Università degli Studi di Perugia

Engineering Department

Master's Degree Course
in Computer Engineering and Robotics

Master Thesis

# Securing databases using Attribute Based Encryption and Shamir's Secret Sharing

Graduand:
Martina Palmucci

Promoter:
Prof. Gianluca Reali

Academic Year 2021–2022

*In theory, there is no difference between theory and practice.*
*But, in practice, there is.*

*Benjamin Brewster*

# Contents

# Acknowledgements

This work has been supported by my promoter and professor Prof. Gianluca Reali. I must thank him for believing in me from the beginning and giving me the opportunity to have an extraordinary experience of research abroad. Throughout this physical and metaphorical journey, his professionalism has represented a main and constant point of reference for me.

I would also like to show gratitude to Prof. An Braeken and Prof. Kris Steenhaut from the Vrije Universiteit Brussel (VUB). They welcomed and trusted me even though they didn't know me. They represented constant points of reference not only in good times but also in difficult situations. Their support, advice and feedback have been fundamental to improve my work and knowledge. The work they do in research and with students is exceptional. It would be impossible not to admire them.

Words cannot express my gratitude to my supervisors Ruben De Smet and Thibaut Vandervelden for their invaluable patience and feedback, technical and moral support. They have always been there to support and guide me. We could laugh in good times and but also in bad ones because they always know how to produce a positive vibe in the office whatever the circumstance. They taught me a lot and generously provided knowledge and expertise. They have been, and will continue to be, guides and models for me as professionals and as people.

Additionally, this endeavour would not have been possible without the support of my family. Their belief in me and constant love have kept my spirits and motivation high during this process. I thank my mom for teaching me to be brave and believe in my dreams. Thanks to my dad for teaching me the priceless importance of simple things. Thanks to my little brother for being so similar and so different from me, for always helping me directly or indirectly to see things from a different point of view, often fresher and lighter.

Lastly but not least, I am grateful to my old and recent friends, and my Italian and foreign colleagues, for their friendship, love, and emotional support. Thank you for sharing this journey with me.

# Abstract

LumoSQL is an ambitious project to bring encrypted storage modes to SQLite embedded databases, while keeping the familiar SQL GRANT syntax. SQLite is embedded, meaning anyone with access to the device has access to SQLite data files and applications communicate with SQLite via local function calls. These two reasons mean that at-rest encryption of the SQLite data files is the only possibility.

Whole-file encryption implies that as soon as the database is unlocked it is fully available to the device, and all any permitted application can do anything it likes with the plain text, such as send it over a network. In contrast, LumoSQL implements persistent data privacy on the device (at-rest encryption) and off the device when subsets of data are sent over a network (per-row encryption). LumoSQL also implements fine-grained data protection, with access rights discriminated by attributes the user possesses including `SELECT`, `UPDATE`, `INSERT`, `DELETE`, `CREATE`. The notion of fine-grained permissions which persist regardless of where a database row is copied is novel, and is the most ambitious of LumoSQL's goals.

We implement Attribute-based Encryption Shamir's Secret Sharing (ABE-SSS), an encryption system capable of achieving these objectives as part of the LumoSQL project. The implementation is written in the Rust programming language, which has suitable features for cryptographic applications. ABE-SSS implements the interface of an Attribute-Based Encryption (ABE) system. ABE allows encrypting a resource against *attributes* instead of concrete users. Only users that possess a correct combination of attributes will be able to decrypt the resource. Concretely, the ABE system is built on Shamir's Secret Sharing (SSS). The latter enables the distribution of the secret cryptographic keys according on the attributes defined by ABE.

Finally, the system is evaluated based on its functionality, memory, and the number of operations necessary to acquire and store the output. The functions required by an encryption system are met. The ABE standards are met with the exception of collision resistance, which is still in the works. The memory required to save the result represents a significant overhead that can be reduced by optimizing the ABE system management, for which a feasible solution has already been proposed. The complexity with which the link between system attributes is implemented determines the number of operations.

# Part I

# Introduction

# Chapter 1

# Problem statement

## 1.1 Data protection

Nowadays, the amount of data stored on our digital devices is huge and the tendency is rising. The type of data traveling on digital medium can be varied. In the case of data protection and attacks, those of greatest interest are data that can jeopardize the freedom, safety and health of an individual. In fact, among the digital data, it is easy to find data that allow direct and indirect identification, such as personal data and images, or the national code, the IP address and the license plate number. There are sensitive data, that is, those that reveal racial or ethnic origin, religious or philosophical beliefs, political opinions, trade union membership, relating to health or sexual life. Not to mention the biometric, financial and judicial data.

In this scenario, private data is becoming a greater concern especially due to recent data breaches, thefts and cyber attacks. Indeed, they can cause devastating damages. Entities and organizations must protect their data proactively and update their safeguards on a regular basis. The rise in these crimes, as well as their significant social and economic impact, has prompted institutions to speak out on the issue. Recent legislation, notably the GDPR [1] and the CDR [2] started to mandate technical measures for data protection.

Cyberattacks that compromise the security of sensitive data can come from both inside and outside a system. Once a hacker has access to valuable information on a server, they are likely to steal data from it. Then they use the data to request a ransom from the company or the organization they targeted, exploit the data or other financial benefits. People are increasingly aware of data security and privacy and want their data to be safeguarded and used only as required. As an example, we report the case of Baltimore. The Baltimore ransomware attack [3] in May 2019 is a well-known example of a ransom note hacking attack. A form of ransomware known as RobbinHood infiltrated the networks and data of the whole city of Baltimore, Maryland. Another big city in the United States, Atlanta, had experienced a similar attack the previous year [3]. In the case of Baltimore, the ransom requested is 13 bitcoins (about $76 280) in exchange for access credentials. In the case of Atlanta, they demanded for $51 000 via Bitcoin. Both cities had to pay far larger sums to restore their systems.

This is not the only instance in which data security is extremely important. So far, we have discussed incidents of external attacks. Internal attacks, on the other hand, are not uncommon. One of the most common types of internal attacks happens in the following circumstances. When an application on a modern mobile device requires access to a system resource, the operating system usually guards this resource through an application programming interface (API). The operating system is able to allow or reject access to the resource this way. However, badly designed APIs or bad implementations of interfaces may fail to guard the resource correctly. In some cases, this may result in a privacy loophole [4]. Cryptographically enforcing such access rights may then provide an additional layer of protection. Among insider attacks, how can we not remember the Facebook–Cambridge Analytica data scandal [5]? In the 2010s, Facebook exposed 87 million users to British consulting firm Cambridge Analytica, predominantly to be used for political advertising. It was also not even the first case of data leakage from the American multinational. Facebook today says that it was its own responsibility, it will solve the problem and enhance the privacy level of their users. However, episodes of Meta platform exploitation seem seem to keep happening but they are trying to be blacked out.

In both of these categories of unsafe conditions, an encrypted database offers better data security. The database data is transformed into "cipher text" (illegible text) by the cryptographic procedure using an algorithm. Security attacks are inevitable, but with improved data security and encryption techniques, hackers may not be able to analyze or decrypt data to better understand it. If you have an encrypted database, an attacker needs a technique to decrypt the data. The complexity of the cipher and the techniques used to produce the encrypted data will determine how far the security of the data can go. As a result, cybersecurity professionals have long identified cryptography as a critical component of cybersecurity preparedness.

Furthermore, to enhance security against internal threats, researchers recently described a new method for using public key cryptography to increase the security of underlying data. Data decryption occurs only using so-called attribute-based encryption when a predetermined set of user attributes coincides with those of the ciphertext. This implies that the person (or machine) checking the data can specify criteria for the person (or machine) receiving the data to make a decision. For example, data can only be decrypted on a certain day, at a certain time, or in a certain place. In other cases, the recipient must have a high-level security clearance to decrypt the message. The most notable advantage of attribute-based encryption over traditional cryptographic algorithms is its flexible and granular access control. Additionally, it is independent of key management or sharing algorithms and has built-in defenses against collision attacks, which are a means of weakening encryption. In addition, attribute-based encryption simplifies data sharing by limiting access only to recipients who meet certain criteria.

## 1.2 LumoSQL

LumoSQL [6] is an ambitious project that focuses on the importance of personal data security. Its purpose is to bring a new encrypted storage option to SQLite embedded databases while retaining the standard SQL-GRANT syntax.

In order to fully appreciate the LumoSQL project and its technological implications, let us first take a step back and try to understand first why SQLite was chosen. The limitations of this system are then discussed, followed by LumoSQL's solutions.

SQLite [7] is a C-language software library that implements an ACID-type SQL DBMS that can be integrated within applications. It is embedded precisely in this sense: it resides within an application software rather than a server. Its inventor, D. Richard Hipp, released it public domain, allowing it to be used without restriction. It is the most often used SQL database software. SQLite offers a number of extensions that allow for encrypted storage. Every extension implements the following security model: unrelated applications should not be allowed to access these data. SQLite Encryption Extension (SEE) [8] is one of the most well-known and important extensions. A SQLite version with SEE can read and write ordinary database files written using a public domain version of SQLite, as well as read and write encrypted ones. Each database file can have a unique encryption key.

However, none of these include a protection against internal database attacks. In other words, if an entity can access the database, it will have unlimited access to all of its resources. LumoSQL provides a novel technology that allows for more granular database encryption. The numerous roles that a user assumes within the system to which LumoSQL applies would provide the granularity. Only those who meet the requirements can decrypt the resources saved in the database. Even if you have complete access to the database, you will not be able to interpret it until you meet the necessary conditions.

Furthermore, it seeks to accomplish so by retaining the existing syntax in order to minimize developers' effort in learning a new syntax and maximize adherence to the project.

## 1.3 Research Topic

The objective of this research project is defined as the creation of an Attribute-Based Encryption (ABE) system, whose key generation is based on Shamir's Secret Sharing (SSS) algorithm.

ABE is an asymmetric cryptographic primitive in which the ciphertext and the user's secret key are both determined by attributes. Only if the set of attributes of the user key matches the attributes of the ciphertext can a ciphertext be decrypted in such a system. Since LumoSQL tends to retaining standard syntax, GRANT-style privileges may be implemented in terms of ABE.

SSS is one of the first cryptographic secret sharing techniques [9]. It allows you to divide a secret number into a certain number of parts and, given a quota threshold, we can recover the secret number. Shamir's Secret Sharing is used to distribute a secret accessible by entities that have one or more attributes.

ABE and SSS work together to guarantee resource confidentiality. An attribute based key protects each resource. The sharing technique can only recover the correct key to decode the resource if you have the necessary attributes.

## 1.4 State of the art

Database encryption is yet an emerging and evolving topic. It is defined as a method that utilizes an algorithm to convert data saved in a database into ”cipher text” that is incomprehensible until first decrypted. In particular, we focus on data at rest, that is commonly described as “inactive” data that is not being modified or transferred over a network at the time.

We differentiate three types of database encryption approaches.

The first category is concerned with the transparent database encryption (TDE), which is used to encrypt an entire database. TDE is included in Microsoft SQL Server 2008, 2008 R2, 2012, 2014, 2016, 2017, and 2019 [10]. IBM offers TDE as part of Db2 as of version 10.5 fixpack 5 [11]. Oracle requires the Oracle Advanced Security option for Oracle 10g and 11g to enable TDE [12]. MySQL enables TDE data encryption at rest with MySQL Enterprise TDE [13].

The second category is concerned with the column-level encryption referred to relational databases. When opposed to encryption systems that encrypt a whole database, such as TDE, the ability to encrypt individual columns allows column-level encryption to be substantially more versatile. However, dncrypting distinct columns with different unique keys in the same database might reduce database performance as well as the speed with which the database’s contents can be indexed or searched. Adoption of this sort of data protection is substantially lower in this scenario than in the prior one. IBM [14] and Oracle are among those adopted column-level encryption.

The third category is field-level encryption. Nobody has yet implemented this kind of encryption. The technological advancements in this subject relating to a field-level concern primarily the query operation. Experimenting with offering database operations (such as searching or arithmetic operations) on encrypted fields without the requirement to decode them is underway. MongoDB [15] is one example of this.

This work aims to lay the groundwork for the development of a field-encryption database. It is thus a completely new and cutting-edge technology, unlike anything now available on the market.

## 1.5 Overview

This thesis is divided into two sections. The first section is dedicated for more theoretical discussions. In instance, we have a discussion of current cryptography and so on in Chapter 2. As a result, it is an essential chapter for comprehending all of the factors underpinning the security of this system and newer systems. The encryption technique employed is shown in Chapter 3. It is a hybrid system that requires prior knowledge of both symmetric and asymmetric cryptography. Instead, Chapter 4 introduces SSS, a cryptographic scheme for securing private information that relies on slitting rather than encryption. With Chapter 5, we move on to the second section of the thesis, which explains the solutions used to finish the outcome. The tools utilized are discussed in detail in Chapter 5. The system’s functionality is detailed in Chapter 6 and Chapter 7. Finally, findings are shown in Chapter 8.

