encryption paradigm (described in subsubsection 1.1.5), in it the receiver needs new private keys for each query and the generation is normally controlled by a central entity, the public key generator (PKG).

The most commons functional schemes for SE are IBE, ABE, PE. IBE allows for one keyword verification. ABE and PE allow for more complex queries that can be expressed by logical expressions, the difference between them being that PE hides the query of the user from the server.

In these types of schemes the sender does not need to check the receiver's public key

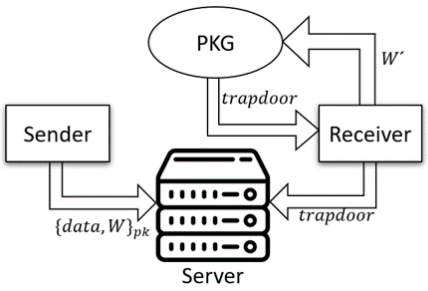


Figure 2.3: General FE scheme

Certificateless Encryption (CLE):

In IBE schemes the sender does not need to verify the user's public key identity, but this gives rise to the need for a central entity that generates all keys and can decrypt all messages. Security wise is a big problem and is what Certificateless Encryption tries to improve on. To achieve it, the PKG generates only a partial private key ad the receiver uses this information to generate the full private key. In this way, the PKG does not know the user's private keys and cannot decrypt all the messages that flow through the system.

The first CLE for searchable encryption scheme was created by Peng in [17] and it was able to perform a conjunctive search. All the public keys and secret keys are generated using a partial private key generated by the PKG and a private secret of the user and there is no need to verify the authenticity of the public keys.

In it, the receiver would encrypt a message and a set of keywords using the public key of the server and the receiver. The receiver can generate trapdoors using its secret key and the public key of the server. The server can test the documents using the trapdoor and the secret key of the server.

The scheme in [17] is not ideal because does not follow the ideas of classical public key cryptography, for instance when encrypting a message, ideally, we would only use the public key of the receiver, but in this case, we need the server's public key as well.

Zheng in [18], constructed a scheme more in line with what is expected from a public key construction and the algorithms defined in subsection 1.1.3, but it needs a secure channel and it can only search for single keywords. The general idea of the scheme can be seen in the image below:

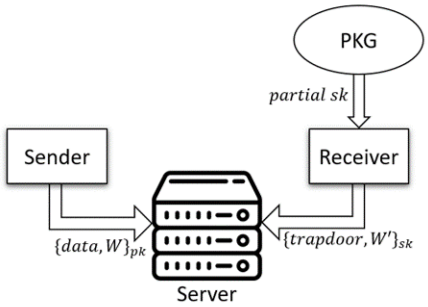


Figure 2.4: General CLE scheme

2.1.7 Security Concerns

Security Proofs

When a new scheme is suggested, the authors have to provide proof of its security. This is normally done in a game-based approach, where a polynomial-time adversary will play a game with the challenger that will use the encryption scheme being tested. The game being played defines the level of security of the scheme.

Modern cryptography is based upon computational hardness assumptions, that is the hypothesis that a certain problem cannot be solved in polynomial time. What the researchers do to prove that a scheme is secure is construct a proof by reduction. Showing that if an adversary breaks the schemes it is also solving a hard problem. To perform those proofs we use the following framework:

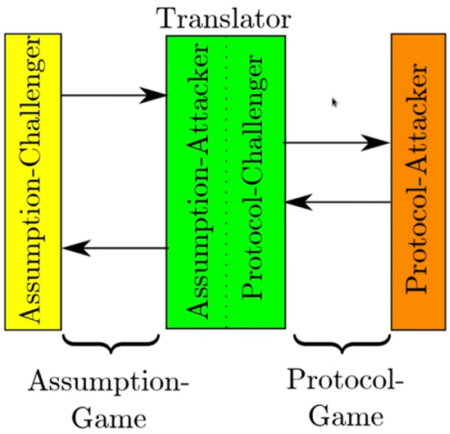


Figure 2.5: Proof by reduction

In figure 2.5 we can see how the proof is constructed. The left side is the computationally hard problem to solve and the right side is the security game that is testing the scheme.

Hard Problems in SE

Some of the hard problems normally used in this field are the following:

- **Decisional Diffie-Hellman (DDH):**

  For a given multiplicative cyclic group $\mathbb{G}$ of order q and generator g and given $a, b \in \mathbb{Z}_q$ the value of g^{ab} it is indistinguishable from g^c where $c \in \mathbb{Z}_q$.

- **Bilinear Diffie-Hellman (BDH):**

  For a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ and $P \in \mathbb{G}_1$ and $a, b, c \in \mathbb{Z}$ it is hard to compute $\hat{e}(P, P)^{abc}$

Security Models in Public Key Encryption

The security model adopted by the authors of the scheme defines the game that is going to be played and consequently the level of security. In standard cryptography, it is enough to prove that the scheme

is indistinguishable under a chosen-plaintext attack (IND-CPA), where an adversary cannot distinguish the encryption of two distinct messages.

However, in the SPE scenario, we must protect the output of the algorithm that generates the indexes. This means the attacker cannot infer what are the keywords of a document if they received them encrypted. The most common security model adopted in SPE was defined by Boneh in [9] and improved on [1], it is called Indistinguishably of Ciphertext Chosen Keyword Attacks (IND-CKA). The game is played between a challenger $\mathbb{A}$ and an adversary $\mathbb{C}$

1. **Setup:** The challenger $\mathbb{C}$ runs the algorithm KeyGen$(1^\lambda) \rightarrow (pk, sk)$ and gives the public key pk to the attacker
2. **Phase 1:** The attacker $\mathbb{A}$ adaptively generates trapdoors for a query Q_i of its choice. Trapdoor$(pk, Q_i) \rightarrow T_Q i$.
3. **Challenge:** The adversary sends to the challenger two queries Q_0, Q_1 that have not been seen in the phase 1. The challenger picks a random bit $\beta \in \{0, 1\}$ and sets $S_\beta =$ PECK(pk, W_β) and sends to the adversary.
4. **Phase 2:** The adversary generates more trapdoors for all queries that are not Q_0, Q_1
5. **Guess:** In the end the adversary outputs the guess $\beta^{'}$ and it wins if $\beta^{'} = \beta$

If the adversary can guess the right β with a probability higher than 0.5 the scheme is not secure.

Most of the proofs are done in the Standard Model or the Random Oracle Model. In the Standard Model, the adversary is only limited by the amount of time and computational power available. Sometimes the proof might be too difficult under the Standard Model so we relax some of the assumptions, the Random Oracle assumes the existence of an ideal collision-free hash function that does not exist in the real world. Because of this, proofs under this model are redeemed less secure than the ones done in the Standard Model.

Types of attacks in SE

- **Keyword Guessing Attack**

  Most of the keyword field names are shared amongst different types of applications, for example in a database of clients it is reasonable to assume that one of the keyword fields will be ”Name” or something very similar. This happens because normally the set of possible keywords is small, meaning it has low entropy.

  Knowing this, an attacker can simply launch a dictionary attack over a captured trapdoor from a client. It generates trapdoors for a set of possible keywords and tries to match them. with the captured trapdoor. These attacks result in the attacker discovering the name of the keyword field of the query. Byun introduced this idea in [19] and was able to successfully attack some of the schemes developed at the time.

- **File Injection Attack**

  Zhang, Yupeng and Katz in [20] introduced the file injection attack, where if the server can trick the client to insert certain files in the database he can know the keywords in every query.

  They achieve that by carefully selecting the keywords in every injected file. Then when the client performs a query they evaluate the infected files that were selected in that query. Because they know the keywords in these files they can construct back the query. With this kind of attack the authors of [20] only need to insert $log(|K|)$ files, where $|K|$ is the number of keywords.

  For this attack to be successful the server must be able to locate the injected files once the client uploads them. This might not be possible because the client encrypts every file before uploading. Also, the attacker does not learn the actual names of the keywords, he can only evaluate which keyword fields are in each query.

POST Quantum in SE

Some of the hardness assumptions that some cryptographic algorithms are built upon can be broken by quantum computers. There are already algorithms for quantum computers that can factor numbers and compute discrete logarithms in polynomial time as seen in [21], this means that schemes based upon DDH or BDH assumptions are not safe in a world in which quantum computers exist.

This gives rise to the necessity of finding another type of hard problem that has not been solved yet. One of the proposed solutions is Lattice-Based cryptography [22] that introduces a new set of interesting problems such as Learning with Errors.

2.1.8 Schemes of interest

In this subsection, we will analyze some of the schemes that have been developed. Boneh developed the first public key encryption scheme in [9], it enabled single keyword search and it was vulnerable to file injection attacks. All cryptographic SE schemes have a trade-off between security and performance. Our goal here is to evaluate it and chose the one that fits us best.

Since the purpose of this work is to test already developed schemes that have been developed recently, a search was made using google scholar to find scheme implementations. The parameters used to evaluate the schemes were the following:

- **Query type**: what kind of queries are supported, they can be Boolean or range
- **Keywords**: schemes can support multiple or single keyword
- **Encryption**: SPE or SSE
- **Post Quantum**: If can resist attacks from quantum computers

The results are summarized in Table 2.1

For this particular setting we are looking for a public key scheme, as it enables multiple writers settings, that allows for maximum query expressiveness, meaning multiple keywords that can be searched

Table 2.1: Schemes Analyzed

Scheme	Encryption	Keyword	Query Type	Post Quantum
Sophos [23]	SSE	Single	Boolean	x
Diana [24]	SSE	Single	Ranged, Boolean	x
Janus [24]	SSE	Single	Boolean	x
Tethys [25]	SSE	Single	Boolean	x
Pluto [25]	SSE	Single	Boolean	x
Wang *et al.* [26]	SSE	Multiple	Conjunctive	✓
Xu *et al.* [27]	SPE	Single	Boolean	✓
Behnia [28]	SPE	Single	Boolean	✓
Boneh [1]	SPE	Single	Conjunctive	x
Chui *et al.* [2]	SPE	Multiple	Conjunctive and Disjunctive	x
Tseng *et al.* [3]	SPE	Multiple	Conjunctive and Disjunctive	x
Hao *et al.* [29]	SPE	Multiple	Conjunctive and Disjunctive	x
Meng *et al.* [30]	SPE	Multiple	Conjunctive and Disjunctive	x

for using logical operations as complex as possible. If possible you are looking for a scheme that is quantum secure and leak as few information as possible about the users query and key.

Looking at Table 2.1 we can see that the scheme created by Wang [26] achieves most of our demands. It is the only quantum secure scheme analyzed that can perform a conjunctive search. It is also secure against file injection attacks as the search pattern is hidden. However, it is an SSE scheme and requires to servers to perform a search. This extra layer of communication overhead can lead the system to be too slow.

The schemes [2, 3, 29, 30] can achieve the highest degree of query expressiveness amongst the analyzed schemes, supporting any combinations of keywords using AND and OR logic gates. Schemes [29, 30] provide also access control, where data users can only search in a limited set of documents defined by the access policy that can be set by the data owner. This extra functionality can be useful but adds on extra overhead that slows done this schemes when compared to [2, 3].

The scheme [2] achieves is faster than Tseng [29] and the authors in [3] claim that their scheme is even faster. However they did not present yet, a performance comparison to support these claims.

2.2 Vehicular Networks

Modern cars have an increasing number of interconnected sensors and actuators. These new technologies already allow for some sophisticated applications that were impossible some years ago, such as automatic parking and lane-keeping assistant. The car of today is a complex system of interconnected components, that relies on different networking technologies, such as Controller Area Network (CAN) or Local Interconnect Network (LIN) buses, for *intra-vehicle* communication.

Researchers soon realized that a new set of applications could be created by connecting vehicles together with road infrastructure using wireless technology. This idea gained a lot of strength with the rise of Internet of Things (IoT) which allows for all kinds of sensors, actuators and devices (commonly

called things) to be connected to the internet. It is estimated that 46 billion things will be connected to the internet by the end of 2021 [31]. So we can see how appealing it is to interconnect vehicles, not only for them to exchange information, but so they can gain access to a new set of services and devices that are spread across the internet.

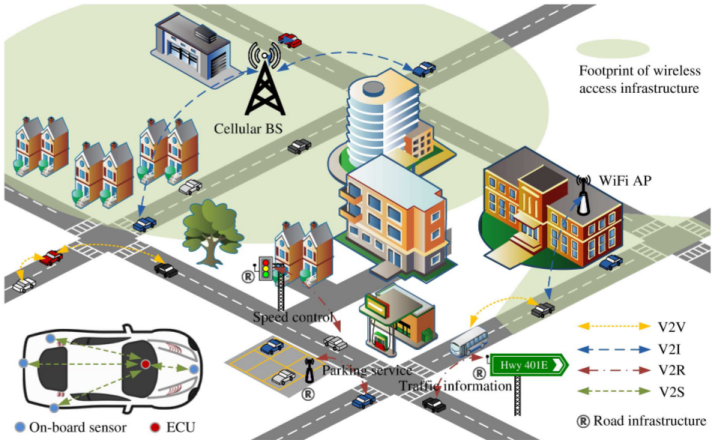


Figure 2.6: Vehicular Network Infrastructure [32]

Figure 2.6 shows a common setting in a Vehicular Network where 4 types of communications are defined:

- V2V: communication of vehicle to vehicle
- V2I: communication of vehicle to the internet
- V2R: communication of vehicle with road infrastructure
- V2S: communication of vehicle to sensor on-board

2.2.1 Vehicular Ad-Hoc Networks (VANETs)

In order to enable V2V and V2I communications researchers started working on top of the concept of a Mobile Ad-Hoc Network (MANET). MANETs are decentralized wireless networks that do not depend on external network infrastructure. Instead, each node acts as router forwarding messages according to the protocol established. VANETs have an extra complication because the network topology is highly dynamic. The nodes are always entering and leaving the network as they are moving at high speed.

The main networking protocol being used for VANETs is IEEE 1609, commonly known as WAVE (Wireless Access in Vehicular Environments), shown in figure 2.7. It uses IEEE 802.11p standard for dedicated short-range communication (DSRC) for the MAC and Physical layer. Higher networking layers are implemented by IEEE 1609.3. WAVE also has a management pane.