# Chapter 2

# Modern cryptography and ECC

Diffie and Hellman created public key cryptography[1] in 1976 [17], but they were unable to find a viable way to apply their notion. Rivest, Shamir, and Adleman [18] created the first viable public key cryptosystem the following year. The Rivest–Shamir–Adleman (RSA) cryptosystem's security is based on the difficulty of factoring huge integers. Diffie and Hellman, on the other hand, described a key exchange technique whose security is based on the discrete logarithm problem in $\mathbb{F}_q^*$, and ElGamal later devised a public key cryptosystem based on the same fundamental problem. In 1985, Koblitz [19] and Miller [20] proposed substituting the finite field $\mathbb{F}_q$ with an elliptic curve $E$ in the hope that the discrete logarithm issue in the elliptic curve group $E(\mathbb{F}_q)$ would be more difficult to solve than the discrete logarithm problem in the multiplicative group $\mathbb{F}_q^*$. In fact, with the advent of increasingly powerful computer resources, traditional systems have become risky unless longer cryptographic keys are used. Their insight resulted in the development of Elliptic-curve Cryptography (ECC).

## 2.1 Elliptic curves

In most cases, the graph of an elliptic curve $E$ can be represented by the points that satisfy the equation:

$$y^2 = x^3 + Ax + B \tag{2.1}$$

where $A, B$ are constants. Eq. (2.1) will be referred to as the Weierstrass equation for an elliptic curve. We must identify which set $A$, $B$, $x$, and $y$ belong to. They are typically regarded as elements of a field, namely a set on which addition, subtraction, multiplication, and division are defined and behave as the corresponding operations on real numbers $\mathbb{R}$ and rational numbers $\mathbb{Q}$ do. Let

[1]James Ellis first revealed public key cryptography in 1969, but his finding was hidden by the British government and not declassified until after his death in 1997. Williamson and Cocks, two additional British government employees, are the original creators of the Diffie-Hellman key exchange algorithm and the RSA public key cryptosystem, respectively, although their discoveries were also classified [16].

$K$ be a field, then we say that $E$ is defined over $K$ when $A, B \in K$. Furthermore, the Weierstrass Eq. (2.1) holds for fields with characteristic other than 2 and 3. Otherwise, the more general form

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{2.2}$$

where $a_1, ..., a_6$ are constants, is useful when working with fields of characteristic 2 and characteristic 3. We call Eq. (2.2) the generalized Weierstrass equation. In [21] it is shown how to derive Eq. (2.1) from equation Eq. (2.2). In most of this thesis, we will develop the theory using the Weierstrass equation.

Secondly, it is beneficial to add a point at infinity, commonly denoted by $\infty$, to an elliptical curve for technical reasons. To make this notion more rigorous, an infinite point is defined as a point on a curve that meets particular mathematical requirements (see Theorem 2.1.1). When the field of definition is real numbers, we can think of the point at infinity as being at the top and bottom of the y-axis. For example, when a line is vertical (i.e., $x = constant$), it is said to pass through exactly $\infty$. The top and bottom of the $y$-axis are the same point. In fact, we have arranged for two vertical lines to intersect in $\infty$. They should meet at the top and bottom of the $y$-axis for symmetry. However, two lines should only intersect at one point, therefore the "top" and "bottom" must be the same. That is, we imagine the ends of the $y$-axis wrapping around and meeting (perhaps someplace in the back of the page) at the point $\infty$. This may appear strange. However, if we are dealing with a field other than real numbers, such as a finite field, there may be no significant ordering of the components, making defining a top and bottom of the $y$-axis impossible.

Thirdly, we do not allow multiple roots of the cubic. That is to say, we assume that

$$4A^3 + 27B^2 \neq 0 \tag{2.3}$$

This assumption permits us to work with not singular curves. They are useful for dealing with the discrete logarithm problem, which we shall solve with elliptic curves. However, the case when the roots are not distinct is still relevant for some problems that we will not address, i.e. factorization problem and primality testing.

Finally, by combining all of the above facts, we can provide a formal definition of an elliptic curve [21].

**Definition 2.1.1** (Elliptic curve)**.** An elliptic curve $E$ defined over a field $K$, with characteristics other than 2 and 3, is a set of points represented by

$$E(L) = \{\infty\} \cup \{(x, y) \in K \times K \mid y^2 = x^3 + Ax + B\}. \tag{2.4}$$

where $A, B \in K$ so that it is not singular ($4A^3 + 27B^2 \neq 0$).

**Example 2.1.1.** In Fig. 2.1 we can see three examples of elliptic curves whereas Fig. 2.2 shows two singular cubic curves, where $\Delta := 4A^3 + 27B^2$. Both figures consider the field of real numbers [16].

### 2.1.1 The group law

Most of the interesting features and applications of elliptic curves are due to the fact that the points on an elliptical curve constitute a so-called "abelian

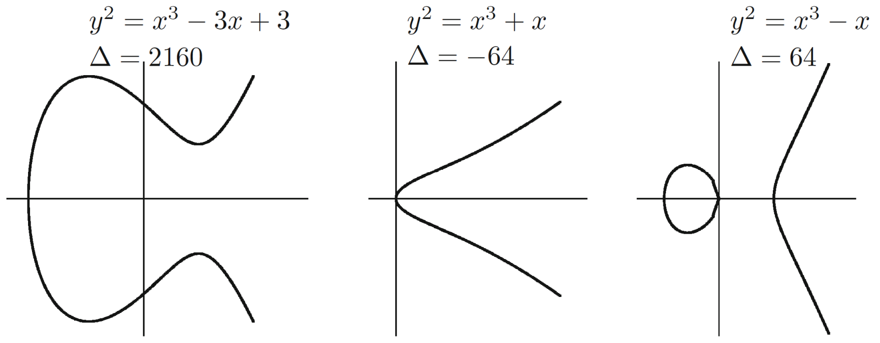

Figure 2.1: Three elliptic curves.

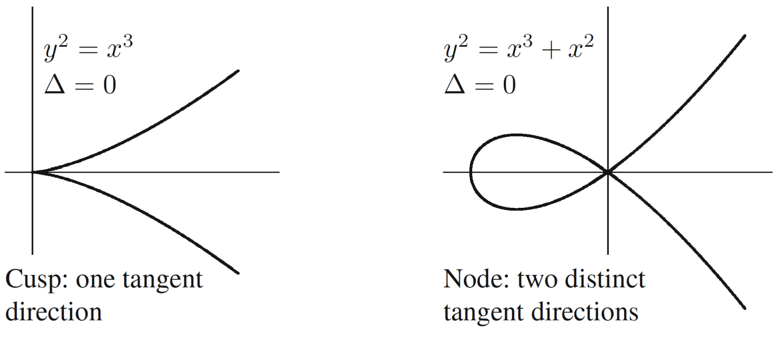
$y^2 = x^3$
$\Delta = 0$
Cusp: one tangent direction
$y^2 = x^3 + x^2$
$\Delta = 0$
Node: two distinct tangent directions

Figure 2.2: Two singular cubic curves.

group". The group of elliptic curves over finite fields, in particular, offers many interesting cryptographic properties and applications. However, before we get into the applications, let us take a closer look at what an abelian group is and why elliptic curves can be defined as such. Therefore, we are now approaching the group law on an elliptic curve $E$.

## Adding Points on an Elliptic Curve

Begin with two points $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ on an elliptic curve $E$, which is defined by the Eq. (2.1). Create a new point $P_3$ as shown below. Draw a line $L$ through $P_1$ and $P_2$. $L$ intersects $E$ in a third point $P_3^{'}$ as we will see below. $P_3$ is obtained by reflecting $P_3^{'}$ across the $x$-axis.

$$P_1 + P_2 = P_3 \tag{2.5}$$

is the definition. This is not the same as adding the point coordinates. This action may be denoted by $P_1 +_E P_2$, but we use the shorter notation because we will never add points by adding coordinates.

First, assume that $P_1 \neq P_2$ and that neither point is $\infty$. Draw the line $L$

through $P_1$ and $P_2$. The slope is

$$m = \frac{y_2 - y_1}{x_2 - x_1}. \tag{2.6}$$

L is vertical if $x_1 = x_2$. We will get to this later, so let us suppose $x_1 \neq x_2$. The equation of $L$ becomes

$$y = m(x - x_1) + y_1. \tag{2.7}$$

Substitute Eq. (2.7) into Eq. (2.1) to get

$$\left(m\left(x - x_1\right) + y_1\right)^2 = x^3 + Ax + B \tag{2.8}$$

and locate the intersection with $E$. This can be rewritten as

$$0 = x^3 - m^2x^2 + \cdots \tag{2.9}$$

by moving everything to the right. The three roots of this cubic correspond to the three points of intersection of $L$ with $E$. In general, solving a cubic is difficult, but in this case, we already know two of the roots, $x_1$ and $x_2$, because $P_1$ and $P_2$ are points on both $L$ and $E$. Therefore, if we have a cubic polynomial with roots $r$, $s$, and $t$, then

$$\begin{aligned} x^3 + ax^2 + bx + c &= (x - r)(x - s)(x - t) \\ &= x^3 - (r + s + t)x^2 + \cdots . \end{aligned} \tag{2.10}$$

As a result, $r + s + t = a$. If we know two roots, $r$ and $s$, we can calculate the third as

$$t = -a - r - s. \tag{2.11}$$

In our case, we see from Eq. (2.9) and Eq. (2.10) that $a = -m^2$. Also, we can assume that $r = x_1$, $s = x_2$ are the known roots and $t = x$ is the unknown one. Thus, substituting in Eq. (2.11)

$$x = m^2 - x_1 - x_2 \tag{2.12}$$

is obtained. The Eq. (2.12) and the Eq. (2.7) define the $x$-coordinate and the $y$ coordinate of the point $P_3^{'}$. Now, reflect across the $x$-axis to obtain the point $P_3 = (x_3, y_3)$:

$$x_3 = m^2 - x_1 - x_2, \qquad y_3 = m(x_1 - x_3) - y_1. \tag{2.13}$$

When $x_1 = x_2$ but $y_1 \neq y_2$, the line between $P_1$ and $P_2$ is a vertical line that crosses $E$ in $\infty$. Reflecting $\infty$ across the $x$-axis results in the same point $\infty$ (which is why $\infty$ is at the top and bottom of the $y$-axis). Therefore, in this situation, $P_1 + P_2 = \infty$.

Now, consider the situation when $P_1 = P_2 = (x_1, y_1)$. When two points on a curve are relatively near, the line connecting them approximates a tangent line. Therefore, when two points coincide, we consider the line $L$ connecting them to be the tangent line. We may find the slope $m$ of $L$ using implicit differentiation:

$$2y\frac{\mathrm{d}y}{\mathrm{d}x} = 3x^2 + Ax, \quad \text{so} \quad m = \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{3x_1^2 + A}{2y_1} \tag{2.14}$$

If $y_1 = 0$, the line is vertical, and we again set $P_1 + P_2 = \infty$. As an aside, if $y_1 = 0$, the numerator $3x_1^2 + A \neq 0$. Therefore, assume that $y_1 = 0$. The equation of $L$ is Eq. (2.7), as previously stated. The cubic equation Eq. (2.9) is obtained again. We only know one root this time, $x_1$, but it is a double root because $L$ is tangent to $E$ at $P_1$. As a result, continuing as previously, we get

$$x_3 = m^2 - 2x_1, \qquad y_3 = m(x_1 - x_3) - y_1. \tag{2.15}$$

Finally, let us say $P_2 = \infty$. The line through $P_1$ and $\infty$ is a vertical line that intersects $E$ at the point $P_1^{'}$, which is the $x$-axis reflection of $P_1$. We return to $P_1$ by reflecting $P_1^{'}$ over the $x$-axis to produce $P_3 = P_1 + P_2$. As a result, for all points $P_1$ on $E$, $P_1 + \infty = P_1$. Of course, this is expanded to include $\infty + \infty = \infty$.

Ultimately, we can conclude that, in the sense that we have explained with the preceding considerations, given two points $P_1$ and $P_2$, distinct or coincident in $E$, there is a third $P_3^{'}$ of $E$ aligned with them. We are so very close to the definition of the group law on $E$, which we will denote by additive notation. It may be tempting to define the sum $P_1 + P_2$ of two points $P_1$ and $P_2$ of $E$ just like the third point of $E$ aligned with $P_1$ and $P_2$. This is not exactly the right idea. Instead, it is necessary to take $\infty$ as the neutral element, i.e. zero, of the group, and define the sum $P_1 + P_2$ as follows:

- first take the third point $P_3^{'}$ of $E$ aligned with $P_1$ and $P_2$;
- $P_1 + P_2$ is defined as the third point $P_3$ of $E$ aligned with $P_3^{'}$ and with $\infty$.

In other words, if we think of an elliptic curve defined by an equation of the type Eq. (2.1), $P_1 + P_2$ is the symmetrical point with respect to the $x$ axis of the third point $P_3^{'}$ of $E$ aligned with $P_1$ and $P_2$ (see Fig. 2.3). In this way, the opposite of each point $P$ is its symmetrical with respect to the $x$ axis.

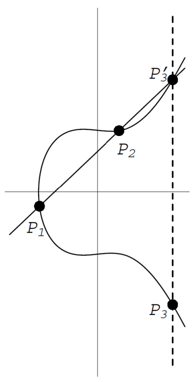

Figure 2.3: Adding points on an elliptic curve.

Let us summarize the preceding discussion:

**Definition 2.1.2** (Additive group law for elliptic curves)**.** Let $E$ be an elliptic curve over a field $K$ defined as in Definition 2.1.1. Consider the binary operation $+$ in $E$

$$E \times E \longrightarrow E, \qquad (P_1, P_2) \longrightarrow P_3$$

where $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are points on $E$ with $P_1, P_2 \neq \infty$, and $P_1 + P_2 = P_3 = (x_3, y_3)$ defined as follow:

1. If $x_1 \neq x_2$, then
$$x_3 = m^2 - x_1 - x_2, \qquad y_3 = m(x_1 - x_3) - y_1,$$
where $m = \frac{y_2 - y_1}{x_2 - x_1}$.
2. If $x_1 = x_2$ but $y_1 \neq y_2$, then $P_1 + P_2 = \infty$.
3. If $P_1 = P_2$ and $y_1 \neq 0$, then
$$x_3 = m^2 - 2x_1, \qquad y_3 = m(x_1 - x_3) - y_1,$$
where $m = \frac{3x_1^2 + A}{2y_1}$.
4. If $P_1 = P_2$ and $y_1 = 0$, then $P_1 + P_2 = \infty$.

Moreover, define
$$P + \infty = P$$
for all points $P$ on $E$.

It should be noted that if $P_1$ and $P_2$ have coordinates in a field $K$ that includes $A$ and $B$, then $P_1 + P_2$ has coordinates in $K$ as well. As a result of the preceding point addition, $E(K)$ is closed. Let us show that this point addition has some interesting properties.

**Theorem 2.1.1.** The points on an elliptic curve $E$ form an additive abelian group $(E, +)$ with $\infty$ as the identity element. That is, the addition of points on $E$ has the following properties:

- commutativity
$$P_1 + P_2 = P_2 + P_1 \quad \forall P_1, P_2 \in E$$
- existence of identity
$$P + \infty = P \quad \forall P \in E$$
- existence of inverses
$$\forall P \in E \quad \exists P' \in E \quad s.t. \quad P + P' = \infty$$
where the point $P'$ is usually denoted $-P$
- associativity
$$(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3) \quad \forall P_1, P_2, P_3 \in E$$

We shall not demonstrate this theorem. It can, however, be found in [21].

Also, keep in mind that in the Weierstrass equation Eq. (2.1), if $P = (x, y)$, then $-P = (x, -y)$. This is no longer true for the generalized Weierstrass equation Eq. (2.2). If $P = (x, y)$ is on the curve defined by Eq. (2.2), then $-P = (x, -a_1x - a_3 - y)$.

**Successive doubling**

If $P$ is an elliptic curve point and $k$ is a positive integer, then $kP$ represents $P + P + \cdots + P$ (with $k$ summands). If $k$ is negative, then $kP = (-P) + (-P) + \cdots (-P)$, with $|k|$ summands. It is inefficient to add $P$ to itself repeatedly to compute $kP$ for a large integer $k$. Using repeated doubling is substantially faster. For example, in order to compute $19P$, we must first compute $2P$, then

$$\begin{aligned} 4P &= 2P + 2P \\ 8P &= 4P + 4P \\ 16P &= 8P + 8P \\ 19P &= 16P + 2P + P \end{aligned}$$

This method allows us to quickly compute $kP$ for very big $k$, say several hundred digits. When we work with rational numbers, the only problem is that the size of the coordinates of the points grows quite quickly. However, we are more interested in finite field, such as $\mathbb{F}_q$. In elliptic curves over a finite field these is not even this issue because we can repeatedly reduce $\mod q$, keeping the numbers involved relatively small. It is worth noting that the associative law allows us to perform these computations without regard for the sequence in which the summands are combined.

## 2.2 Elliptic curves over finite fields

In this chapter, we will examine the fundamental theory of elliptic curves on finite fields [16], [21]–[23]. Not only are the results interesting in their own right, but they are also the starting point for major cryptographic applications, some of which are specifically utilized in this thesis.

### 2.2.1 Finite fields

Finite fields have an important role in many areas of mathematics and computer science, including cryptography.

Finite fields can be thought of as a limited collection of numbers with limited representation. This may sound like a play on words, but it is one of the essential qualities that makes finite fields a good environment for cryptography. Coupled with that, we must remember the property of satisfying certain basic rules when doing specific operations such as multiplication, addition, subtraction, and division. This is not obvious for any set of numbers, whether finite or not.

Mathematically, a finite field, also known as a Galois field, is a field with a finite number of elements. The number of elements in a finite field is referred to as its order. A finite field of order $q$ exists if and only if $q$ is a prime power $p^k$ (where $p$ is a prime number and $k$ is a positive integer). Adding $p$ copies of every element in a field of order $p^k$ always results in zero; thus, the field's characteristic is $p$.

All fields of order $q = p^k$ are isomorphic. Thus, a field cannot have two different finite subfields of the same order. All finite fields with the same order may thus be identified, and they are unambiguously denoted $\mathbb{F}_q$, $F_q$ or $GF(q)$, where the initials $GF$ stand for "Galois field".

Fields of prime order are the simplest examples of finite fields: for each prime number $p$, the prime field of order $p$, $\mathbb{F}_p$, can be formed as the integers modulo $p$, $\mathbb{Z}/p\mathbb{Z}$. The elements of the prime field of order $p$ can be represented by integers ranging from $0$ to $p-1$.

### 2.2.2 Properties

Assume $\mathbb{F}_q$ is a finite field and $E(\mathbb{F}_q)$, abbreviate $E$, is an elliptic curve defined over $\mathbb{F}_q$. That is, the two requirements listed below are met. An elliptic curve over the finite field consists of:

- a set of integer coordinates $(x, y)$, such that $0 \leq x, y < q$;
- staying on the elliptic curve: $y^2 = x^3 + Ax + B \mod q$, according to the Weierstrass Eq. (2.1).

**Example 2.2.1.** Let us take the elliptic curve with equation $y^2 = x^3 + 7$ over the finite field $\mathbb{F}_{17}$. It appears to be a set of points in a $q \times q$ square matrix made up of just integer numbers. It looks like in Fig. 2.4.

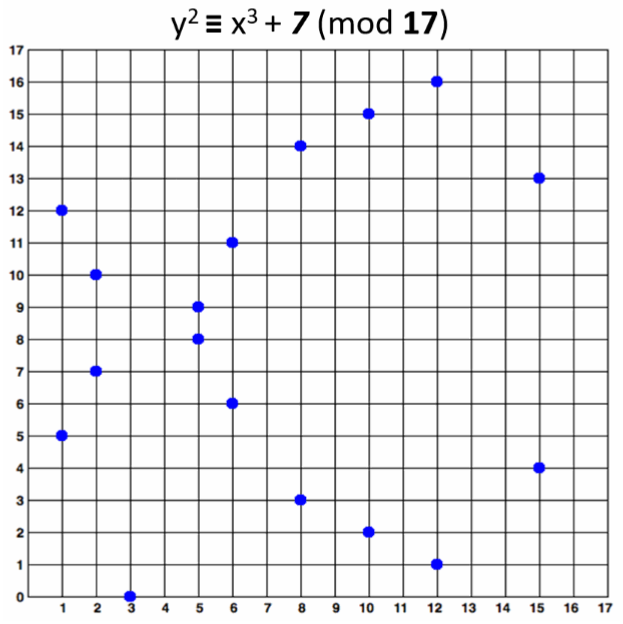

Figure 2.4: Example of elliptic curve over $\mathbb{F}_{17}$.

Obviously, the curve seen in the previous Ex. 2.2.1 is "educational". The number of elements in the finite field $\mathbb{F}_{17}$ is too small to be applied in cryptography.

### Number of elements

Initially, we wish to estimate the number of points in $E(\mathbb{F}_q)$, also called "order" of the group, in order to begin making some observations about the nature of this algebraic structure. The group $E(\mathbb{F}_q)$ is finite because there are only a finite number of pairs $(x, y)$ with $x, y \in \mathbb{F}_q$. Because each $x$ value gives at most two $y$ values, a trivial upper bound is

$$E(\mathbb{F}_q) \leq 2q + 1. \tag{2.16}$$

However, because a "randomly chosen" quadratic equation has a 50% probability of being solved in $\mathbb{F}_q$, we expect the correct order of magnitude to be $q$, not $2q$. The following result, proposed by E. Artin in his thesis and demonstrated by Hasse in the 1930s, demonstrates that this heuristic reasoning is true.

**Theorem 2.2.1** (Hasse)**.** Assume $E(\mathbb{F}_q)$ is an elliptic curve defined over a finite field. Then

$$|\#E(\mathbb{F}_q) - q - 1| \leq 2\sqrt{q} \tag{2.17}$$

This theorem is proved in [16]. Hasse's theorem provides a bound for the number of points in $E(\mathbb{F}_q)$, but no viable procedure for finding $\#E(\mathbb{F}_q)$ when $q$ is huge.

Other theories were then developed to determine the cardinality of the additive group of elliptic curves more accurately. However, this outcome looks to be adequate. From Hasse Theorem 2.2.1, the hypothesis of an analogy in terms of number of elements with the multiplicative group $\mathbb{F}_q^*$ can already be advanced. In this regard, the two groups resemble one other. In cryptography, group cardinality is an important feature. In fact, its components are the core foundation of a system's security. A finite cardinality is desired because we aim to work in a constrained system in terms of both time and space, such as computers and human life. Despite this, the system cannot abandon its complexity because it is a safety feature. As a result, the cardinality must be finite but sufficiently high to deter the most obvious attacks, particularly brute force attacks. Brute force attacks make every attempt possible. If the cardinality of the group is low, this strategy is likely to succeed quickly. This should be avoided at all costs.

In conclusion, we note that the group of $\mathbb{F}_q$-points on an elliptic curve and the multiplicative group $\mathbb{F}_q^*$ have analogues. According to Hasse's Theorem 2.2.1, they have roughly the same number of elements. However, the former construction of an abelian group has a significant benefit that explains its utility in cryptography: for a single (big) $q$, there are many alternative elliptic curves and $\#E(\mathbb{F}_q)$ to choose from. Elliptic curves provide a rich source of finite abelian groups. This will be used to our advantage in the following sections.

### Generator of the group

In addition to the number $\#E(\mathbb{F}_q)$ of elements on an elliptic curve defined over $\mathbb{F}_q$, we may be interested in the actual structure of the abelian group. This abelian group is not necessarily cyclic, but it is always the product of two cyclic groups, namely it is finitely generated. That is to say, it is isomorphic either to the additive group of $\mathbb{Z}_n$, the integers modulo $n$, or to a direct sum of cyclic groups, as described below.

**Theorem 2.2.2.** Let $E$ be an elliptic curve over the finite field $\mathbb{F}_q$. Then

$$E(\mathbb{F}_q) \simeq \mathbb{Z}_n \quad \text{or} \quad \mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2} \tag{2.18}$$

for some integer $n \geq 1$, or for some integers $n_1, n_2 \geq 1$ with $n_1$ dividing $n_2$.

This theorem is proved in [16]. It permits us to apply cyclic group theory and finitely generated abelian group theory. The most significant implications is tied to the generation point of a cyclic group.

**Definition 2.2.1** (Cyclic subgroup generated by $g$)**.** In additive notation, the cyclic subgroup generated by $g$ can be formed for every element $g$ in any group $G$ by combining all of its integer multiples:

$$\langle g \rangle = \{kg : k \in \mathbb{Z}\}. \tag{2.19}$$

The *order* of $g$ is equal to the number of elements in $\langle g \rangle$; that is, one element's order is equal to the order of the cyclic subgroup that it forms.