Normally there are two types of DSRC devices: the onboard units (OBUs) which are the mobile nodes of the network that represent the vehicles and roadside units (RSUs) which are stationary nodes connected to the backbone of the internet. In this context, V2V is performed by OBUs communicating in an Ad-hoc manner and V2R is performed between OBUs and RSUs.

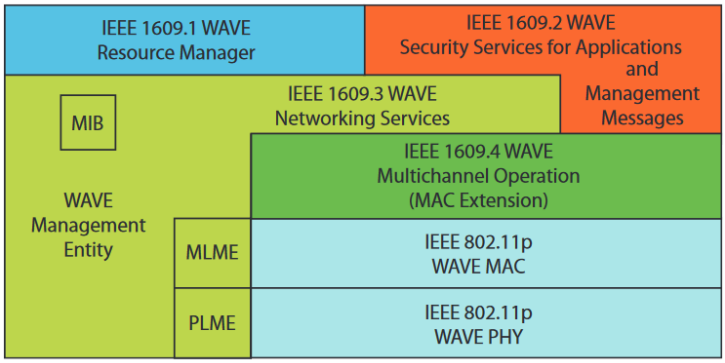


Figure 2.7: WAVE architecture [33]

The WAVE model appears to work fine in low-congestion scenarios. However, studies have shown that in traffic situations the performance of this architecture reduces greatly [33].

2.2.2 Internet of Vehicles

The new era of the Internet of Things is expanding the notion of VANETs to a broader concept of Internet of Vehicles (IoV). Now the interest is not only in V2V and V2I communication but V2X (vehicle to anything) where a vehicle can communicate to all sorts of nodes. On top of that, modern cars have all sorts of sensors and are generating a lot of data that has to be dealt with by the network.

Researchers have not decided on the architecture of this new network yet, but the main idea lies in having a layer of data collection that encapsulates the sensors and vehicles, a networking layer that combines WAVE and 5G for V2X and Ethernet for the (RSUs) and a cloud layer that has servers and applications that operate on that data. The challenge of this topic is how to combine the different networking technologies, deciding, for example, what technology should a node use to transmit information.

Figure 2.8 is an example of how this architecture can be put forward. First for the datalink and physical layer we have a set of protocols designed for nodes with mobility like WAVE and LTE. For the upper layers, we can already use some well-known protocols like IP and HTTP.

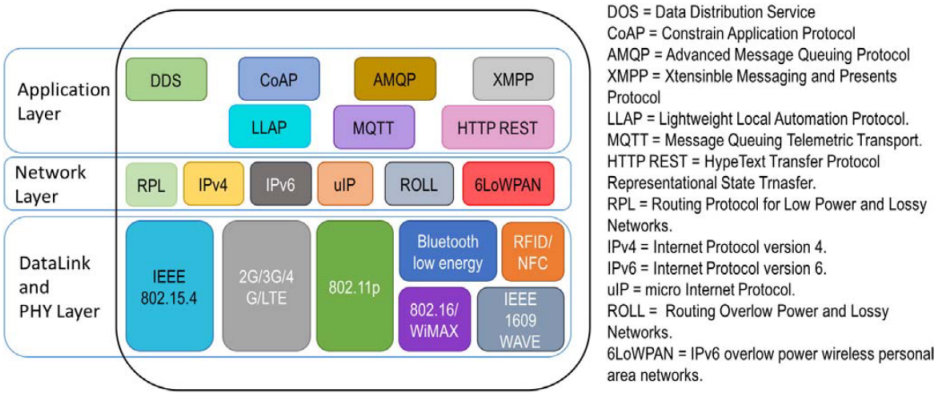


Figure 2.8: Example of IoV architecture exemplified in [4]

2.3 Simulator

Before implementing a new system in the real world it is a good idea to simulate it first. The main reason is that normally it is easier and cheaper to implement and test in a simulation environment, and if any errors in the system occur we can correct them and rerun the simulation.

The idea of this work is to evaluate the performance of some algorithms in the vehicular networks scenario, as described in chapter 2.2. To achieve that we need a tool that can mimic this kind of network topology and protocols and simulate the operations that the nodes are performing.

2.3.1 Simulator Evaluation

The website `https://networksimulationtools.com` provides information about projects developed in network simulators. Table 2.2 was constructed with data provided by this website and can provide a clear picture of what simulators are being currently used for networking projects.

Table 2.2: Most Used Network Simulators

Simulator	PHD	Master Thesis	Total
NS3	98	119	217
OMNET++	103	95	198
NS2	75	117	192
MININET	71	62	133
GNS3	35	89	124

A simulator for Vehicular Networks is different from a regular network simulator because it needs to support node movement. On top of that, the protocols being used in the communication process are different from the usual ones (Ethernet, Wifi). So we need to evaluate if the following simulators can satisfy these requirements.

Mobility

A network simulator can provide mobility in two ways:

1. The simulator controls the movement of the nodes natively by executing functions that change their position and speed
2. The simulator relies on an external source, such as a file or a traffic simulator, for managing the movement of the nodes

It is simpler to work in a simulator of the first kind because all the simulation process is self-contained in one software. However, it is hard to generate realistic movement in the network simulator alone, because the mobility aspect is not the sole focus of the simulator and the mobility functions tend to be very simple.

For that reason researchers tend to prefer to work with external sources, in [34] the authors show three ways in which the simulator can be combined with external sources, they are following:

1. **One Way Decoupled**: The network simulator uses an external trace file that handles the movement of the nodes

2. **Two-way loosely coupled**: Two independent software, normally one traffic generator and one network generator, are combined using a controller software.

3. **Two-way tightly coupled**: Single integrated software that handles both aspects of the simulation

The decoupled strategy works best if the functions being performed by the nodes will not change their movement and speed. This allows for constructing simpler simulations without losing realism.

On the contrary, we couple traffic generators and simulators if the nodes are executing applications that will change their positions. An example of such an application is a traffic application where nodes would communicate congestion information among themselves. Ideally, we would like to see the nodes change their curse when they find faster routes. This kind of interaction can only be achieved with coupling.

The loosely coupled is more flexible and we can even try different traffic simulator software without changing the network simulator code, however, the tightly coupled allows for faster simulations.

NS2 and NS3 we can control movement with native functions and can load trace files for mobility information, they can also be found tightly coupled with a traffic simulator named SUMO in the software, TraNS and iTETRIS.

OMNet++ lacks movement simulation but it is combined with SUMO with the software VEINS to achieve movement. To our knowledge it is not practical to simulate movement in MININET and GNS3, this can be verified by the lack of research on Internet of Vehicles being done with these tools.

Communication

For the networking part NS2, NS3 and Veins have some sort of implementation of the standard communication technologies being used in vehicular networks. All have modules that implement WAVE, but only Veins has built-in 5G support. However, we can find libraries developed by researchers that implement 5G in NS2 and NS3.

Conclusion

Since only NS2, NS3 and Veins have mobility support we need to choose a simulator amongst them. Even though there is still a lot of research being done on NS2, it is old software that has stopped being supported by the company that developed it.

In this project, the path that the vehicles take does not depend on the algorithm being run by the nodes. This means that the network simulator does not need to be coupled with a traffic simulator. Veins is a powerful tool but this extra functionality may lead to unnecessary complexity. Also, NS3 offers simple functions that control the movement of the nodes, this will help the development process as it allows to generate simpler builds in the early phases of the project.

For those reasons, we decide to use NS3 in this project. It is capable of modeling the two communication technologies being used in vehicular networks, it has good mobility support and has a big community of users, being the most used network simulator as seen in Table 2.2.

2.3.2 NS3

The simulator is structured around the idea of *Nodes*, which are an abstraction for computer devices. Nodes run *Applications*, and can send packets to other nodes using *Channels* attached to *NetDevices*. The size of the packets depends on the protocols being used by the node. The general structure can be seen on Figure 2.9.

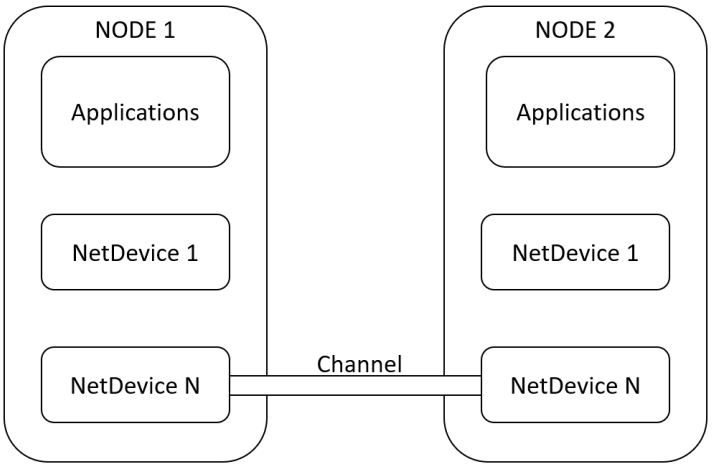


Figure 2.9: NS3 structure

The tutorial guide [35] provided by the NS3 community gives a deeper understanding of the components in Figure 2.9

- **Node:** A NS3 class that defines an abstraction for a computing device. It holds applications, networking devices, and protocol stacks
- **Application:** an NS3 class that simulates a user program. It can be specialized by the developer to mimic the real-world behavior of the application. An example of an application is the built-in `UdpEchoClientApplication` that mimics a client that echoes UDP messages being sent to it.
- **Channel:** is the NS3 class that models the physical media over which the data packets are being sent. An example of channel class is the `WifiChannel` that simulates the Wifi behavior.
- **NetDevice:** is the NS3 class that simulates the hardware peripherals and software drivers that enable the Node to communicate via a Channel

For communication between nodes, NS3 has libraries that implement WAVE and 5G protocols. For WAVE there is an official implementation that allows for the creation of Channels and NetDevices that obey the MAC/PHY layers. They provide two options, a simpler one for single channels and a more

complex one for multi-channel operation mode. For 5G communications, there is a library developed in [36] on top of the LTE module.

Chapter 3

Searchable Encryption Implementations

In this chapter, we will study the implementations of 3 Searchable Encryption schemes. Two of them are ABE schemes and were implemented by [3], however some changes needed to be done to test them in a more realistic scenario. The authors of these schemes claim to offer conjunctive and disjunctive searches, but we will prove that in those schemes the disjunctive searches are reduced to multiple conjunctive ones.

Leveraging that knowledge we will implement the conjunctive scheme [1] from scratch and modify it to allow disjunctive searches. In the end, we are going to benchmark the implementations and find functions that describe the time and space complexity of each algorithm.

3.1 General Framework For Testing

All the schemes analyzed were implemented in Python using a special framework for developing cryptosystems, the Charm Framework [37]. It allows for fast implementation while maintaining a good balance with performance. It also provides tools for benchmarking that facilitate the measures in the time and space domain.

All schemes were implemented as implicit abstract classes that offer the following methods:

- __init__: using the desired paring group as argument
- setup: with no arguments returns the master secret key and master public key
- s_keygen: with no arguments, returns a public and a secret key pair
- encrypt: with the public key and the keyword values as arguments, returns the encrypted cypher-text
- keygen: needs the master secret key, the public key, the attribute values being searched for, and a policy matrix that encodes the boolean query. It returns the trapdoor for that given query

- decrypt: receives a public key, a secret key, a cyphertext, a trapdoor and the combination of attributes being searched for, it returns a boolean.

With that in mind, we wrote a script that tests and benchmarks a generic crypto scheme. It is comprised of the following:

- A function that generates a database with random values for a given number of columns and lines
- A function that constructs random queries for a given database
- A function that runs a given query in a Searchable encryption scheme and in plain text and compares the results, useful to check the correctness of the implementation
- A benchmark function, that measures the time a function took and the memory usage of an object

The random generator of the simulations is seeded by an input parameter and each measure is done three times with different seeds. The results are written to text files that are visualized in python using data visualization techniques to understand them.

All simulations were done in a computer using the AMD 6212 processor, with 8 cores and a clock of 2.6 GHz with 32 GB of RAM using Debian GNU/Linux.

3.1.1 Simulation Parameters

In a searchable encryption scenario, we want to test how the scheme will behave when we change a set of different parameters. The ones we consider more relevant were: the number of keywords, number of lines, size of the query, and number of disjunctions in a query (number of ORs). So we generate several encrypted databases with the following parameters doing all possible combinations of 1, 5, 10, 100 lines and 1, 3, 5, 10, 30, 60 columns. We then generate a random query with a size between 1 and the number of columns / 2, determined by a random number generator. Through that query, we can determine the number of disjunctions just by counting the number of ORs.

Some of the algorithms being tested vary linearly with the number of lines, for those we carry out a normalization to remove the dependencies in the number of lines. This normalization is the division between the result value and the number of lines. In this way, we can evaluate better other relations in the data.

3.1.2 Techniques to analyze outputs

We proceed to use regression techniques to fit our results into a function that approximates well the results. We will use a linear model in the form of:

$$f(x) = w^T \Phi(x) \tag{3.1}$$

Where $x = (x_1, \ldots, x_N)$ is our data, $x_i = (x_i^{(1)} \ldots x_i^{(m)})$ is a data entry with m features and $\Phi(x)$ is a matrix with N rows with row i being equal to $\phi(x_i)$ that can be any function of the type $\phi : \mathbb{R}^m \to \mathbb{R}^n$. By

analyzing the data and the scheme we can come up with the function ϕ that represents well the relations in the data. In the training step, we focus on finding the best weights w.

First, we divide the data into two datasets: training and validation. With the train data, we do a linear regression and find a function $f(x)$ that bests describes our data. We predict values using the features in the validation set and compare them with the real value. We compute the $R2$ score that will give a measure of how well the model describes the data.

3.2 Consideration on Elliptic Curves and Pairing

The creation of RSA [38] and the Diffie Hellman [39] key exchange gave rise to a new era in cryptography, where one could cipher and decypher based on mathematical statements that could be proved to be correct. However new algorithms, such as the Quadratic Sieve, were developed to reduce the complexity of computing the prime factorization of a number. That led to an increment in the key sizes to achieve the same level of security.

One of the solutions to this problem is to use a different mathematical construction, the elliptic curves. We define a group $\mathbb{G}$ whose elements are of the form (x, y) and obey the following equation $y^3 = x^2 + ax + b$, modulus p, where p is a prime number. In this group, we can add and subtract two points P and G and we can also multiply a point by an integer n, such that it is hard to find out n only knowing nP and P.