A cyclic group $G$ is one that is the same as one of its cyclic subgroups: $G = \langle g \rangle$ for some element $g$, also known as generator of $G$. For a finite cyclic group $G$ of order $n$ we have $G = \{e, g, 2g, \cdots, (n-1)g\}$, where $e$ is the identity element and $ig = jg$ whenever $i \equiv j \mod n$; in particular $ng = 0g = e$, and $(-1)g = (n-1)g$. Therefore, an elliptic curve $E$ defined over a finite field is always a finite abelian group. If it is also cycle, we can cover all of its elements by picking $g$, one of its generators, and multiplying $g$ by $k$ for some $k \in \mathbb{Z}$. However, as previously stated, an elliptic curve is not always cyclic. We can extend the idea of cyclic groups when it is the direct sum of cyclic groups, namely finitely generated. Because every finitely generated abelian group is a direct sum of cyclic groups, it can be represented as the direct sum of cyclic subgroups formed by various generators $g_i$. That is, when the elliptic curve is isomorphic to $\mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$ (see Theorem 2.2.2), it can be written as $E = \langle g_1 \rangle \oplus \langle g_2 \rangle$. Coupled with the fast doubling operation (see Section 2.1.1), this appears to be a highly important trait for elliptic curve implementation in cryptography systems.

## 2.3 Elliptic-curve discrete logarithm problem

Elliptic curves do not constitute a cryptographic system on their own. Before we can define a cryptosystem, we must first incorporate another critical component, namely a specific mathematical problem that is simple to declare but difficult to solve. In this context, the concept of simple and difficult is characterized in terms of computing resources and computational time.

**Definition 2.3.1** (Elliptic Curve Discrete Logarithm Problem)**.** Given two points $Q$ and $P$, it states that it is hard to find $k \in \mathbb{Z}$ for the equation $kQ = P$ for specific groups, such as cyclic subgroups defined over finite fields.

The Elliptic Curve Discrete Logarithm Problem (ECDLP) makes Elliptic Curves suitable for cryptography. In fact, thanks to the previously discussed features (see Section 2.2.2), calculating the number $P$ from a random value $Q$ is inexpensive. It can be found in $O(\log k \log^3 q)$ bit operations by the method of repeated doubling [22] [21]. The time estimate is not the most accurate. However, we will be satisfied with the estimates obtained by employing the most obvious algorithms for arithmetic in finite fields. We should remark that, aside from the computing advantages provided by the particular structure of the additive abelian group of elliptic curves, the first search for a group was required since the discrete logarithm problem only applies to groups.

On the other hand, finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point is infeasible. That is, given a finite field $\mathbb{F}_q$ with a large $q$ and a specific elliptic curve $E$ with secure parameters, no efficient algorithm exists.

## 2.4 Elliptic-curve cryptography

ECC is a public-key encryption technique that is based on the algebraic structure of elliptic curves over finite fields. When compared to non-EC encryption (based on ordinary Galois fields), ECC allows for shorter keys to guarantee equal security. Thus, ECC has been introduced because it provides security comparable to classical systems while using fewer bits.

A key size of 4096 bits for RSA, for example, provides the same level of security as 313 bits in an elliptic curve system. This means that elliptic curve cryptosystem implementations require smaller chip sizes, lower power consumption, and so on. Though RSA was somewhat faster in some procedures, such as signature verification, the elliptic curve approaches clearly offer significant speed increases in many scenarios [21].

Elliptic curves can be used for key agreement, digital signatures, pseudorandom number generators, and other purposes.

### 2.4.1 Elliptic-curve Diffie-Hellman

Elliptic-curve Diffie–Hellman (ECDH) is a key agreement protocol that allows two parties to create a shared secret across an unsecured channel. Both parties use an elliptic-curve public-private key pair. Then, the shared secret can be used directly as a key or to generate another key. Following that, the key, or the derived key, can be used to encrypt subsequent communications with a symmetric-key cipher.

#### Key establishment protocol

Assume Alice and Bob wish to establish a shared key, but the only channel accessible to them may be eavesdropped on by a third party. The domain parameters must first be decided upon (that is, $(p, a, b, G, n, h)$ in the prime case or $(m, f(x), a, b, G, n, h)$ in the binary case). The meaning of each element in both sets is the following:

- $p$ is the prime number that characterizes the finite field $\mathbb{F}_p$.
- $m$ is the integer number specifying the finite field $\mathbb{F}_{2^m}$.
- $f(x)$ is the irreducible polynomial of degree $m$ defining $\mathbb{F}_{2^m}$.

- $a$ and $b$ are the elements of the finite field $\mathbb{F}_q$ taking part in the equation of elliptic curve.
- $G$ is the point of the curve that will be used as a generator of the points representing public keys.
- $n$ is the prime number whose value represents the order of the point $G$.
- $h$ is the cofactor of the curve, computed as $h = \#E(\mathbb{F}_q)/n$, where $\#E(\mathbb{F}_q)$ is the number of points on the curve.

Each side must also have a key pair that is suitable for elliptic curve cryptography. In other words, each party much has a private key $d$ and a public key $Q$. The private key $d$ is a integer in the interval $[1, n-1]$ that can be randomly selected. The public key $Q$ is the elliptic curve point based on the parameters found by computing $Q = d \cdot G$, that is, the result of adding $G$ to itself $d$ times.

Let $(d_\mathrm{A}, Q_\mathrm{A})$ be Alice's key pair and $(d_\mathrm{B}, Q_\mathrm{B})$ be Bob's key pair. Prior to executing the protocol, each party must know the public key of the other party. Alice computes point $(x_k, y_k) = d_\mathrm{A} \cdot Q_\mathrm{B}$. Bob computes point $(x_k, y_k) = d_\mathrm{B} \cdot Q_\mathrm{A}$. The shared secret is $x_k$ (the $x$ coordinate of the point). Most standardized protocols based on ECDH derive a symmetric key from $x_k$ using some hash-based key derivation function. The shared secret calculated by both parties is equal, because

$$d_\mathrm{A} \cdot Q_\mathrm{B} = d_\mathrm{A} \cdot d_\mathrm{B} \cdot G = d_\mathrm{B} \cdot d_\mathrm{A} \cdot G = d_\mathrm{B} \cdot Q_\mathrm{A} \tag{2.20}$$

Alice first reveals only her public key as information about her key. So, unless that party can solve the ECDLP, no one else can determine Alice's private key (Alice, of course, knows it because she chose it). Bob's private key is also secure. Except for Alice and Bob, no one else can compute the shared secret unless they can solve the ECDH issue.

While the shared secret can be used directly as a key, it is sometimes advantageous to hash the secret in order to remove weak bits caused by the Diffie-Hellman exchange.

#### Static or ephemeral public key

A public key might be either static or ephemeral. A static public key is trusted according to some specified criteria. It can be exchanged via transmission channel along with its validity proof or it can be recovered by third trusted entities. On the other hand, if the public key is ephemeral, it can simply be generated and transmitted over transmission channel. It is only temporary and can't be confirmed. As a consequence, if authentication is necessary, authenticity assurances must be achieved through alternative ways.

## 2.5 Edwards curves and Curve25519

The equation of an Edwards curve over a finite field $K$ is:

$$x^2 + y^2 = 1 + dx^2y^2 \tag{2.21}$$

They are a family of elliptic curves that were studied by Harold Edwards in 2007. In 2006, Bernstein considerably advanced the state-of-the-art in elliptic curve cryptography [24]. The elliptic curve presented in that paper, called Curve25519, doubled the performance of elliptic curve operations, while improving several security properties. Especially through the use of the Edwards form of this curve, called Ed25519, these performance properties come to their right [25]. Later, implementations of this specific elliptic curve improved the performance even further [26], and this curve is on track for being standardized by the IETF [27]. It has quickly gained popularity for new designs, and is currently at the basis of WhatsApp's and Signal's end-to-end encryption [28].

Many cryptographic protocols that are based on the ECDLP require a prime-order group, whereas elliptic curve groups are often compound. For example, Curve25519 has a cofactor of $n = 8$. Using the Decaf technique [29], it is possible to obtain a prime-order group from an elliptic curve group. Applying the Decaf technique to Curve25519 hence yields Ristretto255 [30].

# Chapter 3

# Elliptic Curve Integrated Encryption Scheme

The Elliptic-curve Integrated Encryption Scheme (ECIES) is a specific hybrid encryption scheme invented by Abdalla, Bellare, and Rogaway [31]. ECIES is the best known elliptic curve encryption technique, and as such, it has been incorporated in various standards like SECG SEC-1, ISO/IEC 18033-2, IEEE 1363a and ANSI X9.63.

The term "hybrid" refers to the use of both symmetrical and asymmetrical cryptosystems inside it. The core edge of hybrid systems is based on the combination of the effectiveness of a symmetric-key cryptosystem with the ease of a public-key cryptosystem. Moreover, ECIES adds up ECC benefits, namely the same security level as standard public key systems, such as RSA, employing lower key length.

## 3.1 Functional components of ECIES

Like in any hybrid cryptosystem, it is feasible to see in ECIES scheme two important aspects: the asymmetric Key Encapsulation Mechanism (KEM), used to secure symmetric key material for transmission, and the data encapsulation scheme, which is a symmetric-key cryptosystem. In ECIES, the asymmetric KEM is a Key Agreement (KA) function. The data encapsulation scheme, intead, is made up of the Key Derivation Function (KDF), the Message Authentication Code (MAC) and the symmetric cipher.

Now, we examine each function in depth but we can also start taking a look at Fig. 3.1 and Fig. 3.2 while we are reading. The two figures show the ECIES encryption scheme and the ECIES decryption scheme. First and second steps belong to the KEM whereas the rest represents the data encapsulation scheme.

### 3.1.1 Key-agreement protocol

The ECIES standard do not assume any share secret between the sender and recipient before the communication starts. In order to communicate through a potentially insecure channel, they need to establish a share secret key $k_S$ first.

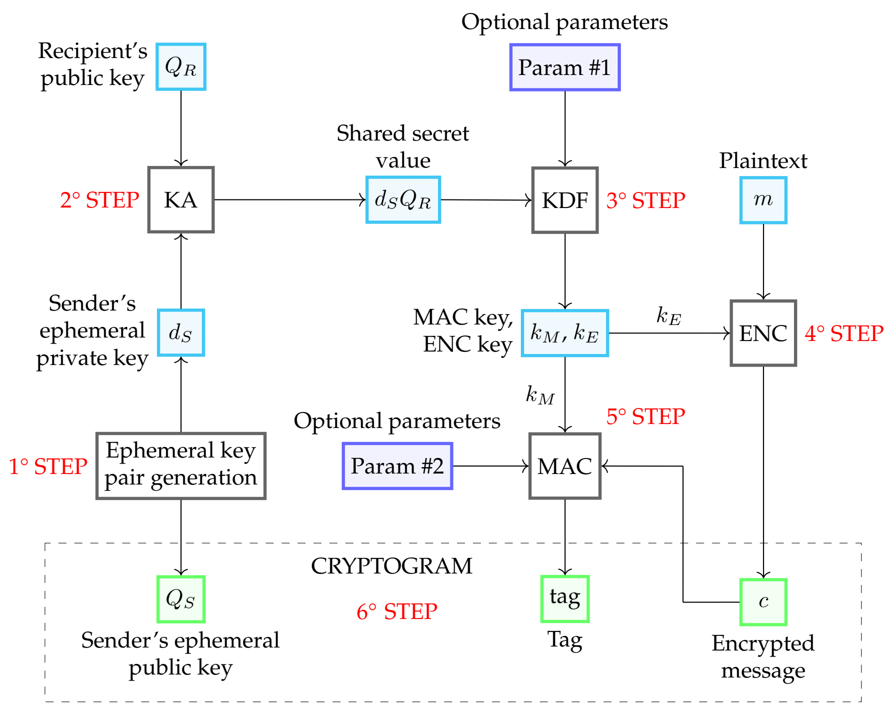

Figure 3.1: ECIES encryption scheme

A KA protocol is a protocol that allows two or more parties to agree on a key in such a way that both parties have an effect on the outcome. Protocols that are useful in practice do not divulge the key to any eavesdropping party.

### 3.1.2 Hash function

Hash functions take a variable length binary string as input and return a fixed length binary string corresponding to the starting data. Other primitives under the scope of ECIES use hash functions (e.g. KDF or MAC).

### 3.1.3 Key derivation function

When the sender and recipient share a secret $k_S$, the KDF can be used. A KDF is a cryptographic algorithm that uses a secret value and some optional parameters, $k_S$ and $O_1$ in this case, to generate one or more secret keys. Coupled with the shared secret key, ECIES requires one key $k_E$ for symmetric encryption/decryption and one key $k_M$ for integrity checking.

### 3.1.4 Message authentication code

The ECIES cryptosystem provides a MAC, also known as *tag*, a short piece of information that protects the message's data integrity as well as its authenticity. The tag is computed taking the MAC key $k_M$, the encrypted message $c$ and some optional parameters $O_2$ as input.