Because elliptic curve cryptography seems to be based on a harder problem it can use smaller keys and cyphertexts to achieve the same level of security as RSA.

In some elliptic curves we can define a bilinear map defined as a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_\mathbb{T}$ with the following properties:

1. $e(P, Q + R) = e(P, Q) * e(P, R)$
2. $e(P + S, Q) = e(P, Q) * e(S, Q)$

These maps are called pairings and elliptic curves that allow them are called Pairing Curves. There are three types of pairings:

1. Type 1: $\mathbb{G}_1 = \mathbb{G}_2$
2. Type 2: $\mathbb{G}_1 \neq \mathbb{G}_2$, but there is a map between $\mathbb{G}_1$ and $\mathbb{G}_2$
3. Type 3: $\mathbb{G}_1 \neq \mathbb{G}_2$

3.2.1 Pairing Curves

The schemes mentioned in this chapter were implemented using pairing curves. The framework Charm [37] offers support to some of them such as BN (Barreto-Naehrig), MNT (Miyaji, Nakabayashi and Takano) and SS (Stress–strain). BN and MNT are type 3 and SS is type 1, for the sake of being as general as possible, we decide to choose a curve of type 3.

We choose the BN curve because it is more frequently used in practical implementations. The size chosen is 254 bits which offers around 100 bits of security according to a recent draft published by the IETF (Internet Engineering Task Force) [1].

The computation times measured in this chapter depend strongly on the pairing curve used. In table 3.1 we measure the computation times of the operations in a pairing curve. For every measure, we did 1000 operations and average the result.

Table 3.1: Computation times in milliseconds in the pairing curves

Curve	Pairing	Group multiplication	Scalar and Group Element multiplication
BN254	54.197	0.013	1.721
SS512	1.819	0.016	2.076
MNT224	9.385	0.012	1.398

3.3 The ABE approach

As it was already described in 2.1.5, attribute-based encryption allows for attributes $a_1, \ldots, a_n$ and policies $P : a_1 \times \ldots \times a_n \rightarrow \{0, 1\}$ to be embedded in the encryption and decryption process so that, information can only be recovered if $P(a_1, \ldots, a_n) = 1$.

Many authors started to construct their searchable encryption schemes based on ABE schemes. They do that by associating the attributes with the ciphertext and the policy with the key in what is called a KP-ABE. It is easy to see that the policy function is the one that defines how expressive can a query be.

The schemes that we choose to analyze construct their policy function based on linear secret sharing schemes (LSSS), an idea that was first used in an ABE scheme in [40], and allows for conjunctive and disjunctive queries on the data.

3.3.1 LSSS

We start by defining LSSS as in [40]. In a Linear Secret Sharing Scheme, a secret s is shared over a set of n parties P, such that:

1. The shares of each party form a vector in $\mathbb{Z}_p$.
2. Each party has a corresponding vector in the form $\rho = (\rho_1, \ldots \rho_n)$. We can then form a matrix M where the row i is the vector ρ from the ith party. We generate the share by selecting a vector $v = (s, r_1, \ldots, r_n)$ where $r_1, \ldots, r_n$ are random numbers, $(Mv)_i$ is the share of the ith party

A set of parties $P_1, \ldots, P_N$ is part of the authorized set if and only if there exists $w_1, \ldots w_N \in \mathbb{Z}_p$ such that $\sum^N w_i * \rho_i = (1, 0, \ldots, 0)$.

[1] https://www.ietf.org/archive/id/draft-irtf-cfrg-pairing-friendly-curves-08.html

The general idea behind the ABE schemes that use LSSS is to map attributes into parties. To generate a trapdoor for a query we build a matrix M that will generate shares that build the authorized set based on the desired attributes.

But we can already see how this can be problematic in a Searchable Encryption scenario, to perform the verification of a trapdoor and a cyphertext, all the schemes analyzed pass the $w = (w_1, \ldots, w_n)$ as plaintext to the server. This means that the server can reconstruct part of the query, and find out which attributes are being searched for, just by looking for $w_i = 1$.

Converting from Boolean Formulas to LSSS Matrices

A boolean formula can be encoded in the matrix M using the following algorithm provided in [40]:

1. Convert the boolean formula into an access tree, that has **AND**s and **OR**s as its nodes and attribute values as its leafs
2. We label the root node of vector $v = 1$ and set the global counter $c = 1$
3. We go through the tree in Breadth-first, exploring all nodes of a given level before moving to the next level. If the node n_i is an OR we label its children with $n_i.v$. If n_i is an AND we pad $n_i.v$ with zeros in the end until it has length c. We label one children with $n_i.v|1$ and the other with $(0, \ldots, 0)| - 1$ where $(0, \ldots, 0)$ has size c, in the end increment c.
4. At last we pad all vectors with zeros at the end until they have length c.

We can see by this algorithm that an **OR** gate adds a new dimension to the solution space, so when adding a disjunction, the equation $Mw = (1, 0, \ldots, 0)$ does not have a single solution anymore. Moreover, there is at least one extra vector w for every **OR** gate in the boolean formula. In order to evaluate if a set of parties belongs to the authorized set we need to test all possible w.

To realize this LSSS based schemes a set of Python functions were implemented:

- A function that converts a boolean query to an LSSS matrix M
- A function computes the weights w that solve the system $Mw = (1, 0, \ldots, 0)$
- A function that extracts all keywords and their values from the boolean query

3.3.2 Leakage

Anonymity Notion

As it was explained in chapter 2.1, the ABE attributes are not hidden from the server. Although this might be reasonable in some applications in a SE scenario, the server should not be able to know the query of the client.

To get around that, what is normally done is to divide the keywords inside the query into Keyword names and Keyword Values. The information on the keyword names is exposed to the server while the

keyword values remain hidden. For example, in queries such as *"Name = Afonso and Age = 23"*, the server would only know that a search for Name and Age was made, but Afonso and 23 are kept secret.

This is a weaker notion of anonymity, but the sensitive part of the query is still protected and makes the ABE schemes possible.

LSSS Limitations

In SE schemes the LSSS approach for testing a query has a big flaw, that researchers seem to disregard. We cannot evaluate a disjunctive query with a single execution of the Test algorithm. This happens because in a disjunction we will have at least two sets of authorized parties which need to be tested separately.

We also have to leak extra information. Because the server does not know the shares of each party (that would leak the keyword values of a cyphertext) the client needs to provide the appropriate weights for every keyword. With that information, the server can reconstruct the query operations between keywords.

Combining these two types of leakages, in a query such as *"Name = Afonso and Age = 23 or Job = Unemployed"* the server would know: *"Name = ? and Age = ? or Job = ?"*

3.3.3 Searchable Encryption ABE - CW Scheme

Given this brief introduction to some of the instruments used to build an ABE scheme, we proceed to analyze the one described in [2]. From now on, we will mention it as the CW Scheme. The algorithms that comprise it are defined as follows:

- *Setup:* chooses a group G of prime order p, a generator g and random group elements $u, h, w \in G$. Choose randomly $\alpha, d_1, d_2, d_3, d_4 \in \mathbb{Z}_p$. Pick a hash function H that maps elements of G_1 to elements in G. Publish as public parameters $pars = (H, g, u, h, w, g_1, g_2, g_3, g_4, e(g,g)^\alpha)$ and as master secret key $msk = (\alpha, d_1, d_2, d_3, d_4)$.

- *KeyGen:* Take as input $pars$ and randomly chooses $\gamma \in \mathbb{Z}_p^*$. Outputs, $(pk_s, sk_s) = (g^\gamma, \gamma)$

- *Trapdoor:* takes as input, $pars$, pk_s, msk, the LSSS matrix M that encodes the query with dimension $l \times n$ and the attribute values $W_1, \ldots, W_n$. It randomly chooses a vector $y = (\alpha, y_2, \ldots, y_n)$,

$r, r', t_{1,1}, t_{1,2}, \ldots t_{l,1}, t_{l,2} \in \mathbb{Z}_l$. It computes for $i \in 1, \ldots l$:

$$
\begin{aligned}
v_i &= M_i y \\
T &= g^r \\
T' &= g^{r'} \\
T_{i,1} &= g^{v_i} w^{d_1 d_2 t_{i,1} + d_3 d_4 t_{i,2}} \\
T_{i,2} &= H(e(pk_s, T')^r) g^{d_1 d_2 t_{i,1} + d_3 d_4 t_{i,2}} \\
T_{i,3} &= ((u^{W_i} h)^{t_{i,1}})^{-d_2} \\
T_{i,4} &= ((u^{W_i} h)^{t_{i,1}})^{-d_1} \\
T_{i,5} &= ((u^{W_i} h)^{t_{i,2}})^{-d_4} \\
T_{i,6} &= ((u^{W_i} h)^{t_{i,2}})^{-d_3}
\end{aligned}
\tag{3.2}
$$

It returns $T_M = (M, T, T', \{T_{i,1}, T_{i,2}, T_{i,3}, T_{i,4}, T_{i,5}, T_{i,6}\}_{i \in [1,l]})$

- *Encrypt:* Takes as input the $pars, W_1, \ldots, W_m$ where W_i are the attributes values. It randomly chooses $\mu, s_{1,1}, s_{1,2}, \ldots, s_{m,1}, s_{m,2}, z_1, \ldots, z_m \in \mathbb{Z}_p$. It computes:

  $C = e(g,g)^{\alpha\mu}, D = g^{\mu}, E_{i,2} = g_2^{s_{i,1}}, F_{i,2} = g_4^{s_{i,2}}$

  $D_i = w^{-\mu}(u^{W_i} h)^{z_i}, E_{i,1} = g_1^{z_i - s_{i,1}}, F_{i,1} = g_3^{z_i - s_{i,2}}$

- *Test:* The test algorithm has as inputs the trapdoor and the cyphertext. We check:

$$
\prod (e(D, T_{i,1}) e(D_i, \frac{T_{i,2}}{H(e(T,T')^{\gamma}}) e(E_{i,1}, T_{i,3}) e(E_{i,2}, T_{i,4}) e(F_{i,1}, T_{i,5}) e(F_{i,2}, T_{i,6}))^{w_i} = C \tag{3.3}
$$

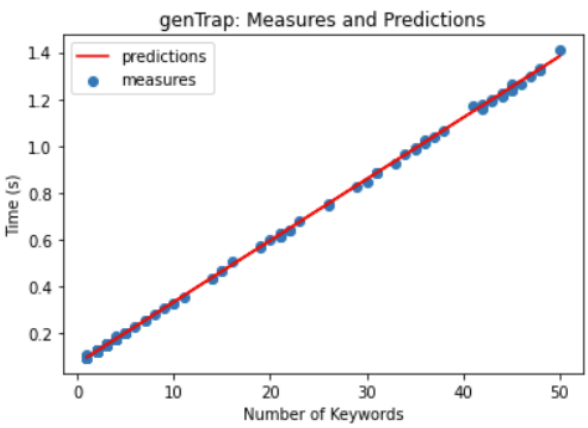
The left hand side of equation 3.3 results in: $e(g,g)^{\mu \sum v_i w_i} = e(g,g)^{\mu y \sum M_i w_i}$. Which will only be equal to C if $\sum M_i v_i = (1, 0, \ldots, 0)$.

Performance Analysis

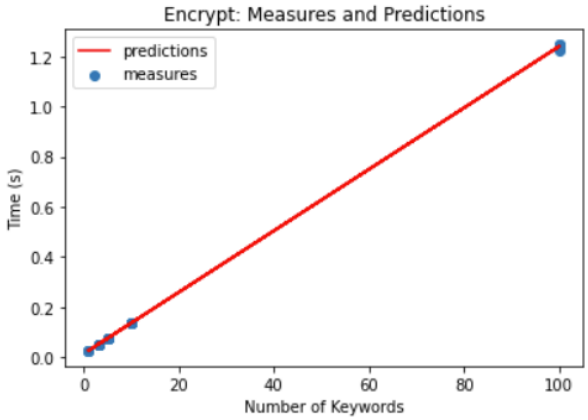
We want to study the complexity of the algorithm when we increase the number of keywords. It is obvious that the Setup and Keygen executions times are independent of the number of keywords used in the tests, while the Trapdoor, Encrypt and Test algorithm execution times grow linearly with the number of keywords used in the queries. The Encrypt and Test also depend linearly on the number of lines of the database.

In a searchable encryption scheme, the Test is the most critical algorithm that is going to be run for every line in the database, for every search made. We see in Equation 3.3 that it demands 6 pairing operations and will lead to very slow performances for most of the pairing curves.

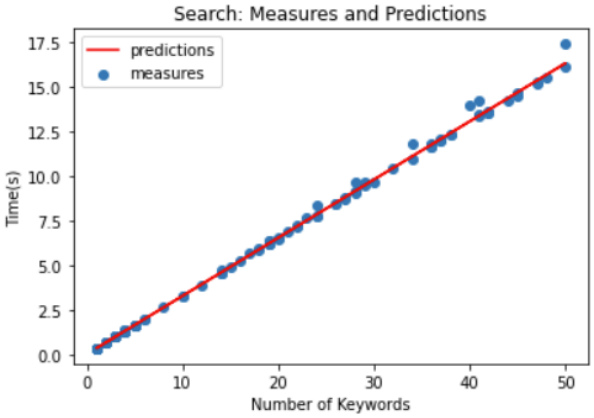
Doing the simulations as described in Section 3.1 we can obtain functions that estimate the time and space of every algorithm.



(a) Trapdoor generation time



(b) Encryption time normalized in the number of lines



(c) Search time normalized in the number of lines

Figure 3.1: Execution time of the algorithms in [2] when varying the number of keywords in a document

Table 3.2: Functions that describe the execution time in seconds depending on the number of keywords x

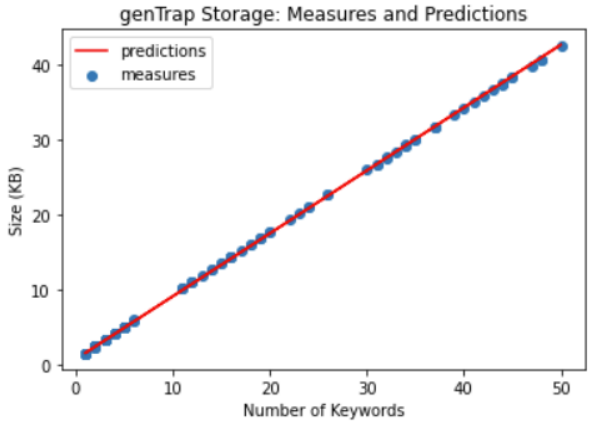
Algorithm	GenTrap	Encrypt	Test
$f(x)$	$0.026x + 0.072$	$0.012x + 0.014$	$0.326x + 0.072$
$R2$	1.000	1.000	0.999

In Table 3.2 we can see the functions that approximate the execution times, the R2 scores are very close to 1, so the approximations are good. While the *GenTrap* and *Encrypt* functions have a decent performance, the *test* function is extremely slow, taking on average 0.3 seconds for every searched keyword in a row! To search for 3 keywords in a small database of 1000 entries:

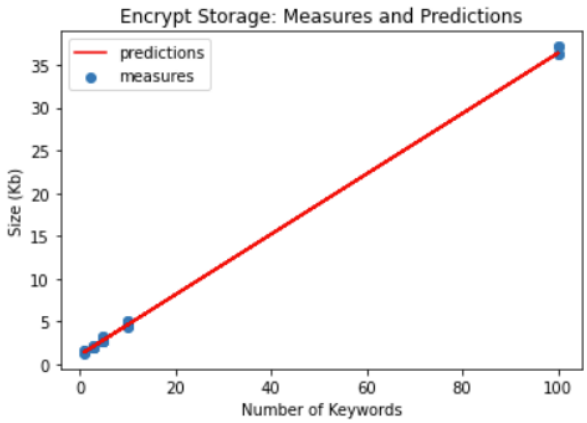
$$(0.326 * 3 + 0.072) * 1000 = 1050s \tag{3.4}$$

It takes 17.5 minutes! This deems it impractical in real word scenarios.

We take a similar approach to evaluate the size of a trapdoor and of the encryption of a single line. By inspecting the return values of the *genTrap* and *Encrypt*, we see that they grow linearly with the number of keywords. We can see this in a concrete way in Figure 3.2 and Table 3.3:



(a) Size of the trapdoor



(b) Size of the Encryption normalized on the number of lines

Figure 3.2: Sizes in kilobytes of the cyphertext when varying the number of keywords in a document

Table 3.3: Cyphertext sizes in KB as a function of the number of x keywords

Algorithm	Trapdoor	Encrypt
$f(x)$	$0.835x + 0.821$	$0.354x + 1.054$
$R2$	1.000	0.999

3.3.4 Searchable Encryption ABE - TF Scheme

As we already described, a Searchable Encryption scheme can be derived from an ABE where the trapdoor generation is analogous to a key generation and the Test is analogous to the decryption. In [41] an ABE scheme with fast decryption was proposed which required only two pairing operations for decryption. This work inspired several SE schemes that tried to improve the efficiency of the Test function. We will analyze one of them, the work presented in [3], and we will mention it as the TF Scheme.

The functions of the scheme are implemented in the following way:

- *Setup:* chooses a group G of generator g and a group $\hat{G}$ of generator $\hat{g}$ and $e : G \times \hat{G} \to G_T$. Choose randomly $\phi, \alpha, \beta, m \in \mathbb{Z}_p$ and compute: $h = g^{\phi}$, $\hat{h} = \hat{g}^{\phi}$, $U = e(g, \hat{g})^{\alpha(\beta-1)}$ and $V = e(g, \hat{g})^{\alpha\beta}$. Pick a hash function H that maps elements to $\mathbb{Z}_p$. The public parameters $pars = \{G, \hat{G}, e, g, U, V, h, H\}$ and the master secret key $MSK = \{\hat{g}, \hat{g}^{\alpha}, \hat{h}\}$

- *KeyGen:* Takes as input the public key PK, the private key msk and the LSSS matrix M that encodes the query and the attribute values: W.

  For every row i in M compute: select the random value $r_i \in \mathbb{Z}_p$ $\lambda_i = M_i v_i^T$ compute $\hat{\sigma}_i = \hat{g}^{H(W_i)}\hat{h}$ compute $d_{i,0} = \hat{g}^{\alpha}\lambda_i\hat{\sigma}_i^{r_i}$, $d_{i,1} = \hat{g}^{r_i}$, $Q_{i,j} = \hat{\sigma}_j^{r_i}$ for $j! = i$. It outputs the secret key $SK = (\{d_{i,0}, d_{i,1}, Q_{i,j}\}_{i=1}^{l})$

- *Encrypt:* The algorithm takes as input the public parameters PK, and a document with keywords = $W_1 \dots W_n$. Select a random number $k \in \mathbb{Z}_p$, compute $C_1 = V^k m$, $C_2 = U^k$, $C_3 = g^k$. For every attribute W_i compute: $\sigma_i = g^{H(W_i)}h$, $C_{4,i} = \sigma_i^k$ The cyphertext $CT = (C_1, C_2, C_3, \{C_{4,i}\})$.

- *Test:* Takes as input the cyphertext and the trapdoor and computes: $\hat{d}_{i,0} = d_{i,0} \prod_{x!=i} Q_{i,j}$ and $L = \prod_{x \in Query} Q_{4,x}$

  Compute $Z = \frac{e(C_3, \prod_{i \in I} \hat{d}_{i,0})}{e(L, \prod_{i \in I} d_{i,1})}$), check if $m = \frac{C_1}{C_2 Z}$

We modified the scheme to allow for an even faster Test algorithm by doing more computations in the Keygen phase.This new keygen returns $SK = (\{\hat{d_{i,0}}, \hat{d_{i,1}}\})$. In this way, we cut extra multiplications in the Test phase without the loss of security since those values can already be computed with all public information.

Performance Analysis

We proceed to test the scheme as described in Section 3.1 and try to build functions that predict the execution time of the algorithm. It is important to consider that in [3] we perform two pairing operations for every query, independently of the number of keywords. As we already showed in LSSS schemes a disjunctive query needs to be split into conjunctive queries and evaluated individually. That means that the Test algorithm execution time varies linearly with the number of $\mathbf{ORs}$ in a query.

The genTrap algorithm has quadratic complexity, but this increase is tolerable since the algorithm will be used only one time per search. The other algorithm's execution time as well as the storage increase remain linear. The execution times can be seen in Figure 3.4 and the respective functions in 3.4.

The size of the cyphertext grows linearly and the observations and equations are shown in Figure 3.4 and Table 3.4.

Table 3.4: Time in seconds as a function of number of keywords x and number of disjunctions y

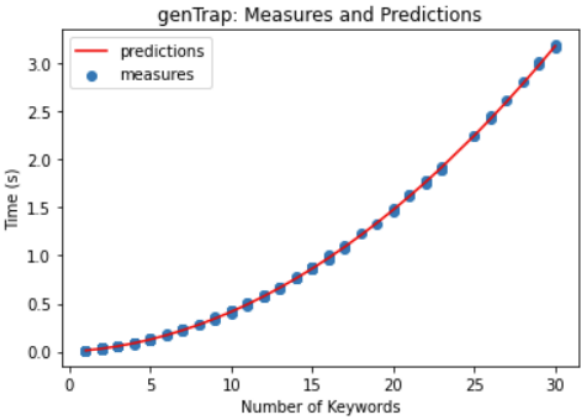
Algorithm	GenTrap	Encrypt	Test
$f(x,y)$	$0.003x^2 + 0.010x$	$0.003x + 0.027$	$0.105y + 0.113$
$R2$	1.000	1.000	0.919

Table 3.5: Size of the cyphertext in kilobytes as a function of the number of keywords x and number of disjunctions y

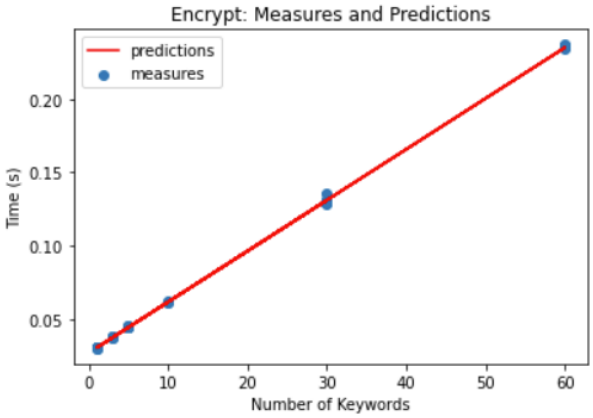
Algorithm	Trapdoor	Encrypt
$f(x,y)$	$0.337y + 0.533$	$0.074x + 2.078$
$R2$	0.962	0.915

3.4 Another Perspective - AF Scheme

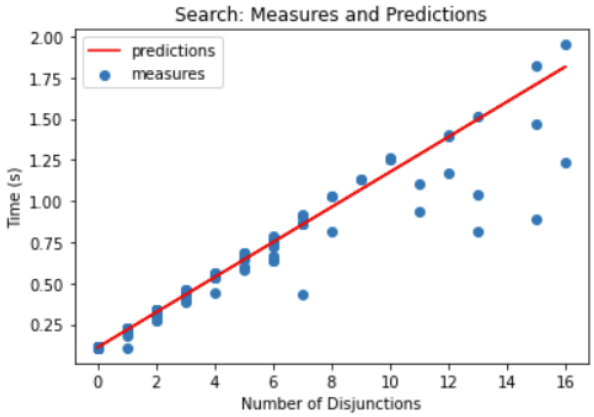
As we saw, to implement an LSSS scheme we need to split disjunctive queries into several conjunctive queries. So we can construct a scheme, with the same leakage and functionality (disjunctive + conjunctive) as the two previous ones, by using a scheme that performs only conjunctive searches. If the Test function has less than two pairing operations it should be faster.



(a) Execution time of the genTrap algorithm

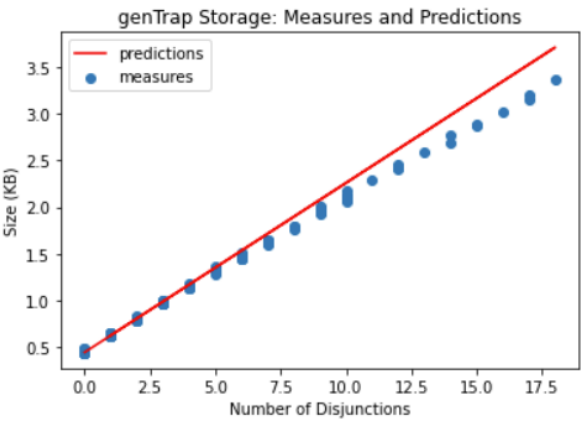


(b) Execution time of the Encryption algorithm

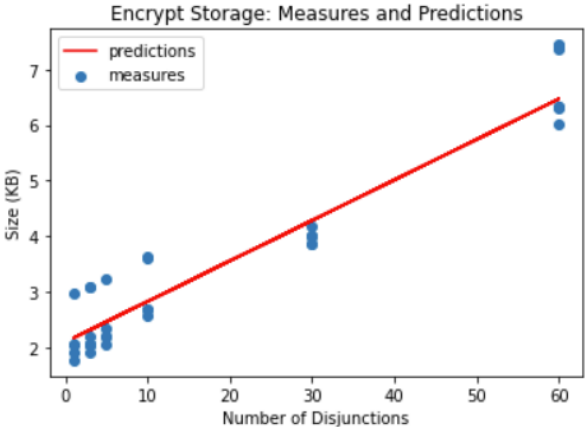


(c) Execution time of the Test algorithm when varying the number of ORs in a query

Figure 3.3: Execution times of the algorithms in [3] when varying the number of keywords or the number of disjunctions in a query



(a) Size of the trapdoor



(b) Size of the Encryption normalized in the number of lines

Figure 3.4: Sizes in kilobytes of the cyphertext when varying the number of keywords in a document

For that reason we choose the scheme: [1] that has only one pair operation in the Test algorithm, we will call it the AF Scheme. It goes as follows:

- *Setup:* chooses a group G of generator g and $e : G \times \hat{G} \rightarrow G_T$. Pick a hash function H that maps elements to $\mathbb{G}_1$. The public parameters are $pars = \{G, G_T, e, g, H\}$.

- *KeyGen:* Takes as input the public parameters $pars$, chooses randomly $s_1, s_2 \in \mathbb{Z}_p$ and compute: $Y_1 = g^{s_1}, Y_2 = g^{s_2}$. The public key $PK = \{g, Y_1, Y_2\}$ and the private key $SK = \{s_1, s_2\}$.

- *Encrypt:* The algorithm takes as input the public parameters $pars$, the public key PK and a document with keywords = $W_1 \dots W_n$. Select a random number $r \in \mathbb{Z}_p$, compute for every keyword i $A_i = e(rH(W_i), Y_1)$. Return the cyphertext $CT = (\{A_i\}, rY_2, rP)$.

- *Trapdoor* Takes as input the private key SK and the conjunctive query Q, that searches for keywords $W_1, \dots, W_n$, select a random $uin\mathbb{Z}_p$ and computes: $T_1 = (\frac{s_1}{s_2+u} modp)(H(W_1) + \dots + H(W_n)), T_2 = u$. Publish $T_Q = (T_1, T_2)$

- *Test:* Takes as input the cyphertext and the trapdoor and computes: $A = \prod A_i$ and $B = e(T_1, rY_2 + T_2 rP)$. Check if $A = B$.

3.4.1 Performance Analysis

The above scheme is much simpler than the other two, and it has only one pairing operation in the Test function. We can perform disjunctive queries simply by dividing the query Q with N **ORs** into $N+1$ conjunctive queries Q_i and executing the test algorithm for every Q_i. The result of the query Q is simply: $Test(Q_1)$ **OR** $Test(Q_2)$ **OR** $\dots$ **OR** $Test(Q_{N+1})$.

We implemented the scheme in python using the charm framework. As it as explained in Section 3.2, we have been testing the schemes in the BN254 pairing curve which has a pairing of the type $e : G_1 \times G_2 \to G_T$, different from the one used in [1]. We apply the techniques described in [42] to change the presented scheme in order for it to work under the BN254 curve. The only difference is that now, the hash function H now maps elements into $\mathbb{G}_{\not\equiv}$.