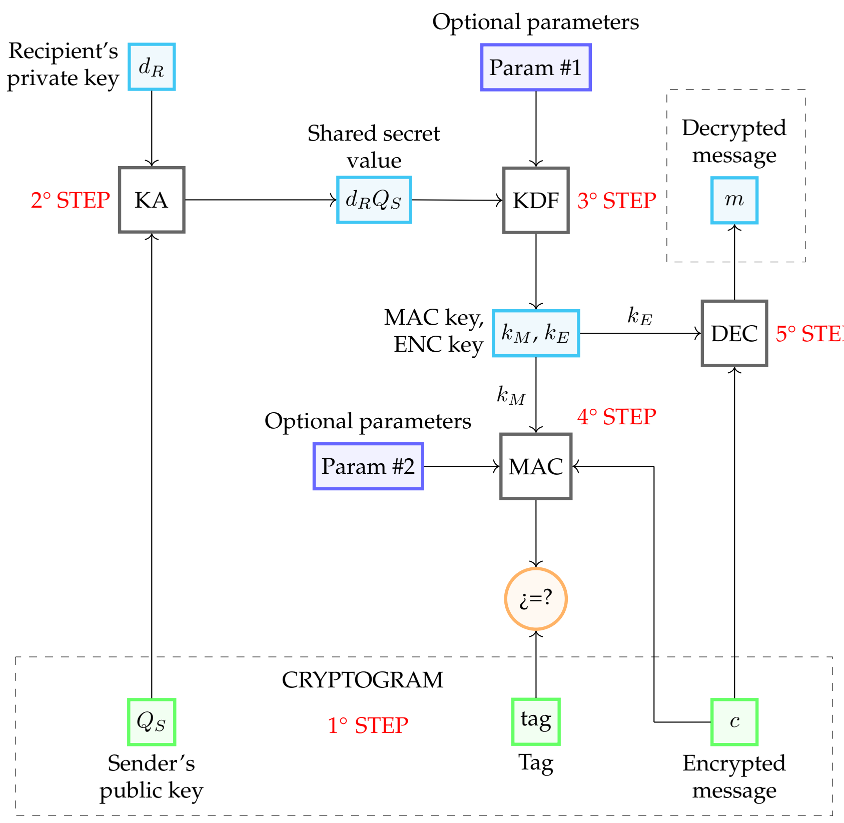

Figure 3.2: ECIES decryption scheme

### 3.1.5 Symmetric scheme

Finally, a symmetric cipher is used to encrypt and decrypt a message. In this chapter, we call them $ENC$ for encryption and $DEC$ for decryption. The encryption key is $k_E$, which was obtained from the KDF. Since it is symmetric, the same key is adopted for both encryption and decryption process.

## 3.2 ECIES encryption and decryption

Let us now integrate what has just been mentioned and attempt to outline the process of encryption and decryption of a communication using ECIES.

The whole encryption procedure shown in Fig. 3.1 is summarized by Algorithm 1. In order to describe the steps that must be taken to encrypt a clear message, we will assume that a *sender* wants to send a message to a *recipient*. The steps that the sender must complete in order to encrypt a plaintext are the following:

1. generate the sender's key pair $(d_S, Q_S)$ at random;
2. compute the share key applying the KA function with the sender's private key $d_S$ and the recipient's public key $Q_S$ as input;

3. use the KDF with the share key and some optional parameters as input in order to get the encryption key $k_E$ and the MAC key $k_M$;

4. encrypt the target message $m$ using a symmetric encryption method and the encryption key $k_E$;

5. use the selected MAC function, together with the encrypted message, the MAC key, and some optional parameters in order to produce a tag;

6. send a cipher text that combines the encrypted message $c$, its ephemeral public key $Q_S$ as well as the tag.

**Algorithm 1:** ECIES Encryption

```
Data: Message $m$, Recipient's public key $Q_R$, Optional shared
      parameters $O_1, O_2$
Result: Ciphertext $Q_S||c||d$

$(d_S, Q_S) \leftarrow$ generate_keypair();    /* $d_S$ is ephemeral sender's
  private key, $Q_S$ is ephemeral sender's public key */

$k_S \leftarrow$ KA($d_S$, $Q_R$) = {return $d_S Q_R$};      /* $k_S$ is shared key */
$(k_E, k_M) \leftarrow$ KDF($k_S||O_1$);     /* $k_E$ is encryption key, $k_M$ is
  MAC key */

$c \leftarrow$ ENC($k_E$, $m$);              /* $c$ is encrypted message */
$d \leftarrow$ MAC($k_M$, $c||O_2$);   /* $d$ is tag of encrypted message */
```

In order to access the original message and verify its integrity, the recipient has to perform the following actions:

1. retrieve the ephemeral public key $Q_S$, the tag, and the encrypted message $c$ from the cryptogram so that they can be managed individually;

2. compute the shared secret $k_S$ using the KA function with the private key $d_R$ and the ephemeral public key $Q_S$ as input. As previously stated, this is the identical one used by the sender;

3. using KDF, obtain $k_E$ and $k_M$ originating from the shared key $k_S$;

4. compute the element tag′ using the MAC key $k_M$, the encrypted message $c$, and the same optional parameters used by sender. After that, the recipient must compare the tag′ value with the tag received as part of the cryptogram. If the values are different, the receiver must reject the cryptogram due to a failure in the MAC verification procedure. Otherwise, it indicates that certain problems or assaults occurred during the operation;

5. decrypt the encrypted message $c$ using the same symmetric cipher as the sender used and the encryption key $k_E$.

Algorithm 2 summarizes the decryption process from keys derivation to message decryption.

**Algorithm 2:** ECIES Decryption

```
Data: Ciphertext $Q_S||c||d$, Recipient's private key $d_R$, Optional shared
      parameters $O_1, O_2$
Result: Message $m$

$k_S \leftarrow$ KA($d_R$, $Q_S$) = {return $d_R Q_S$} ;      /* $k_S$ is shared key */
($k_E, k_M$) $\leftarrow$ KDF($k_S||O_1$) ;    /* $k_E$ is encryption key, $k_M$ is
 MAC key */

if $d \neq MAC(k_M, c||O_2)$ then
    $failed$;
else
    $m$ = DEC($k_M$, $c$)
end
```

## 3.3 Selected functions in ABE-SSS

Given the number of functions and options available, the most difficult aspect of utilizing ECIES is determining the best combination of functions and parameters to employ.

### 3.3.1 ECDH-based KA

According to ECIES standards, we chose to use a KA function based on ECDH principle of operation (see Section 2.4.1).

The process adopted is as following. Initially, the sender and recipient agree to use a certain cryptographic elliptic curve $E$ over a finite field $\mathbb{F}_q$, along with its generator point $G$ and order $n$. Also, each side must have a suitable key pair for ECC. Assume the sender's key pair is $(d_S, Q_S)$ and the recipient's key pair is $(d_R, Q_R)$. According to ECIES standard, the sender's public key is ephemeral. Once each party has derived its own key pair, the recipient sends its public key to the sender. The sender computes the shared secret as $k_S^{'} = d_S Q_R$. At a later time, the sender sends its ephemeral public key to the recipient so that the latter can derive the same secret as well. The recipient computes $k_S^{''} = d_R Q_S$. The sender and receiver now share the same secret, i.e. $k_S \equiv k_S^{'} = k_S^{''}$, as already demonstrated in Eq. (2.20).

### 3.3.2 Chacha20-Poly1305

For the symmetric encryption scheme, we chose to use the Chacha20 [32] stream cipher with a Poly1305 MAC [33]. Chacha20-Poly1305 is widely available, well studied, has good security margin, is fast in software and very modern. This combination is being standardized by the IETF [34].

# Chapter 4

# Shamir's Secret Sharing

Shamir's Secret Sharing (SSS) scheme is a technique invented by famous Israeli cryptographer Adi Shamir in 1979 [9].

It is an important cryptographic technique that allows private information, namely *secret*, to be distributed safely across an untrusted network. Secret sharing works by splitting confidential information into smaller parts, or *shares*, and then distributing those pieces to a group or network. Individual shares are meaningless on their own, but when a specific fraction of those shares are combined, they recreate an original secret. This implies that, rather than requiring all shares to recover the original secret, Shamir's approach required only a minimum number of shares. This minimum is known as the *threshold*.

## 4.1 Mathematical formulation

Given a secret $S$, Shamir's Secret Sharing aims to split the secret into $n$ pieces $S_1, \cdots, S_n$ called shares, in such a way that:

- the secret $S$ is easily computed if you know any $t$ or more parts;
- the secret $S$ can't be reconstructed with fewer than $t$ pieces.

Shamir's Secret Sharing is based on the Lagrange interpolation theorem, which states that $t$ points are sufficient to uniquely calculate a polynomial of degree less than or equal to $t-1$.

Moreover, a finite field $\mathbb{F}_q$ of order $q$ is used with the intention of providing a higher security level. It must be $q > S, q > n$ where $q$ is publicly known.

### 4.1.1 Making shares

We generate the polynomial $f(x) = \sum_{i=0}^{t-1} a_i x^i \bmod q$ by choosing $a_0 = S \in \mathbb{F}_q$ and picking $t-1$ random components $a_1, \cdots, a_{t-1} \in \mathbb{F}_q$. Let us create any $n$ points $(x_0, y_0), \cdots, (x_{n-1}, y_{n-1})$ out of it. Every participant is given at least one point, whose ordinate needs to be a non-zero in order not to reveal the secret.

Since everyone who receives a share also has to know the value of $q$, it may be considered to be publicly known. Therefore, one should select a value for $q$ that is not too low.

Algorithm 3 shows the pseudocode for this first part of the SSS algorithm, known as "making shares".

**Algorithm 3:** Making shares

```
Data: Number of shares $n$, Threshold $t$, Secret $S$
Result: Random shares $\{S_i\}_{i=1,...,n}$

$a_0 \leftarrow S$;
for $i \leftarrow 1$ to $t$ do
    $a_i \leftarrow$ random();
end
$p(x) \leftarrow a_t x^t + ... + a_1 x + a_0 \bmod q$;

for $j \leftarrow 1$ to $n$ do
    $x_j \leftarrow$ random();
    $S_i \leftarrow (x_j, p(x_j))$;
end
```

### 4.1.2 Recovering secret

Interpolation can be used to obtain $a_0$ from any subset of $t$ of these pairs. Lagrange interpolation is formulated as follows. Given a set of $t$ points $(x_0, y_0)$, $\cdots$, $(x_{t-1}, y_{t-1})$, where no two $x_j$ are the same, the interpolation polynomial in the Lagrange form is a linear combination

$$L(x) := \sum_{j=0}^{t-1} y_j l_j(x) \bmod q$$

of Lagrange basis polynomials

$$l_j(x) := \prod_{\substack{0 \le m \le t \\ m \neq j}} \frac{x - x_m}{x_j - x_m}$$

where $0 \le j \le t$.

Hence, the secret $S = a_0$ can be obtained through interpolation, with one possible formula for doing so being $a_0 = L(0)$. It is worth noting that $L(0)$ is equal to the first coefficient of the polynomial $L(x)$.

Algorithm 4 shows the pseudocode for the second part of the SSS algorithm, known as "recovering secret".

## 4.2 Observations

### 4.2.1 Information theoretically secure

In order to reconstruct the secret, the threshold must be met. The secret cannot be reconstructed if there is anything less than the threshold. That makes

**Algorithm 4:** Recovering secret

```
Data: Threshold $t$, Shares $\{S_i\}_{i=1,\ldots,m}$
Result: Secret $S$

$\{(x_i, y_i)\}_i \leftarrow \{S_i\}_i$;

if $m \leq t$ then
    secret can't be recovered ;
else
    $S \leftarrow 0$;
    for $j \leftarrow 1$ to $t$ do
        $l_j \leftarrow 1$;
        for $k \leftarrow 1$ to $t$ and $k \neq j$ do
            $l_j \leftarrow \frac{x_k}{x_k - x_j}$;
        end
        $S \leftarrow S + y_j * l_j$;
    end
    $S \leftarrow S \bmod q$
end
```

Shamir's Secret Sharing secure against an adversary with unbounded processing power, i.e. a hostile attacker. In cryptography, this is referred to as information theoretically secure. The term "information theoretically secure" merely refers to the fact that not even an attacker with unlimited computer capacity would be able to decrypt the encrypted secret.

### 4.2.2 Share revocation

One of the advantages of Shamir's approach is its flexibility and extensibility. It means that the secret owner could add, edit, or remove shares at any time without affecting the original secret.

# Part II

# Attribute-Based Encryption using Shamir’s Secret Sharing

# Chapter 5

# ABE-SSS project

In the previous chapters, the fundamental principles were explained. Now, those techniques can be combined to create an Attribute Based Encryption scheme using Shamir's Secret Sharing, which will be explained in the following chapters.