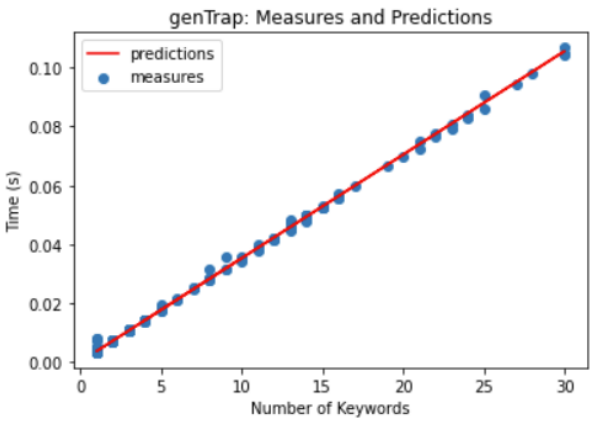
The *genTrap* and *Test* algorithms execution time grow linearly with the number of disjunctions in the query and *Encrypt* grows linearly with the number of keywords. The same logic applies to the size of the cyphertexts, the trapdoor grows linearly with the number of disjunctions and the encryption with the number of keywords. The execution times, cyphertext sizes, and prediction functions can be seen below:

Table 3.6: Time in seconds as a function of number of keywords x and number of disjunctions y

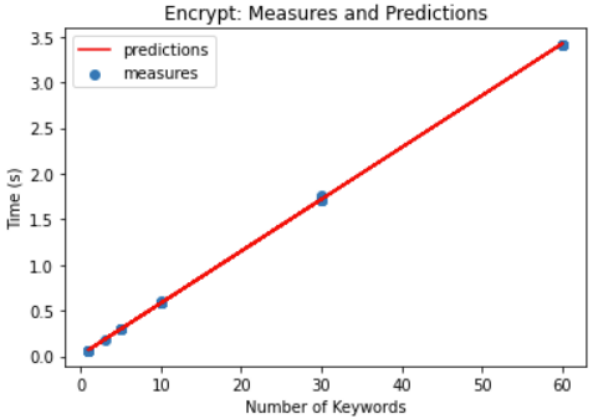
Algorithm	GenTrap	Encrypt	Test
$f(x, y)$	$0.004x$	$0.057x + 0.014$	$0.054y + 0.057$
$R2$	0.999	1.000	0.912

Table 3.7: Equations for estimating the size in KB of the cyphertext as a function of the number of keywords x and number of disjunctions y

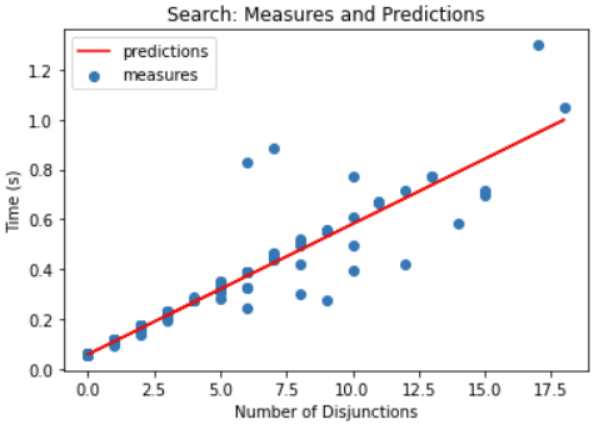
Algorithm	Trapdoor	Encrypt
$f(x, y)$	$0.179y + 0.447$	$0.781x + 0.307$
$R2$	0.978	0.999



(a) Time of trapdoor generation

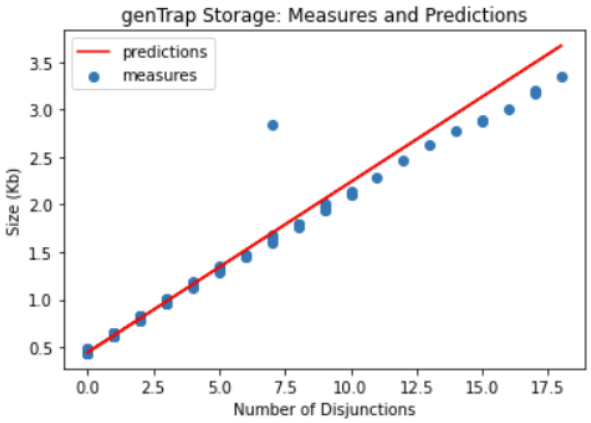


(b) Time for encrypting the keywords

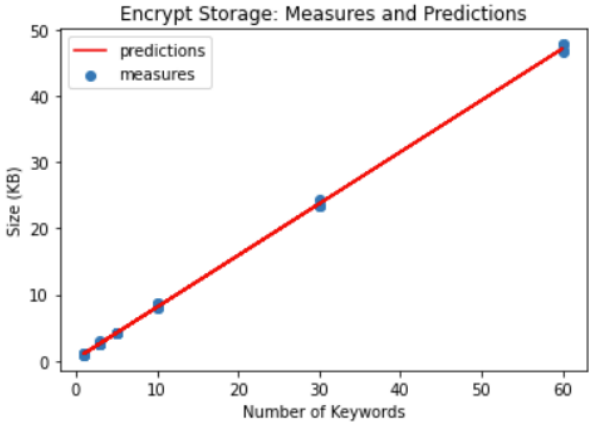


(c) Time for the Test algorithm

Figure 3.5: Execution times of the algorithms in [1] when varying the number of keywords or the number of disjunctions in a query



(a) Size of the trapdoor



(b) Size of the Encryption normalized in the number of lines

Figure 3.6: Sizes in kilobytes of the cyphertext when varying the number of keywords in a document

3.5 Conclusion

In this chapter, we studied the practical implementation of two SE Conjunctive and Disjunctive schemes, with an ABE realization. We showed that implementations that relied on LSSS are not optimal because one cannot evaluate the full query with a single execution of the Test algorithm and the server

is able to find out the structure of the query made.

To show that LSSS constructions are not efficient for Searchable Encryption we implemented and modified the conjunctive SE scheme [1] to perform the same types of queries as the other two. Our results showed that it outperformed them in the most critical algorithm, Test. It also can be used in any conjunctive scheme to extend it to a disjunctive scheme.

Our adaptation (AF Scheme) also has the fastest genTrap algorithm but has the slowest Encrypt, being more than 10 times slower than TF Scheme. In practice, the AF Scheme is ideal for read-heavy types of systems, and the TF Scheme is better for write-heavy systems. Storage-wise, AF needs about 5 times more storage than TF.

It is important to mention that our results depend greatly on the pairing curve we are using. Doing computation in other curves might speed up the schemes.

There are solutions that allow evaluating a disjunctive query in a single step and, like the inner product encryption [10], but the number of pairing computations grows linearly with the number of keywords deeming it not practical for real-world application. In our research, we have not come across with any scheme that allowed for both, a proper disjunction evaluation and a constant number of pairings per number of keywords in the query. Also worth mentioning is, that in this work we did not study lattice-based schemes, so a better solution in that space may exist.

Chapter 4

VANET Simulation

In this chapter, we are going to evaluate, in a vehicular network scenario, the implementations of the schemes [3] and [1] presented in chapter 2.1. For that, we are going to put forward a framework for testing protocols in realistic vehicular network scenarios using NS3[1].

Our goal is to build a system that can be used, not only for our specific needs, but that can test many other protocols in the application layer. This system supports the use of mobility traces to allow the realistic movement of nodes. We will build some using SUMO [2]. After building the scenario, the framework collects important metrics and evaluates how the protocols impact the vehicular environment.

In the case of searchable encryption, we are going to build two simple protocols that will allow the evaluation of the most important aspects of SE.

4.1 Simulation Environment

What makes a simulation successful is how well it mimics the real world. It is our understanding that two very important components need to be addressed in a realistic way, vehicular physical movement and vehicular information movement.

Vehicular physical movement is the way the cars move across the map. A common technique used is making cars move in random ways inside the map. This is not ideal because such movements do not occur in the real world. Better approaches are to use car movement information collected by special sensors or to use traffic simulation tools to simulate real traffic behavior.

Vehicular information movement is about how the data moves in a vehicular network. Most of the simulations done only have the cars exchange information relevant to the protocol being tested and neglect that in a real network other types of data are being exchanged that can impact the performance of the network and the protocol.

[1] https://www.nsnam.org/
[2] https://www.eclipse.org/sumo/

4.1.1 Vehicular Physical Movement

As mentioned there are two ways of obtaining a realistic vehicle movement in the simulation, by real data of car movement or by traffic simulation tools. The first approach would be ideal since it is as realistic as possible, but normally the data traces are too sparse as they only measure a small subset of vehicles in the global traffic. The second approach is less realistic but allows us to generate a trace containing all the information on the traffic in the region. Dealing with incomplete data might lead to unrealistic results which is why we choose to use the second approach and use some traffic simulation tools to build a data trace from a real-world location.

Our approach is summarized in image 4.1, first, we create a region of interest using the Netedit tool we created a double-lane road with cars moving in both directions with 1 km of size, the speed limit is $28.0m/s$.

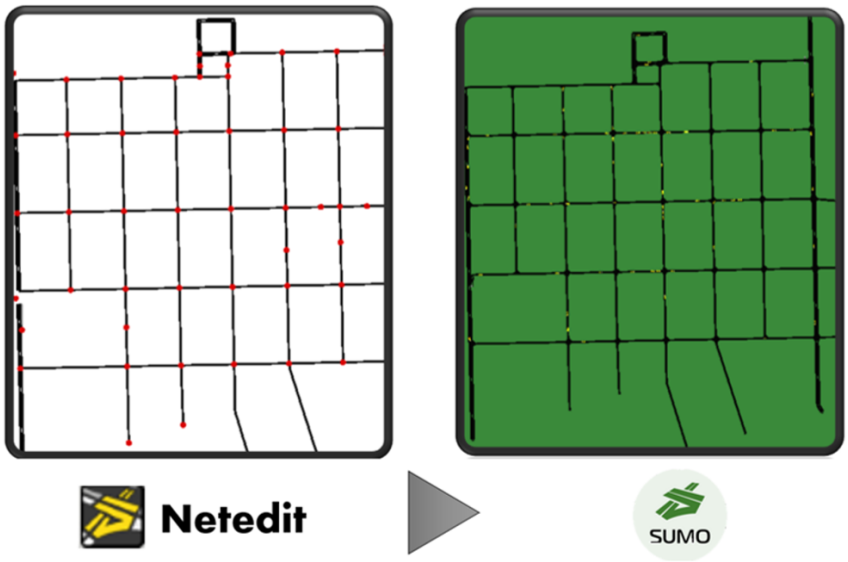


Figure 4.1: Steps in the transformation of a map into a simulation trace

With this better map representation, we are able to generate some traffic on it. We use the *randomTrips.py* script, provided by SUMO that generates random traffic in the map. The script takes as input a seed value, that generates different routes for every value. We use ten different seeds to test the same scenario in different conditions.

Finally, we write a configuration file that combines the map and the routes. The result simulation can be seen in SUMO in the Figure in 4.2 where the yellow points are cars. This configuration file can be used to generate an NS2 trace file that NS3 can read and interpret. By the end of this whole process, we have three different NS2 trace files for the chosen map.

The traces generated in this process are summarized in table 4.1. It is important to note that it does not mean that we are going to do a simulation of 150 seconds with 58 cars in NS3. Once we have the full traces we can simulate subsets of them with less time and fewer cars. Doing the NS3 simulation with fewer cars than in the trace actually adds realism because it means not all cars have connectivity

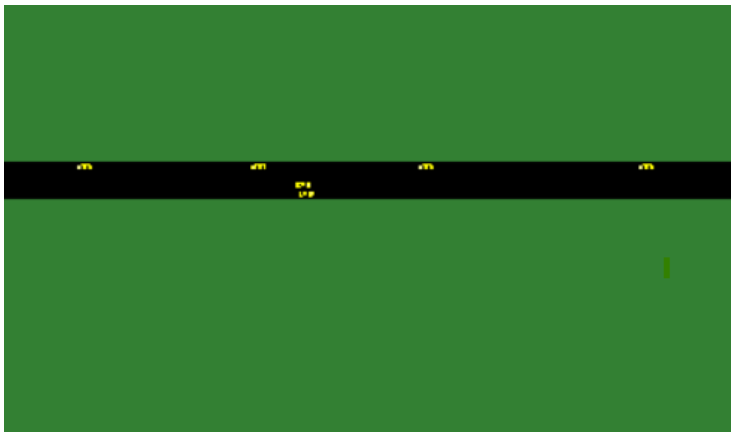


Figure 4.2: Sumo simulation excerpt

equipment.

Table 4.1: Simulation Traces specification

Size ($m \times m$)	1000×12
Number of cars	58
Duration (s)	150
Seeds	$1, 2, 3, 4, 5, 6, 7, 8, 9, 10$

4.1.2 Vehicular Information Movement

The original idea of this project was to have inter-vehicle communications using the WAVE protocol, as seen in Figure 4.3. However, NS3 does not provide full support for WAVE so we chose to use only the IEEE 802.11p.

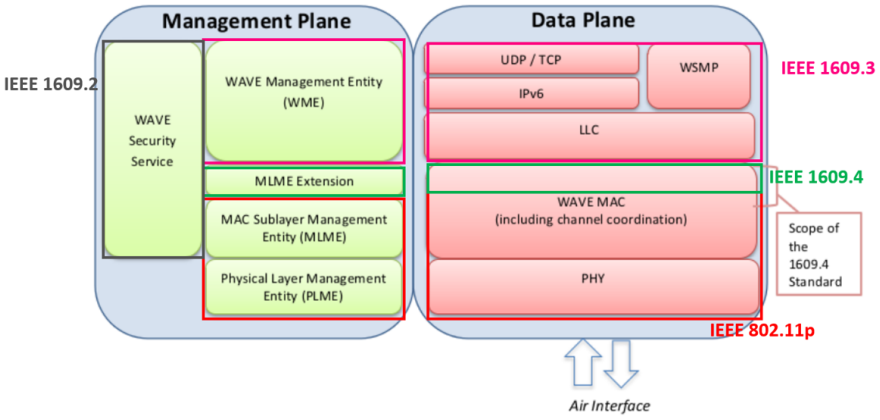


Figure 4.3: Wave protocol described in [43]

For the wave parameters, we used the same ones as described [44] that are in line with the IEEE 802.11p standard, and are shown in table 4.2.