The implementation of the system is written in the Rust programming language [35]. The first official release of the programming language was in 2015. Rust is a compiled language that has promising features, such as performance that is comparable to the C programming language. In addition, it uses new programming paradigms which guarantee safe memory handling. This is very appealing for software used for cryptography. Around 70 % of security vulnerabilities come from memory related bugs.

For the elliptic curve operations, the curve25519-dalek-ng library is utilized [36]. This library provides group operations on the Montgomery and Edwards forms of Curve25519, as well as on the prime-order Ristretto group. The chacha20poly1305 library, which is a pure Rust implementation of the algorithm ChaCha20Poly1305 [32], is also used.

# Chapter 6

# Resource securing

A database resource is any valuable object stored in a database and represented as a binary string of $0$ and $1$.

Due to recent cyber data and theft attacks, data assets are becoming a greater concern. Such data can in fact represent valuable information, which is often private, and should be kept safe from the clutches of malicious people who would use it for illegal purposes.

An encrypted database provides not only an improved data security but also a solution to that issue. The cryptographic procedure employs an algorithm to convert database resource into "cipher text" (illegible text). Security breaches are unavoidable. Nevertheless, thanks to improved data security and encryption techniques, hackers may be unable to analyze or decrypt the data to gain a better understanding of it. If your database is encrypted, an attacker will need a method to decrypt the data. The complexity of the cipher and the techniques used to generate the encrypted data determine how thoroughly the data can be secured.

As a result, cybersecurity professionals have long recognized cryptography as an important component of cybersecurity readiness. Researchers, moreover, have recently described a new cryptographic method to improve the security of underlying data. When a predetermined set of user attributes coincides with those of the ciphertext, data decryption occurs.

## 6.1 Resource encryption

The resource $R$ must be encrypted using an encryption function $E$ and a Data Encryption Key (DEK). The pseudocode is shown in Algorithm 5.

```
Algorithm 5: Resource encryption
  Data: Resource $R$, Data encryption key $dek$
  Result: Encrypted resource $c$
1 $c \leftarrow E(dek, R)$
```

The encrypted value $c = E\left(dek, R\right)$ is stored (see Fig. 6.1). By contrast, the DEK will be treated differently. That will be covered in greater detail in the following chapters.

***Store***

$$\boxed{E\left(dek, R\right)}$$

Figure 6.1: Stored encrypted resource
using the data encryption key $dek$.

The resource encryption method $E$ will not be discussed in detail. It is not our intention. It could be any symmetric cipher. The DEK will rather be the focal point.

## 6.2 Resource decryption

After recovering the DEK, the resource $R$ can be decrypted using the decryption function $D$. That is to say, if we call $c$ the ciphertext of the resource $R$ encrypted with the DEK, then $c = E(dek$,R$)$. As a result, the resource $R$ can be recovered by using $R = D(dek, c)$.

The pseudocode is shown in Algorithm 6.

**Algorithm 6:** Resource decryption

```
Data: Encrypted resource $c$, Data encryption key $dek$
Result: Decrypted resource $R$
1 $R \leftarrow D(dek, c)$
```

The methods for recovering the DEK will be discussed in the next section.

# Chapter 7

# Data encryption key securing

In the previous chapter, we discussed the importance of using an encrypted database to defend against new and increasingly frequent hacker attacks. Furthermore, we presented the option of making our resources even more secure by employing an attribute-based encryption method. In other words, despite having access to a database, we will only be able to decrypt the resources associated with our role within the database. This mechanism provides increased security against both internal and external attacks on the system.

We also learned how to encrypt and decrypt a database resource. Despite the fact that this appears to be a simple task, the main challenge is including information about the resource access control policy into the DEK.

In this chapter, we will first look at how to manage an access control policy associated with a resource. We will see how it can be represented by a Boolean expression, which can then be managed as a tree structure (Section 7.1). Following that, we will take a glance at the procedure for integrating the policy tree with the DEK. We will see that the DEK can be chosen at random and independently because its connection to the access control policy is not established until much later. This bond will be formed by constructing a second tree known as the share tree (Section 7.2). Finally, the issue of how to secure and limit access to these shares to specific classes of users emerges, which was the primary goal. In fact, whoever has access to these shares has access to the DEK, and thus to the resource itself, resolving the original problem (Section 7.3).

## 7.1 Policy tree

Given that not everyone has access to all of the information in the database, a method for managing resource access is required. For this reason, every resource has an access control policy associated with it. A resource policy establishes who is authorized to access the resource. It may determine no control, allowing anybody to access the resource, or it may establish certain limits.

Since the purpose of this work is to safeguard the secrecy of data depending on the role a user assumes for a given database, an access control policy based on attributes must be implemented.

A resource access control policy can be modelled as a Boolean expression, with the variables represented by system attributes. If the entity requesting resource access has a certain attribute, the variable corresponding to that attribute is true; otherwise, it is false. The resource access is permitted when the result of the expression is true. Access is denied otherwise.

**Example 7.1.1.** For instance, assume this access control and confidentiality protection management technique is applied to a university system. Assume also there are four possible roles in this system: administrator, professor, assistant and student. Define these properties as Boolean values:

$$\begin{aligned} q &= \text{being an administrator} \\ p &= \text{being a professor} \\ r &= \text{being an assistant} \\ s &= \text{being a student} \end{aligned}$$

Hypatia is a PhD student and teaches a few courses at our example university. Therefore, she has two roles: professor and student. In the case of Hypatia, the Boolean variables are $q = False$, $p = True$, $r = False$, $s = True$. Hypatia requests to access a resource $c$ with the following access control policy:

$$p \wedge q \vee p \wedge (r \vee s) \vee q \wedge (r \vee s) \tag{7.1}$$

In case of Hypatia, the result of the Boolean expression is $True$. That means she can access the resource $c$.

A Boolean expression, in turn, can be represented as a tree structure, with internal nodes (or branch nodes) containing the Boolean operators and external nodes (or leaf nodes) containing the expression's variables. As a result, each access control policy linked with a resource is represented using a tree structure.

**Example 7.1.2.** The tree structure representation of the Boolean statement Eq. (7.1) is depicted in Fig. 7.1.

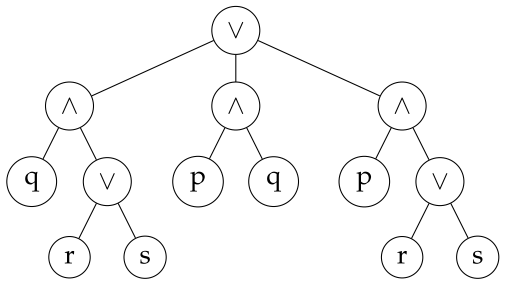

Figure 7.1: The tree structure representation of the Boolean expression Eq. (7.1) using Boolean operations in inner nodes.

Moreover, the tree representation may become more compact if the Boolean representation is dropped in favour of allowing internal nodes to take on integer values, known as threshold. Given an inner node $n_i$ and a threshold value $t$, the expression associated with the subtree of $n_i$ is satisfied if at least $t$ of her children's expressions are satisfied.

**Example 7.1.3.** To give an illustration of what that means, the Fig. 7.2 depicts a tree $T$. It is the compact representation of the Boolean expression Eq. (7.1). In other words, Fig. 7.1 and Fig. 7.2 are interchangeable. Firstly, examine the subtree $T'$. It is rooted in the internal node $n_{1,2}$ and has two child nodes, $n_{2,1}$ and $n_{2,2}$. The expression corresponding to $T'$ is true if and only if the user requesting access has at least one (1) of the $r$ and $s$ attributes. Threshold 1 is the same as the Boolean operator *or* ($\vee$). Secondly, take into account the complete tree $T$ rooted in $n_{0,0}$. Its expression is satisfied if and only if the user requesting access meets at least two of the following requirements: $p$ is true, $q$ is true, $T'$ corresponding expression is true.

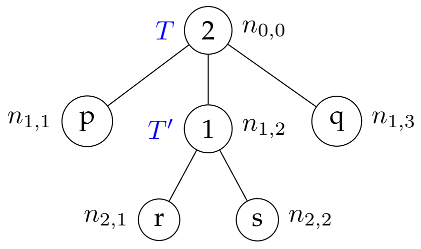

Figure 7.2: The simplified tree structure representation of the Boolean expression Eq. (7.1) using thresholds, instead of Boolean operators, in the inner nodes.

Let us name this representation of an access control policy the *policy tree* of a resource. Every policy tree node has a value, which varies depending on the type of node. Branch nodes have thresholds, whereas leaf nodes have attributes.

### 7.1.1 Attribute keypairs

In order to protect a resource from unauthorised access, we assume that each attribute $a_i$ is associated with an elliptic curve private-public key pair $(d_{ai}, Q_{ai})$ (see Section 2.4.1). By definition, public keys can be revealed to the entire system. Instead, private keys are only issued to those who have verified the attribute linked with the particular private key. Encryption is performed using public keys attributepublickey. As a result, anyone who wants to securely save content for a specific group of users who verify the attribute $a_i$ will need the public key associated with $a_i$. The private key $d_{ai}$, on the other hand, is valuable to descriptors. The revelation of the private key is not influenced by knowledge of the public key.

**Example 7.1.4.** Consider again the case of Hypatia in Example 7.1.1. The keypairs in the university system under examination are $(d_p, Q_p)$, $(d_q, Q_q)$, $(d_r, Q_r)$, $(d_s, Q_s)$. The public keys $Q_p$, $Q_q$, $Q_r$, $Q_s$ are accessible to all system users, and Hypatia has the secret keys $d_p$ and $d_s$ because she is classified as both a student and a teacher. Assume a second system user wants to preserve a data item for all the system's teachers and encrypts it with the public key $Q_p$. At this point, Hypatia and the other teachers can access the data by decrypting it with their private key $d_p$.

## 7.2 Share tree

At this stage, we must determine how to connect the DEK to the resource access policy. Furthermore, it is important to specify how to defend the entire system without any information being spread. Indeed, only those who match the required criteria can access the data. Everyone else must not even be able to acquire partial knowledge. This leads to identify two major issues. The first is to generate a key that can be recovered by as many different components as there are requirements. The second issue is securing those components with an attribute-based cryptographic primitive.

One way to investigate these issues was to examine the tree concept once more. Consider visiting the policy tree from the leaves to the root, only crossing nodes with possessed attributes. Instead, branch nodes with threshold $t$ can only be crossed if at least $t$ of their child nodes have been reached. Thus, whoever reaches the top, i.e. the tree root, has access to the resource. Then we could perform the following. We can create a second tree, first known as the secret tree. It has the same structure as the policy tree, and its root represents a major secret, which is the DEK used to encrypt the resource. Likewise, we can assure that only those who reach the top of the policy tree can reach the root of the secret tree. Now we must consider what the other nodes of the secret tree should include. The SSS algorithm allows you to split a secret into multiple shares and reconstruct the primary secret by collecting a certain minimum number of shares. This algorithm is exactly suited to our scenario. It begins with the construction of the root, whose value is the DEK, and we divide it into many Shamir's shares. To be more specific, we produce a share for each node in the secret tree. The decomposition occurs in such a way that reconstructing the major secret corresponds to ascending to the upper level of the tree. So, while the secret tree can be visited from the leaves to the root, the shares are created in the exact opposite direction: from the roots to the leaves.

According to SSS algorithm, what we previously referred to as secrets are actually Shamir's shares. Consequently, the secret tree became known as the share tree. The concept of secrecy, on the other hand, must be preserved because only those who are permitted can recombine the numerous shares and recover the primary secret. If the shares were saved in plain text, anyone might read them and recover the DEK. Remember that the DEK was utilized to safeguard the initial resource, which was the fundamental goal of the entire project. For this reason, the SSS algorithm is used in conjunction with the share encryption. In order to encrypt shares, attribute public keys are used, while primate keys are needed to decrypt them.

In the following sections, we will go through each step of this technique, from the formation of the share tree to its encryption and decryption. Finally, we will summarize the entire procedure. Initially, we will begin with a simple one-level tree to examine how the algorithm operates in simple scenarios (see Section 7.2.1). Then, we will try to get to the heart of the matter gradually by analysing a two-level tree (see Section 7.2.2). Finally, we will look at the broader situation of the recursive step with the multi-level tree Section 7.2.3.

### 7.2.1 One-level tree

Consider a resource $R$ with a policy tree $T$ that has only one level of child nodes. Let us refer to it as a one-level tree. The root $n_0$ has threshold $t$ and the child nodes $\{n_i\}_i$ have attributes $\{a_i\}_i$ respectively.

**Example 7.2.1.** Fig. 7.3 shows an example of this case when there are three children. The resource under consideration can be accessed only by those who meet at least $t$ of $a_1$, $a_2$ or $a_3$ attributes, where $t \leq 3$.

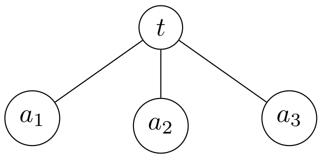

Figure 7.3: Example of a one-level policy tree.

#### Share tree generation

The first problem to be overcome is to generate the DEK that can be recovered by as many different components as there are requirements. A possible solution to that problem is as follows. According to our solution design, the DEK does not need to be a particular one. It can be generated randomly. Then, the SSS algorithm is applied to the DEK, so the latter becomes the secret to be split. We compute as many Shamir's shares $\{S_i\}_i$ as there are first generation child nodes (policy tree nodes at the first level). The outcome is another tree with the same structure as the policy tree. It is known as the share tree.

Algorithm 7 shows the share tree generation algorithm for one-level trees. First and foremost, along with the policy tree $T$, the DEK is an input. As previously stated, it can, however, be generated at random during previous stages. Secondly, the "sss.make_shares" function described in Algorithm 3 is invoked. Its inputs are the number of shares $n$ , the threshold $t$ and the secret $s$. This application defines $n$ as the number of children of the policy tree root, $t$ as the value set in the root, and $s$ as the DEK to be protected. Finally, the share tree is constructed using the secret $s$ as root's value and the Shamir's shares as children's values.

**Example 7.2.2.** Let us now apply the share tree generation algorithm to the policy tree shown in Fig. 7.3. Fig. 7.4 illustrates the resulting share tree. The value of the root is the DEK that must be protected. The root is then connected to the Shamir's shares $\{S_i\}_i$. As we can see, the structure of the share tree is identical to that of the policy tree from which it originated.

#### Share tree encryption

The second challenge is to secure the shares in order to store them safely. They are inaccessible unless a user meets the requirements.

In order to achieve this, we can encrypt each share $S_i$ with its corresponding attribute public key $Q_{a_i}$ using ECIES (see Chapter 3). The output is a concatenation of ephemeral public key $Q_{S_i}$, MAC $tag_i$ and encrypted share $c_i$. After

**Algorithm 7:** The share tree generation for one-level policy tree

```
Data: Policy tree $T$, Data encryption key $dek$
Result: Share tree $T'$

$n \leftarrow$ T.root.children.len();    /* $n$ is the number of shares */
$t \leftarrow$ T.root.threshold ;    /* $t$ is the threshold */
$s \leftarrow dek$;    /* $dek$ is the secret */
$\{S_i\}_i \leftarrow$ sss.make_shares($n, t, s$);

$T' \leftarrow new\_tree()$;
$T'.root.value \leftarrow s$;
for $i \leftarrow 1$ to $n$ do
    $child \leftarrow T'.root.children[i]$;
    $child.value \leftarrow S_i$;
end
```

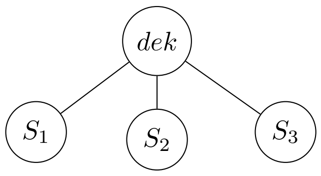

Figure 7.4: The one-level share tree corresponding to the policy tree of Fig. 7.3.

that, the resulting cryptogram $eS_i = Q_{Si}||tag_i||c_i$ is saved in place of the plain share. The DEK can be forgotten.

This process's pseudocode is shown in Algorithm 8. The share tree serves as the data input. Let's call it plain share tree in this case because the shares within it are still saved in clear text. The "ecies.encrypt()" function from Algorithm 1 is invoked within the for-loop. The output tree has the same structure as the input tree. The root content has been deleted, and the shares have been encrypted.

**Example 7.2.3.** Fig. 7.5 depicts the share tree when the technique is applied to the one-level tree $T$. As previously stated, we obtained as many shares as the feasible requirements for the resource in question. The requirements in this situation are the three attributes $a_1, a_2, a_3$. We can then run the Algorithm 8 to the shares $S_1, S_2, S_2$ using the public keys $Qa_1, Qa_2, Qa_3$ of the respective attributes. The resulting cryptograms $\{eS_i\}_i = \{Q_{Si}||tag_i||c_i\}_i$ with $i = 1, 2, 3$ are saved. As we can see, the DEK is missing because it can and must be forgotten. Only profiles that meet the system's requirements will be able to recover from it.

### 7.2.2 Two-level tree

In this section, we will stare at a slightly more complicated case. Assume that the policy tree $T$ of a resource $R$ has two levels. It is also referred to as a two-level tree. The root $n_0$ has threshold $t_0$. The $i$-level child nodes $\{n_{ij}\}_j$ that are

**Algorithm 8:** The share tree encryption for one-level share tree

```
Data: Plain share tree $T$, Attribute's public keys $\{Q_{ai}\}_i$
Result: Encrypted share tree $T'$

$n \leftarrow$ T.root.children.len();    /* $n$ is the number of shares */

$eS \leftarrow$ [ ];
for $i \leftarrow 1$ to $n$ do
    $S_i \leftarrow T.root.children[i].value$;
    $eS_i \leftarrow$ ecies.encrypt($S_i, Q_{ai}$);
    eS.append($eS_i$);
end

T′.root ← node.new();
for $i \leftarrow 1$ to $n$ do
    T′.root.children[i] ← node.new(value = $eS_i$);
end
```

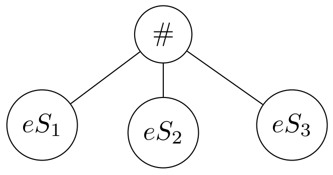

Figure 7.5: The one-level encrypted share tree corresponding to the plain share tree of Fig. 7.4.

leaves have attributes $\{a_{ij}\}_j$, those nodes that are branches have thresholds $\{t_{ij}\}_j$.

**Example 7.2.4.** Fig. 7.6 shows an example of this case when there are three children in the first level and two children in the second level. It is obvious that $t_0 \leq 3$ and $t_{12} \leq 2$.

### Share tree generation

One approach to generate a share tree for a two-level policy tree is as follow. Algorithm 7 can be applied to the root $n_0$ and its children first, and then to each first-level branch node and its children. This recursive approach allows for code reuse and generalization to a tree with an arbitrary number of levels.

**Example 7.2.5.** We therefore analyze the example depicted in Fig. 7.6. The tree looks like in Fig. 7.7 after running the Algorithm 7 once. The DEK and the first shares $\{S_{1j}\}_{j=1,2,3}$ are located at the very top of the tree. The bottom, however, still represents the policy tree. Then, Fig. 7.8 depicts the outcome of the second step of the method. In this case, the entire tree contains keys and shares. Like in Fig. 7.6, there is an overlap in the node $n_{12}$. Indeed, it corresponds to both share $S_{12}$ and the Key Encryption Key (KEK) $kek$. We will learn how to manage this link and why the key has a different name from the one on the root in the

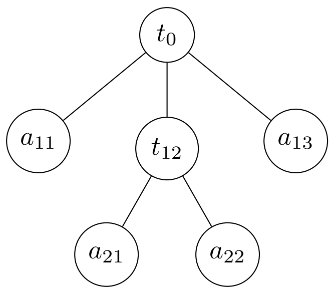

Figure 7.6: Example of a two-level policy tree.

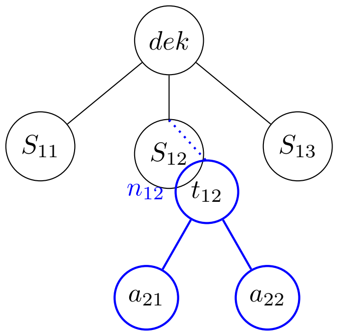

Figure 7.7: Example of a two-level tree half share tree (in black) and half still policy tree (in blue).

sections that follow. However, the share $S_{12}$ is expected to be the object to be protected with the KEK.

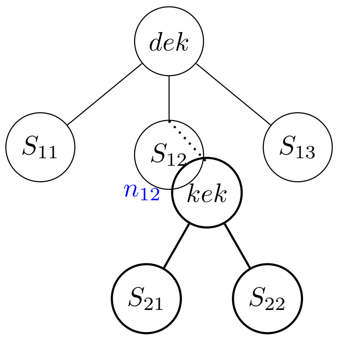

Figure 7.8: Example of a two-level share tree.

### Share tree encryption

It is a fair guess that the trivial case Algorithm 7 shall be applied twice to encrypt a two-level share tree. We just need to clarify the connection between the branch share $S_{12}$ and the KEK. There exist many methods for dealing with this problem, we chose the following. A branch node has no linked attributes and thus no associated key pair by definition. The KEK can be considered as the private key $d$ of a key pair linked with branch node $n_{12}$. According to Section 2.4.1, the matching public key $Q$ is calculated from the private key $d \equiv k_E^{'}$. As a result, we can use the same hybrid cryptosystem, i.e. ECIES, on all nodes, with different receiver's keypairs depending on whether they are leaf or branch nodes. The receiver's keypair is the attribute keypair $(d_{a_i}, Q_{a_i})$ when the node is a leaf; when the node is a branch, the receiver's keypair $(d_R, Q_R)$ is specially constructed. In branch case, the receiver's private key $d_R$ ($\equiv k_E^{'}$) is generated automatically. The private key $d_R$ is used to generate the public key $Q_R$. The private key $d_R$ represents the following iteration's input, namely the Shamir's secret to be split.

The Algorithm 9 specifies the encryption phase of a branch node. First, a private and public key pair is created. The private key is then used to determine the shares to be assigned to the various child nodes, while the public key is used to encrypt the node's content, which is its share.

**Algorithm 9:** The branch node encryption

```
Data: Branch node $node_j$, Plain share $S_j$
Result: Encrypted share $eS_j$, Plain derived shares $\{S_i\}_i$

$d_R \leftarrow$ ecies.private_key.random();
$Q_R \leftarrow$ ecies.public_key.from_private($d_R$);

$n \leftarrow$ node$_j$.children.len();     /* $n$ is the number of shares */
$t \leftarrow$ node$_j$.threshold;                   /* $t$ is the threshold */
$s \leftarrow d_R$;     /* $s$ is the ECIES receiver's private key */
$\{S_i\}_i \leftarrow$ sss.make_shares($n, t, s$);

$eS_j \leftarrow$ ecies.encrypt($Q_R$, $S_j$);              /* encrypted share */
```

**Example 7.2.6.** The entire algorithm for share tree encryption is applied to the share tree in Fig. 7.8 and the result is displayed in Fig. 7.9. The leaf node values are $eS_{ij}$ = ecies.encrypt($S_{ij}$, $Q_{a_{ij}}$) and attribute public key $Q_{a_{ij}}$ of attribute $a_{ij}$ with $ij = 11, 13, 21, 22$. The node $n_{12}$, on the other hand, is a branch, hence Algorithm 9 is utilized. The branch node value is $eS_{12}$ which was encrypted by taking the plain share $S_{12}$, the recipient's public key $Q_{R12}$ using the ECIES encryption function (see Algorithm 1). The recipient's public key $Q_{R12}$ was generated from its corresponding private key $d_{R12}$. The latter is also the Shamir's secret that gives rise to the shares $S_{21}$ and $S_{22}$.

## 7.2.3 Multi-level tree

We have finally reached the general case. The shares are generated and then encrypted in a recursive approach.

#### Share tree generation and encryption

Before we introduce the general recursive method, we need to make a few minor changes to what has been mentioned thus far. We must, in particular, har-

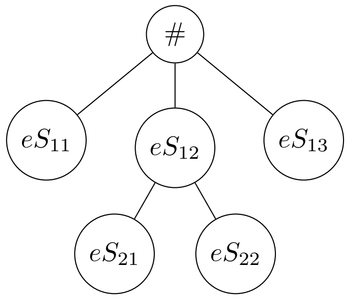

Figure 7.9: The two-level encrypted share tree corresponding to the plain share tree of Fig. 7.8.

monize the inputs. In fact, when discussing two-level share tree encryption in Section 7.2.2, we managed to encrypt every node value using the same mechanism regardless of whether the node was a leaf or a branch. As long as all node values represent the same type of data, this is possible. This is not true for every node in the tree, however: the root is still treated separately. The root value is overlooked, whereas any other node value is an encrypted share. One way to overcome these problems is to consider a first share $S_0$. It should be the recursive algorithm's input. $S_0$'s first coordinate is always zero, whereas its second coordinate is the DEK. $S_0$ can then be encrypted as any other node value.

**Example 7.2.7.** Fig. 7.10 shows an example of a generalized tree. It arises from the same policy tree as in the two-level instance, namely the policy tree depicted in Fig. 7.6. Let us compare the share trees in the multi-level case (see Fig. 7.10) and in the two-level case (see Fig. 7.9). We see that in the two-level case, the root content is termed absent. In the multi-level situation, we instead consider a share for each node, including the root. We know that $eS_0 =$ ecies.encrypt($Q_R$, $S_0$) for some $Q_R$, where $S_0 = (0, dek)$.