Table 4.2: Wifi 802.11p simulation parameters

Transmitted Power	23 dBm
Transmitted Gain	15 dBi
Received Gain	5 dBi
Frequency	5.9 GHz
Bandwidth	10 MHz

The idea is to install apps in the nodes that simulate the information transmitted in a normal network. First, we install in every node a basic safety message (BSM) app that broadcast a 200 bytes packet with a frequency of 10Hz. This is the elementary application that every vehicular network should have and enables basic safety functionalities to its users.

We also simulate extra data exchange in the network by making the nodes transmit UDP packets. To do that we first install an echo UDP server in every node. Then we install a UDP client with a random destination in 10 nodes. Through the simulation, the nodes will transmit UDP packets that will be routed to their destination through the network. Upon receiving a packet, the node will echo the response back to the client. Our goal is to mimic the seemingly chaotic nature of a real network. The parameters of every application are summarized in table 4.3, the UDP Server does not have a value for the data generation rate because it depends on the number of incoming packets.

Table 4.3: Network Environment specification

App	Number of apps per Node	Data Generation Rate (bits per second)	Type
BSM	1	16000	Broadcast
Echo UDP Server	1	$\times$	Unicast
UDP Client	0 or 1	170	Unicast

After establishing the environment we also have to install in the nodes the protocol we want to evaluate. For that we built a new app in NS3 called *SendOnRecv*. The app works as follows:

- Waits on a given port sp for a message to be received
- When a message is received, check if it is from a user that is already a client of the app. If it is not waits for a time t and starts sending n packets with a data rate dr to the client address with a destination port dp
- If the user is already a client of the app, do nothing.
- Tracks the number of packets sent by using NS3 callbacks.

The parameters sp, dr, n, dp can be set by the user. The goal is to have a kind of server app, that after receiving a request (packet received), does some processing that takes t milliseconds and sends back a response to the client (packets send). With this tool, we can build a realistic scenario in a realistic environment.

4.2 Simulator Data Flow

We wanted to keep the network configuration as general as possible so researchers would be able to use it in other situations. However, we still need a way to evaluate the schemes we are interested in (described in chapter 2.1). As a solution, we use a Python script that serves as a middleman between the user and the NS3 scenario and converts metrics that are relevant to the searchable encryption scenario into metrics that are relevant to the vehicular network scenario. The Python script also parses the NS3 output and stores the results in *csv* file, so they can be analyzed in a convenient way.

So, the NS3 simulator has as relevant inputs the number of nodes, processing time, and a number of packets (parameters t and n of the *sendOnRecv* app), seed (specifies the trace map to use). And, as seen in chapter 2.1, the searchable encryption schemes have as relevant inputs: number of lines nl, number of keywords x, and number of disjunctions y in a query scheme used. In order to convert from one to the other we use the time functions $t_k(x, y)$, described in tables (3.2, 3.4, 3.6) and the size functions $s_k(x, y)$, described in tables (3.3, 3.5, 3.7), k being the encryption scheme used.

The processing time is simply $t_k(x, y) * nl$ and the number of packets is the size of the cyphertext divided by the packet capacity (1024 bytes), $s_k(x, y) * nl/1.024$. In summary, the metrics of the Python script are the number of nodes in the simulation, scheme, number of lines, and query size.

Finally, the metrics we want to evaluate are total lost packets, total received packets, and average delay. The information on the lost and received packets is obtained with NS3 callbacks, where every time a packet is sent or received by an app, a function is executed that increments a counter. The delay is computed with the NS3 flow monitor [45] library. The data flow described can be seen in image 4.4.

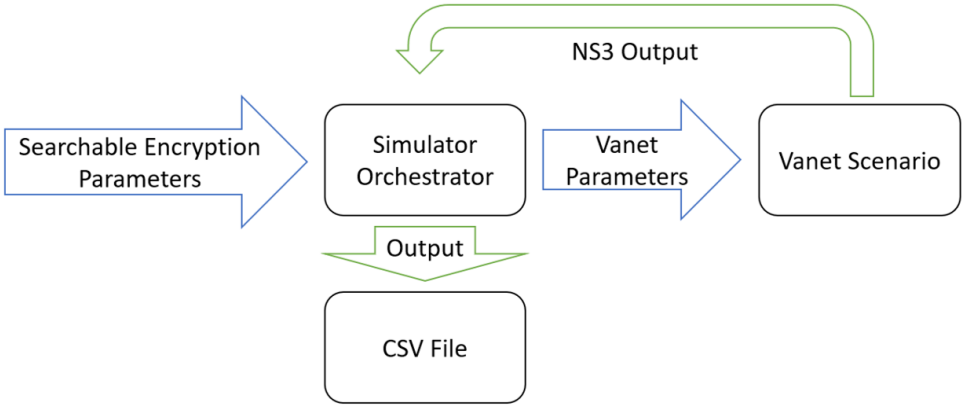


Figure 4.4: Data Flow

4.3 Scenario 1 - Encryption Test

In this section, we are going to describe in detail the first VANET scenario we want to analyze. The main objective is to see how encryption speed and size impacts the VANET network. First, we are going to present the scenario that we think is realistic, then we are going to explain the advantages of this protocol, implement it in NS3 and gather results.

For this scenario, we are going to install a base station node in the center of the map. This base station is going to communicate with the cars using the radio as a medium. It broadcasts a message containing some public keys and some information that the vehicles need to share. When a given vehicle receives this broadcast message, it encrypts the information requested with a searchable encryption scheme, using the public key, and sends it back to the base station. The protocol can be seen in image 4.5.

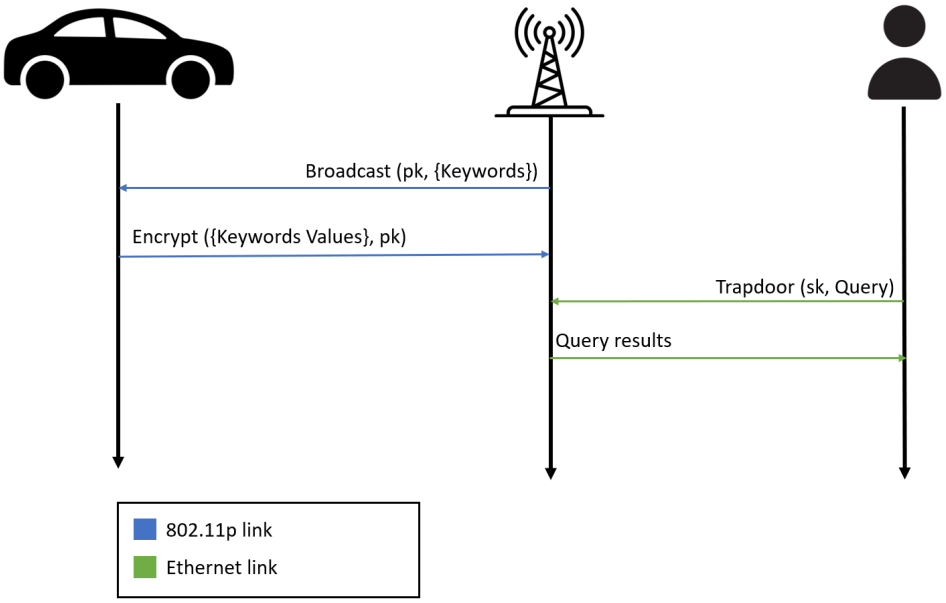


Figure 4.5: Protocol of the first scenario

The base station owner can now query the base station for relevant information while keeping everything hidden. Since, normally, the base stations are connected to the general network infrastructure, the queries can be made via a cabled network to the Internet and the information exchange is faster and more reliable.

4.3.1 Safety Improvements

This protocol protects user data on two fronts. First, it allows for safer communication between the base station and the vehicle by adding an extra layer of encryption. Because the radio medium can be listened to by everyone it is important to encrypt communications. Layers 1 and 2 protocols are historically bad at providing secure communications (WPA for example), providing extra security in the application layer is fundamental.

Second, it hides the data from the base station. Apart from all the normal insecurities a piece of software might have, the base station hardware is exposed in the middle of the street and this can lead to all kinds of attacks. Apart from that, we can imagine that the base station might be shared by several users, running different sets of applications, making it extra important to keep the data secure among apps.

Most importantly, even though the data is encrypted, the data owner is still able to use it and process

it by generating queries using the trapdoor algorithm. The base station can respond to the queries by executing the Test algorithm.

As an example application of this protocol, we can image an entity that is responsible for a given segment in the road and would like passing cars to send some of their sensor readings to the base station. With SE cars can send their readings safely and the entity can safely query the base station for a more fine-grain analysis.

4.3.2 Implementation in NS3

To be realistic we are going to assume that every node starts the protocol at a different time, in this case, it will be 0.75 times the node id. The implementation of the scheme in our NS3 environment can be seen in Listing 4.1). We can see how simple and intuitive it is. The underlying operations that the simulator is doing are the following:

1. Place the base station node in the given coordinates and install a UDP Echo Client on it. This app is going to broadcast a packet with a frequency of 10 Hz and wait for a response (first line of Listing 4.1).

2. Install the SendOnRecv app in every vehicle node. Set the destination port to be the port of the echo client app from the base station (second line of Listing 4.1).

Listing 4.1: Scenario 1 implementation

```
setBroadcastNodes(carLstPort, baseStation, end, start);
setReplayNodes(carLstPort, 0, nodes, nPackets, MilliSeconds(processingTime), end, start);
```

In the end, we will be able to collect the following metrics: packets sent by the vehicles, packets received by the base station, and average delay. We will simulate for schemes [1, 3] and for the protocol running in plain text. The parameters are as follows:

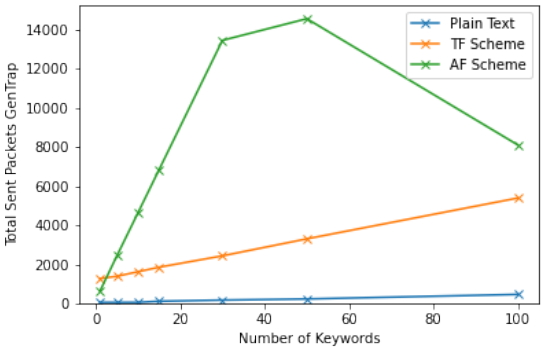
Table 4.4: Scenario 1 - Simulation Parameters

Simulation Time	150
Number of Nodes	58
Base Station Data Rate	0.1 packet / second
Vehicle maximum data generation rate	32 kbps
Number of database entries sent	10
Protocol starting time	node id $\times$ 0.75
Number of keywords in database entry	1, 5, 10, 15, 30, 50, 100
Searchable Encryption Schemes	Plain Text, [1, 3]
Number of samples per data point	10

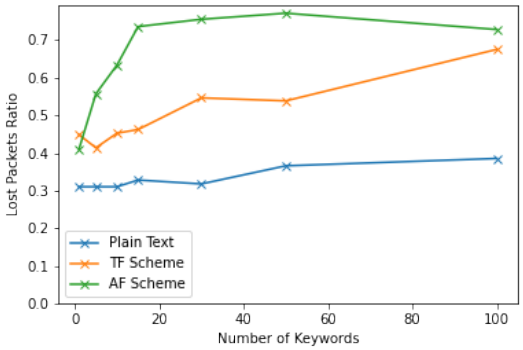
4.3.3 Results

The simulation results can be seen in Figure 4.6. As was described in chapter 3 the processing time and the size of the encryption depends on the number of keywords. Where scheme [1] (AF Scheme) produces bigger cyphertexts and takes more time to complete than [3] (TF Scheme).

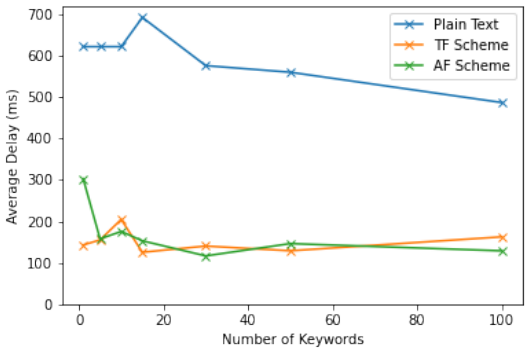
This means that by increasing the number of keywords the encryption schemes will produce more packets than the plain text version. The encryption schemes will also have a computational overhead over the plain text. In Figure 4.6 we can see how those overheads translate into the VANET scenario. Specifically, how does the average delay, and the number of packets sent and received increase.



(a) Number of Packets Sent



(b) Packets Lost



(c) Average Delay in Miliseconds

Figure 4.6: Scenario 1 simulation metrics for multiple encryption schemes, with multiple nodes

The first thing interesting to point out is how the percentage of lost packets might influence the average delay. Normally, we would expect an increase in the average delay when we compare the plain text scheme with the encryption schemes. However, in Figure 4.6 we do not see this behavior. This phenomenon can be explained by the number of lost packets, while the delay decreases, the percentage of lost packets increases, meaning that, probably, only the first packets sent by the vehicle reach their destination, and the rest (the ones with higher delay) were lost. In conclusion, if the number of lost packets is high enough we may see a reduced average delay.

In figure 4.6 we can see how the cyphertext expansion, affects the number of packets sent. The three schemes increase linearly with the number of keywords, but at a very different rates. As was

already described, the AF scheme has the highest cyphertext expansion and this results in more packets sent. An interesting phenomenon occurs in the AF scheme, between the 40th and 50th keywords the number of sent packets starts to decline. This happens because the duration of the simulation is finite and so there is a limit to the number of packets a node can send. The reason we see a decrease instead of a constant number of sent packets is that the processing time increases with the number of keywords, meaning that the nodes have less available time to transmit the packets. We could increase the simulation time to solve these problems, but that would also mean increasing the number of nodes resulting in a much slower simulation scenario.

The percentage of packets lost goes as expected, as we increase the number of keywords we increase the number of packets sent and the network congestion, leading to an increase in the number of packets lost. As anticipated, when using the plain text we have a smaller percentage of lost packets, starting at 33% but going up to around 50% with a steady increase. The TF scheme has also had a steady increase but with a much higher percentage of lost packets. The AF scheme experiences a fast increase but starts to decrease after the 10th keyword. This decrease is misleading because it probably happened due to the decrease in the number of sent packets, which has nothing to do with the scheme but with how the simulation is built.

Overall, the AF Scheme has the slowest encryption algorithm and the biggest cyphertext expansion leading to the biggest percentage of lost packets. The scheme does not scale well for a high number of keywords and the metrics become much worst than for the plain text one. Scheme 1 deals better with an increase in keywords (up to the 10th keyword). The TF scheme scales well in the number of keywords but imposes a big overhead in the system. The second scheme is the worst in every metric and does not scale well when we increase the number of keywords. It should only be used in small databases or in systems that require few write operations.

4.4 Scenario 2 - Testing the Test algorithm

This scenario was built to test the impact of the gentrap and the test algorithm. We are going to build on top of the first scenario by having the same base station broadcasting a message to all nearby vehicles. However, we invert the role of the base station, having it store the data that belongs to the vehicles that pass by.

So, after receiving the broadcast message, the vehicles run the genTrap algorithm to query the base station. The BS receives it and does the Test algorithm to find data entries that match the query. It will then send back to the car those entries. The protocol can be seen in Figure 4.7.

4.4.1 Safety Improvements

This protocol allows for the safe delegation of data from the car to the base station. The vehicles can store encrypted data in BS but are still able to query it. The query and the data are kept hidden from the base stations and attackers as explained in chapters 2.1 and 3 from the base stations and attackers.

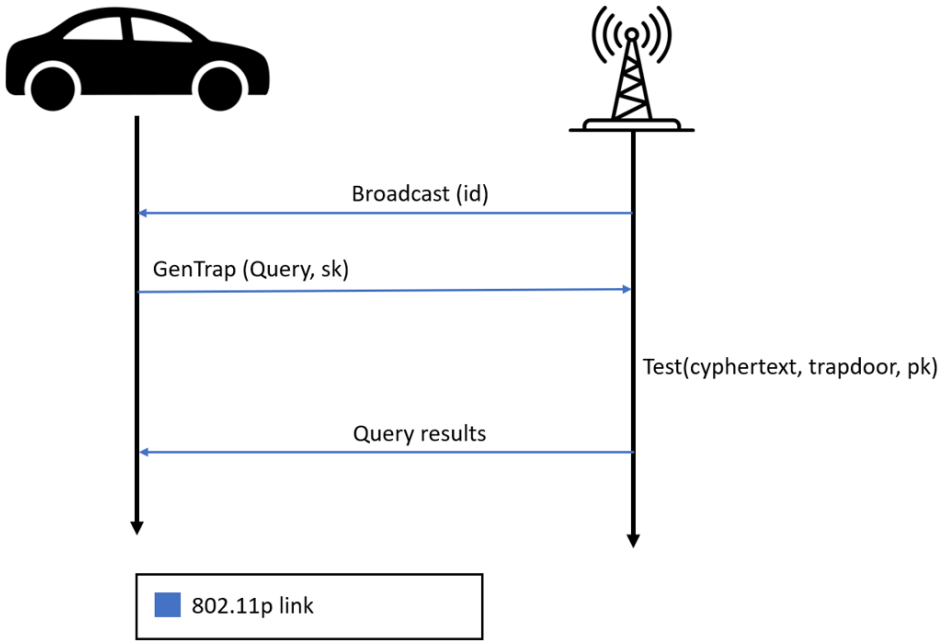


Figure 4.7: Protocol of the second scenario

Without searchable encryption, this type of scheme would be much harder to implement, because normal encrypted data cannot be queried. What would have to be done is having the base station send back the entire database back to the car, and then the vehicle could decrypt it and evaluate the desired query. It is easy to see how this approach is not as efficient, especially in a VANET scenario where communications are more difficult, sending back the entire encrypted database would be a problem.

4.4.2 Implementation in NS3

As mentioned earlier, this protocol is built on top of the previous one and can be easily built into our framework, we will also have the protocol starting at different times for different vehicles. We do the following:

1. Transform the base station in a broadcasting node, for port $p1$
2. Install sendOnRecv apps in the vehicles nodes, set the listening port to $p1$ and the destination port to $p2$. Set the processing time and number of packets accordingly.
3. Install new sendOnRecv apps on the base station, set the listening port to $p2$ and destination port to $p3$, and set the processing time and number of packets
4. Finally we install a server app on the vehicle nodes, listening to port $p3$

The metrics will be collected for communications between ports $p1 \rightarrow p2$ and $p2 \rightarrow p3$. The simulation input parameters are going to be: Number of keywords and lines in the database, the number of keywords, and disjunctions in the query.

The processing time and the number of packets are going to be calculated by using those input parameters as inputs in the functions defined in 3. For communication $p1 \rightarrow p2$ the processing and number of packets are defined by the genTrap functions. For $p2 \rightarrow p3$, the processing time is defined by the Test function times the number of lines. The number of packets is more complicated because it depends on the number of matches a given query has. To simplify this process we assume every query matches 10% of the database. So the number of packets is going to be defined by the size of the encryption times the number of lines divided by the size of the packets. The simulation parameters can be seen in table 4.5

Table 4.5: Scenario 2 - Simulation Parameters

Simulation Time	100 s
Number of Nodes	58
Base Station Frequency	10 Hz
Vehicle Maximum Data Generation Rate	32 kbps
Number of database entries	5, 50, 100, 150, 200
Number of keywords in database	10
Number of disjunctions in query	1, 2
Protocol starting time	node id $\times$ 0.75
Searchable Encryption Schemes	Plain Text, [1, 3]
Number of samples per data point	10

4.4.3 Results

We need to evaluate this scenario in two different steps. The first step is the generation of the trapdoor, we want to see how the system reacts to the querying phase. Next, we evaluate how the base station will test and respond to that query. We will capture the relevant metrics and evaluate the system's performance.

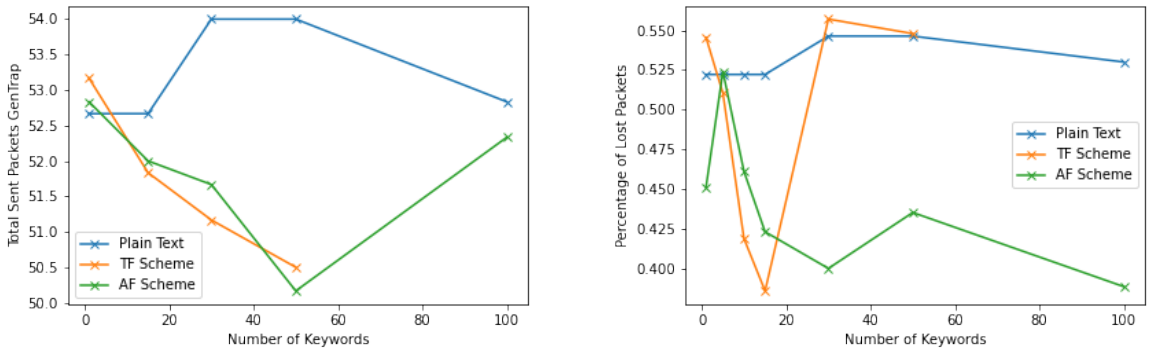
Step 1

In this part, we wish to evaluate the trapdoor generation phase. As seen in image 4.7 the base station broadcasts messages to notify the nearby vehicles. Upon receiving one of those, the vehicles generate a trapdoor for the keywords they wish to search and send it back to the base station.

The trapdoor generation time depends on the number of keywords and the size depends on the number of disjunctions, as seen in Tables 3.4, 3.5, 3.6, 3.7. Even though the size is bigger, that does not translate into more packets being sent, since it is still fewer bytes than the capacity of the packet.

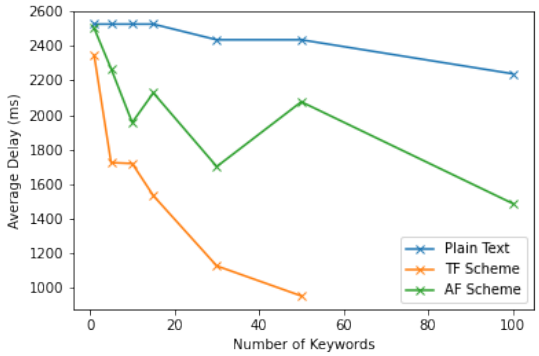
An interesting phenomenon occurs when evaluating the average delay and the percentage of lost packets. As already mentioned, by intuition we tend to think that by increasing the overhead in computation we increase the average delay and the percentage of lost packets. We see in the simulation results that this is not true, both encryption schemes perform better in those metrics than the plaintext version.

This may happen because the trapdoor part of the protocol is the smallest and fastest one, so it does not impact the system in a meaningful way. That would explain why the data in Figure 4.8 seems



(a) Number of packets sent

(b) Percentage of Lost Packets



(c) Average Delay in Miliseconds

Figure 4.8: Scenario 2 step 1, simulation metrics for multiple encryption schemes with multiple nodes

uncorrelated with the number of keywords.

Step 2

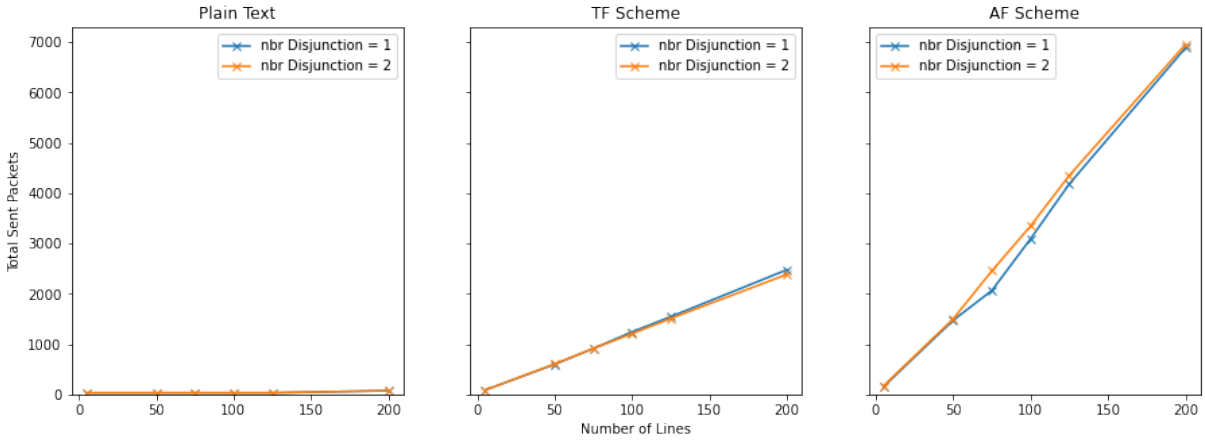
In step 2, the base station receives a trapdoor and performs the test in every line of the database, it then sends back the corresponding entries back to the vehicles. To simplify the simulations, we assume that every trapdoor matches 10% of the number of lines.

As seen in equations 3.4, 3.5, 3.6, 3.7, the computation overhead and the size of the cyphertext depends on the number of disjunctions. The simulation results can be seen in Figure 4.9 for a fixed number of 10 keywords.

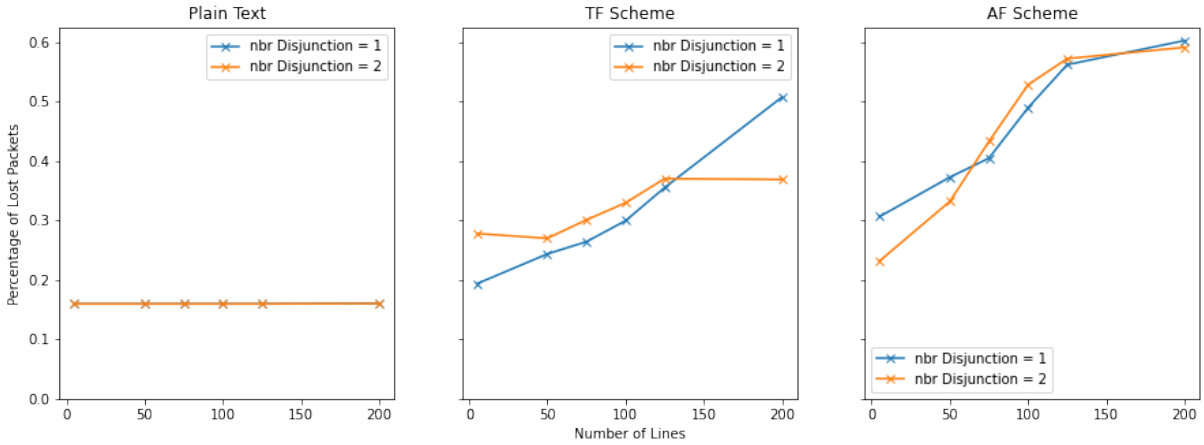
Figure 4.9 shows that an increase in the number of lines leads to an increase in the number of lost packets. Scheme AF responds worst to this and for more than 100 lines has more than 50% of lost packets.

In this protocol, the number of lost packets is smaller due to the fact that fewer packets are being exchanged, when can confirm that by comparing Figure 4.6 (a) with 4.9 (a).

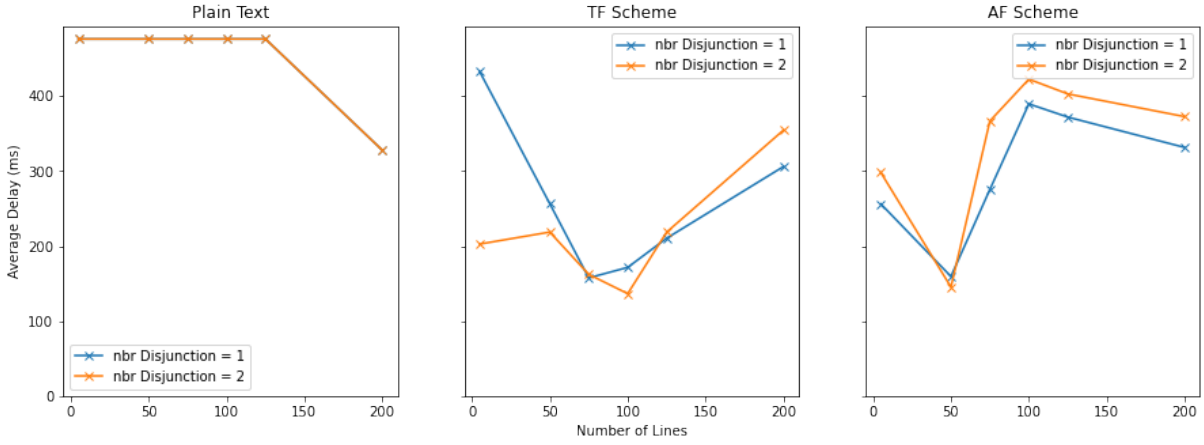
It is interesting to observe how the number of disjunctions is affecting the metrics. As it was shown in chapter 3, the number of disjunctions will increase the computation time when doing the Test algorithm, as bigger databases mean more executions of it. We can see in Figure 4.9 (b) that increasing the



(a) Number of Packets Sent



(b) Percentage of lost packets



(c) Average Delay in Miliseconds

Figure 4.9: Scenario 2 step 2, simulation metrics for multiple encryption schemes, with multiple nodes

number of disjunctions does not increase the number of lost packets, this signals that the overhead in computation is not a critical factor for network metrics. And that the cyphertext expansion represents the biggest bottleneck for the network.

4.5 Conclusion

In this chapter, we aimed to test the cryptographic schemes presented in chapter 3 in VANET scenario. To carry off this goal we built a simulation framework in NS3 that allowed for rapid prototype simulation of VANET protocols. We also built two protocols that leverage Searchable Encryption capabilities and simulated for different schemes, comparing the results with the plain text equivalent.

After simulating the two protocols, the conclusion we reach is that the schemes perform poorly in the first scenario, to the point that the amount of lost packets probably deems the protocol infeasible in the real world. Protocol 2 yields better results and is achievable if the databases are in the order of 500 entries. Queries should also have limited conjunctions, where 2 might be too much depending on the size of the database.

This might seem limiting, but we could create a database that only has certain fields encrypted using SE, we could then create queries that search for secret and nonsecret attributes. We first run the nonsecret query through the database and only run the secret one with matching elements from the first step. In this way, we could combine the fast querying that modern databases offer with the capabilities of searchable encryption.

The main differences in performance between the 2 scenarios have to do with the number of packets generated by the protocols. Scenario 1 generates much more than scenario 2. We can leverage this knowledge to further improve protocol 2. Since we know that in this scenario the car is the data owner we can keep in the database two encryptions for every entry. One is the keywords generated by the searchable encryption scheme $enc1$, the other can be generated from any other public key cipher available $enc2$. In this way, the base station does the Test algorithm with $enc1$ but returns $enc2$. If the size of $enc2$ is smaller than $enc1$ (probably is) we can achieve much better performance.

Of course, this can not be used in scenario 1 because the cars need to send searchable encryptions to the base stations, otherwise, they would not be able to search the cyphertexts.

Chapter 5

Conclusions

In this master thesis, we set out to explore searchable encryption and how it could be applied in practical IoV architectures. Currently, security is a concern in vehicular networks and searchable encryption can provide some solutions in that space.

5.1 Achievements

In this work, we presented an in-depth practical analysis of the implementations of searchable encryption schemes. Normally SE papers focus more on the theoretical aspects of the schemes and the implementations are only briefly analyzed with superficial tests. Our work reliably tested those implementations by creating meaningful queries with multiple **AND** and **OR**, the queries made on the encrypted database were also done in plain text and the results were compared to test the correctness of the implementations.

In this work, we presented an in-depth practical analysis of the we showed that schemes that rely on LSSS to do disjunctive queries were equivalent to a conjunctive scheme doing multiple conjunctive queries. To confirm that, we implemented from scratch a Searchable Encryption Conjunctive scheme [1] and modified it to perform disjunctive queries, these modifications had a Test algorithm twice as fast as the fastest disjunctive scheme we tested.

From the vehicular network standpoint, we built one of the most realistic VANET scenarios using vehicular traces and simulated network traffic. It is also extremely modular and with just a few lines of code, one can simulate a network protocol. With this tool, we tested two different protocols that hinted at what type of situations in searchable encryption could be used best.

5.2 Conclusions

In chapter 3 we verified that ABE searchable encryption is probably bonded on having at least one pairing operation in the Test algorithm. This means that evaluating a query in this setting depends on the computation speed of a pairing, which is slow. To circle this problem we discuss the idea of mixing

encrypted and plain text keywords, we would first evaluate the query in plain text and then, in a reduced number of entries, we would do the searchable encryption test.

In a VANET situation, however, the cyphertext expansion of the SE schemes seemed to be the bigger problem. We verified that the network can tolerate delays in the computation, but reacts very badly to the increasing number of transmitted packets. This means that protocols that rely on transmitting a lot of encrypted keywords via IEEE802.11p are not feasible. Protocols that only need to transmit the results of queries work better because we can combine regular SE schemes with classic public key encryption and transmit only the second one.

5.3 Future Work

For future work, more searchable encryption schemes can be tested, mainly post-quantum ones that rely on a completely different set of assumptions and use different techniques. More scenarios on the IoV architecture can also be explored using different transmission technologies, such as 5G and the complete WAVE stack. New vehicular traces can also be used to have a more complete picture of how the protocols are performing.

Overall, the literature lacks studies that try to estimate the traffic on vehicular networks. Answers to questions such as: ”how many applications an average node would use?” and ”how much data is exchanged between nodes?” could provide useful insight when building a more realistic network scenario.

Bibliography

[1] D. J. Park, K. Kim, and P. J. Lee. Public key encryption with conjunctive field keyword search. In C. H. Lim and M. Yung, editors, *Information Security Applications*, pages 73–86, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31815-6.

[2] H. Cui, Z. Wan, R. H. Deng, G. Wang, and Y. Li. Efficient and expressive keyword search over encrypted data in cloud. *IEEE Transactions on Dependable and Secure Computing*, 15(03):409–422, may 2018. ISSN 1941-0018. doi: 10.1109/TDSC.2016.2599883.

[3] Y.-F. Tseng, C.-I. Fan, and Z.-C. Liu. Fast keyword search over encrypted data with short ciphertext in clouds. Cryptology ePrint Archive, Report 2021/974, 2021. `https://ia.cr/2021/974`.

[4] J. Contreras-Castillo, S. Zeadally, and J. A. Guerrero-Ibañez. Internet of vehicles: Architecture, protocols, and security. *IEEE Internet of Things Journal*, 5(5):3701–3709, 2018. doi: 10.1109/JIOT.2017.2690902.

[5] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini, and Y. Khamayseh. Comprehensive study of symmetric key and asymmetric key encryption algorithms. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–7, 2017. doi: 10.1109/ICEngTechnol.2017.8308215.

[6] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585062. doi: 10.1145/1536414.1536440. URL `https://doi.org/10.1145/1536414.1536440`.

[7] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, page 113–124, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450310048. doi: 10.1145/2046660.2046682. URL `https://doi.org/10.1145/2046660.2046682`.

[8] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000*, pages 44–55, 2000. doi: 10.1109/SECPRI.2000.848445.

[9] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 506–522, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24676-3.

[10] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In N. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, pages 146–162, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-78967-3.

[11] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In Y. Ishai, editor, *Theory of Cryptography*, pages 253–273, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-19571-6.

[12] A. Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology*, pages 47–53, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg. ISBN 978-3-540-39568-3.

[13] A. Sahai and B. Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 457–473, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-32055-5.

[14] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 89–98, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595935185. doi: 10.1145/1180405.1180418. URL `https://doi.org/10.1145/1180405.1180418`.

[15] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, pages 321–334, 2007. doi: 10.1109/SP.2007.11.

[16] Y. Zhou, N. Li, Y. Tian, D. An, and L. Wang. Public key encryption with keyword search in cloud: A survey. *Entropy*, 22(4), 2020. ISSN 1099-4300. doi: 10.3390/e22040421. URL `https://www.mdpi.com/1099-4300/22/4/421`.

[17] P. Yanguo, C. Jiangtao, P. Changgen, and Y. Zuobin. Certificateless public key encryption with keyword search. *China Communications*, 11(11):100–113, 2014. doi: 10.1109/CC.2014.7004528.

[18] Q. Zheng, X. Li, and A. Azgin. Clks: Certificateless keyword search on encrypted data. In M. Qiu, S. Xu, M. Yung, and H. Zhang, editors, *Network and System Security*, pages 239–253, Cham, 2015. Springer International Publishing. ISBN 978-3-319-25645-0.

[19] J. W. Byun, H. S. Rhee, H.-A. Park, and D. H. Lee. Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In W. Jonker and M. Petković, editors, *Secure Data Management*, pages 75–83, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-38987-3.

[20] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, page 707–720, USA, 2016. USENIX Association. ISBN 9781931971324.

[21] P. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. doi: 10.1109/SFCS.1994.365700.

[22] H. Nejatollahi, N. Dutt, S. Ray, F. Regazzoni, I. Banerjee, and R. Cammarota. Post-quantum lattice-based cryptography implementations: A survey. *ACM Comput. Surv.*, 51(6), jan 2019. ISSN 0360-0300. doi: 10.1145/3292548. URL https://doi.org/10.1145/3292548.

[23] R. Bost. Sophos: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1143–1154, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341394. doi: 10.1145/2976749.2978303. URL https://doi.org/10.1145/2976749.2978303.

[24] R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1465–1482, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468. doi: 10.1145/3133956.3133980. URL https://doi.org/10.1145/3133956.3133980.

[25] A. Bossuat, R. Bost, P.-A. Fouque, B. Minaud, and M. Reichle. Sse and ssd: Page-efficient searchable symmetric encryption. Cryptology ePrint Archive, Report 2021/716, 2021. https://ia.cr/2021/716.

[26] Y. Wang, S.-F. Sun, J. Wang, J. K. Liu, and X. Chen. Achieving searchable encryption scheme with search pattern hidden. *IEEE Transactions on Services Computing*, pages 1–1, 2020. doi: 10.1109/TSC.2020.2973139.

[27] L. Xu, X. Yuan, R. Steinfeld, C. Wang, and C. Xu. Multi-writer searchable encryption: An lwe-based realization and implementation. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, Asia CCS '19, page 122–133, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367523. doi: 10.1145/3321705.3329814. URL https://doi.org/10.1145/3321705.3329814.

[28] R. Behnia, M. O. Ozmen, and A. A. Yavuz. Lattice-based public key searchable encryption from experimental perspectives. *IEEE Transactions on Dependable and Secure Computing*, 17(6):1269–1282, 2020. doi: 10.1109/TDSC.2018.2867462.

[29] R. Meng, Y. Zhou, J. Ning, K. Liang, J. Han, and W. Susilo. An efficient key-policy attribute-based searchable encryption in prime-order groups. In T. Okamoto, Y. Yu, M. H. Au, and Y. Li, editors, *Provable Security*, pages 39–56, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68637-0.

[30] J. Hao, J. Liu, H. Wang, L. Liu, M. Xian, and X. Shen. Efficient attribute-based access control with authorized search in cloud storage. *IEEE Access*, 7:182772–182783, 2019. doi: 10.1109/ACCESS.2019.2906726.

[31] S. Barker and M. Rothmuller. The internet of things: Consumer, industrial & public services 2020-2024. Technical report, Juniper Research, 2020. URL https://www.juniperresearch.com/researchstore/devices-technology/internet-of-things-iot-data-research-report/subscription/consumer-industrial-public-services.

[32] N. Lu, N. Cheng, N. Zhang, X. Shen, and J. W. Mark. Connected vehicles: Solutions and challenges. *IEEE Internet of Things Journal*, 1(4):289–299, 2014. doi: 10.1109/JIOT.2014.2327587.

[33] S. Gräfling, P. Mähönen, and J. Riihijärvi. Performance evaluation of ieee 1609 wave and ieee 802.11p for vehicular communications. In *2010 Second International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 344–348, 2010. doi: 10.1109/ICUFN.2010.5547184.

[34] F. J. Ros, J. A. Martinez, and P. M. Ruiz. A survey on modeling and simulation of vehicular networks: Communications, mobility, and tools. *Computer Communications*, 43:1–15, 2014. ISSN 0140-3664. doi: https://doi.org/10.1016/j.comcom.2014.01.010. URL https://www.sciencedirect.com/science/article/pii/S0140366414000267.

[35] *ns-3 Tutoria*. NS3, ns-3-release edition.

[36] M. Mezzavilla, M. Zhang, M. Polese, R. Ford, S. Dutta, S. Rangan, and M. Zorzi. End-to-end simulation of 5g mmwave networks. *IEEE Communications Surveys Tutorials*, 20(3):2237–2263, 2018. doi: 10.1109/COMST.2018.2828880.

[37] J. A. Akinyele, M. D. Green, and A. D. Rubin. Charm: A framework for rapidly prototyping cryptosystems. Cryptology ePrint Archive, Paper 2011/617, 2011. URL https://eprint.iacr.org/2011/617. https://eprint.iacr.org/2011/617.

[38] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978. ISSN 0001-0782. doi: 10.1145/359340.359342. URL https://doi.org/10.1145/359340.359342.

[39] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. doi: 10.1109/TIT.1976.1055638.

[40] A. Lewko and B. Waters. Decentralizing attribute-based encryption. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 568–588, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-20465-4.

[41] S. Hohenberger and B. Waters. Attribute-based encryption with fast decryption. In K. Kurosawa and G. Hanaoka, editors, *Public-Key Cryptography – PKC 2013*, pages 162–179, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-36362-7.

[42] M. Abe, J. Groth, M. Ohkubo, and T. Tango. Converting cryptographic schemes from symmetric to asymmetric bilinear groups. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, pages 241–260, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44371-2.

[43] Ieee guide for wireless access in vehicular environments (wave) architecture. *IEEE Std 1609.0-2019 (Revision of IEEE Std 1609.0-2013)*, pages 1–106, 2019. doi: 10.1109/IEEESTD.2019.8686445.

[44] R. Halili, M. Weyn, and R. Berkvens. Comparing localization performance of ieee 802.11p and lte-v v2i communications. *Sensors*, 21(6), 2021. ISSN 1424-8220. doi: 10.3390/s21062031. URL https://www.mdpi.com/1424-8220/21/6/2031.

[45] G. Carneiro, P. Fortuna, and M. Ricardo. Flowmonitor - a network monitoring framework for the network simulator 3 (ns-3). 01 2009. doi: 10.4108/ICST.VALUETOOLS2009.7493.