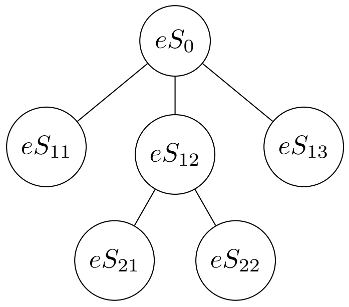

Figure 7.10: Example of multi-level encrypted share tree.

We are now ready to move on to the general algorithm. Taking a policy tree $T$, a plain first share $S_0$, and a succession of attribute public keys $\{Q_{ai}\}_i$ as inputs, the Algorithm 10 constructs an encrypted share tree $T'$. Let us examine the provided code beginning at the end, this will help us to better understand the recursive step. We can deduce from the last five lines of code (lines from 24 to 28) that our goal is to create the root of the encrypted sharing tree. When we create the root, we associate it with all of its children and descendants, in addition to the value of the encrypted share. The latter are generated in the preceding code section. Hence, let us go back to the beginning of the code and figure out how to generate the root descendants. Firstly, a distinction is made between branch and leaf nodes (if-else statement lines 2 to 22). There are no children if a node is a leaf (see line 21). Otherwise, if a node represents a branch, two major operations are carried out: the first one is the production of shares (lines 6 to 9), and the second operation is the recursive call (line 14). The newly made shares are used as input in the recursive call on the children of the $T$ root. Not only do we obtain the descendant nodes from the if-else statement, but we also obtain another crucial component of the algorithm: the ECIES recipient's public key. The latter is used to encrypt the root's value, which is the first share $S_0$. If the node is leaf and has attribute $a_i$, the recipient's public key is $Q_R$ (lines 18 and 19). Otherwise, if the node is branch, the recipient's public key is computed from a randomly generated private key (lines 3 and 4).

Finally, the ECIES encrypt function (line 24) accepts the share $S_0$ as input. This is a simplification due to pseudocode that aids comprehension. The real ECIES encryption and decryption functions accept a single binary string as input. A share, on the other hand, is a point formed by two coordinates. This indicates that, before moving from share to function, a concatenation of the point's first and second coordinates is required. In the decryption step, the inverse process is performed. Also during decryption process, concatenation is not an issue because, knowing the fixed dimension of both coordinates, it will be able to retrieve the share from their concatenation.

## 7.3 Recovering algorithm

Since the fundamental principles have previously been well described, we will concentrate on the general case in this section. The decryption and recovery algorithm for the initial share that included the DEK is then presented.

### 7.3.1 Share tree decryption and recovering

The purpose of this function is to recover the first share $S_0$, which is the plaintext share corresponding to the root of the share tree at hand. This is not always doable. As a consequence, we will see that the outcome of this stage of the procedure can be either the plain share or nothing. The Algorithm 11 shows the pseudocode of this last phase. During the share tree generation and encryption, we used the Shamir's "make shares" and ECIES's "encrypt" functions. The opposite functions are now employed. Shamir's "recover secret" (see Algorithm 4) and ECIES's "decrypt" (see Algorithm 2) are used. The algorithm can be divided into two macro-steps. During the first phase, we attempt to recover the ECIES decryption key, which is the recipient's private key $d_R$. This key is used to decrypt the encrypted share during the second phase. If the current tree's root is branch (see lines 4 to 17), then the ECIES recipient's private key is the Shamir's secret derived from those shares gained via a recursive call

**Algorithm 10:** The share tree generation and encryption

```
Data: Policy tree $T$, Plain first share $S_0$, Attribute public keys $\{Q_{ai}\}_i$
Result: Encrypted share tree $T'$

if $T.root$ is $Branch$ then
    $d_R \leftarrow$ ecies.private_key.random();
    $Q_R \leftarrow$ ecies.public_key.from_private($d_R$);

    $n \leftarrow$ T.root.children.len();      /* $n$ is the number of shares */
    $t \leftarrow$ T.root.threshold;            /* $t$ is the threshold */
    $s \leftarrow d_R$;  /* $s$ is the ECIES receiver's private key */
    $\{S_i\}_i \leftarrow$ sss.make_shares($n, t, s$);

    share_children $\leftarrow$ [ ];
    for $i \leftarrow 1$ to $n$ do
        policy_child$_i$ $\leftarrow$ T.root.children[i];
        share_child$_i$ $\leftarrow$ recursive_call($policy\_child_i$, $S_i$, $\{Q_{ai}\}_i$);
        share_children.append($share\_child_i$);
    end
else
    $a_i \leftarrow$ T.root.value ;             /* current root is leaf */
    $Q_R \leftarrow Q_{a_i}$ of attribute $a_i$;

    share_children $\leftarrow$ [ ];
end

$eS_0 \leftarrow$ ecies.encrypt($Q_R$, $S_0$) ;    /* encrypted first share */

T$'$.root $\leftarrow$ node.new() ;              /* new tree from root */
T$'$.root.value $\leftarrow eS_0$;
T$'$.root.children $\leftarrow share\_children$;
```

(line 10). In contrast, if the root is a leaf with the attribute $a_i$, the recipient's private key coincides with the attribute private key $d_{ai}$. If everything goes well, we should be able to acquire the first share in plaintext. However, we may be unable to get it because an error may occur in both the branch and leaf scenarios. When we do not the attribute private key or enough shares to retrieve the secret, the algorithm output is "None". It is impossible to recover the desired share.

**Algorithm 11:** The share tree decryption and recovery

```
Data: Share tree $T'$, Policy tree $T$, Attribute private keys $\{d_{ai}\}_i$
Result: Either Plain first share $S_0$ or None

$d_R \leftarrow$ ecies.private_key.new();     /* ECIES receiver's private key */

if $T.root$ is $Branch$ then
    $shares \leftarrow$ [ ];
    $n \leftarrow$ T.root.children.len();  /* $n$ is the number of children */
    for $i \leftarrow 1$ to $n$ do
        share_child$_i \leftarrow$ T'.root.children[i];
        policy_child$_i \leftarrow$ T.root.children[i];
        $S_i \leftarrow$ recursive_call($share\_child_i$, $policy\_child_i$, $\{d_{ai}\}_i$);
        shares.append($S_i$);
    end
    $t \leftarrow$ T.root.threshold;                 /* $t$ is the threshold */
    $secret \leftarrow$ sss.recover_secret($t$, $shares$);
    if $secret$ is $valid$ then
        $d_R \leftarrow secret$;
    end
else
    $a_i \leftarrow$ T.root.value ;               /* current root is leaf */
    if $d_{ai}$ exists then
        $d_R \leftarrow d_{ai}$;
    end
end

if $d_R$ is $valid$ then
    $eS_0 \leftarrow$ T$'$.root.value;
    $S_0 \leftarrow$ ecies.decrypt($d_R$, $eS_0$) ;        /* Plain first share */
else
    None("Decryption is not possible")
end
```

**Example 7.3.1.** Finally, let us conclude the Example 7.1.1 of Hypatia, whose policy tree was the one depicted in Fig. 7.2, trying to decode her share tree shown in Fig. 7.10. Remember that Hypatia has the attribute, and so the corresponding private key, for the roles $p$ and $s$. According to the Algorithm 11, we begin the decryption and recovering process from the root tree. However, we do not process it immediately since we first recurse on its children. Starting from its first child node, the share $eS_{11}$ in position $n_{11}$ can be decrypted using the attribute private key $d_p$ (see Fig. 7.11). In the following step, we go

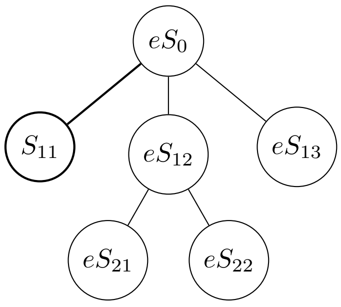

Figure 7.11: Example of decryption, first step.

to node $n_{12}$ but immediately recurse to its children, that are child nodes. Only the node in position $n_{22}$ can be decoded because it is linked to the $s$ attribute. The attribute private key $d_s$ is then used to obtain $S_{22}$. The share in position $n_{21}$, on the other hand, remains concealed since it lacks the corresponding $r$ property (see Fig. 7.12). We can now handle the node $n_{12}$. The recovering secret algorithm of SSS comes into play in this scenario (Algorithm 11, line 14). The corresponding threshold in the policy tree was $t_{12} = 1$, and the number of decrypted shares accessible is exactly 1. Therefore, we can apply the procedure to recover the secret of $n_{12}$ and use it to decrypt $eS_{12}$ (see Fig. 7.13). The share in position $n_{13}$ is unreadable since it is linked to the $q$ attribute, which Hypatia does not own (see Fig. 7.14). Lastly, we went back to the root of the tree. This had threshold $t_0 = 2$ and we are in possession of exactly two decrypted shares, $S_{11}$ and $S_{12}$. This means that we can recover the secret linked to $n_0$ and use it to decrypt $eS_0$ (see Fig. 7.15).

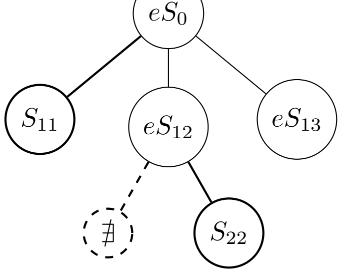

Figure 7.12: Example of decryption, second step.

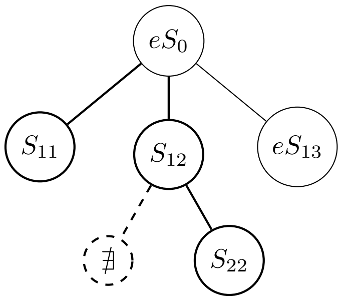

Figure 7.13: Example of decryption, third step.

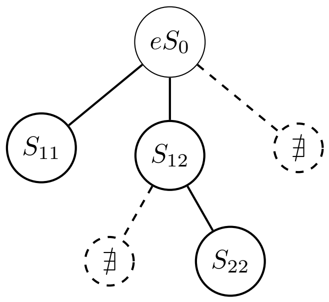

Figure 7.14: Example of decryption, fourth step.

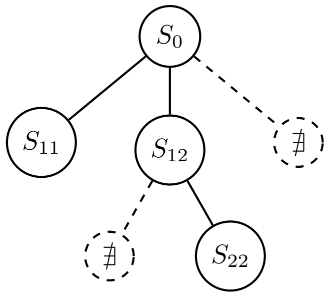

Figure 7.15: Example of decryption, fifth step.

# Chapter 8

# Results

This section outlines three key aspects of the work's outcome. Section 8.1 contains the proof of concept, which demonstrates the proper functioning of the algorithm under two alternative initial conditions. The memory and necessary macro-operations performance is reported in Section 8.2. Finally, in Section 8.3, an evaluation is conducted.

## 8.1 Proof of concept

The proof of concept is an important result for a new algorithm. Two of the basic tests used to validate Attribute-based Encryption Shamir's Secret Sharing (ABE-SSS) are shown in this section. They both function by carrying out the following procedures. A DEK is used to encrypt a generic database resource. The DEK is then added to the first share in the share tree, and the encryption process begins. The opposite process, that is, the reconstruction of the DEK and the decryption of the resource, is next performed.

Despite this, the two methods are distinct because they reflect two distinct starting points. The first test in Algorithm 12 describes a circumstance in which the user possesses the attributes required to reconstruct and decrypt the resource. Indeed, as we can see, the expected outcome is success.

Instead, Algorithm 13 offers the reverse circumstance, in that it is expected to fail in resource decryption since the user lacks the necessary prerequisites.

Both tests were unsuccessful. In other words, we obtained the resource from Algorithm 12 but not from Algorithm 13.

## 8.2 Solution performance

In this section, we present the performance of ABE-SSS project. The results are discussed in terms of the amount of bits necessary to save the solution and the number of macro-operations that must be executed.

The ABE-SSS technique returns a tree structure holding cryptograms, we called each one of them as $eS$. Remember that any $eS$ is made up of the sender's public key $Q_S$, tag, and encrypted share $c = E(k_E, S)$, where $S$ is a Shamir's share. We have a cryptogram for each node of the tree. Let us examine each

**Algorithm 12:** Test 1 - Should succeed

```
Data: Resource $R$, Policy tree $T$, Attribute public keys $\{Q_{ai}\}_i$ for every
      $i$, Attribute private keys $\{d_{aj}\}_j$ for some $j$ such that $T$
      requirements are met
Result: Successful test
/* Resource encryption process */
$dek \leftarrow$ random();
$c \leftarrow E(dek, R)$;

/* DEK encryption process */
$S_0 \leftarrow (0, dek)$;
$T' \leftarrow$ generate_encrypt($T$, $S_0$, $\{Q_{ai}\}_i$);

/* DEK decryption process */
$S_0^{'} \leftarrow$ decrypt_recover($T'$, $T$, $\{d_{aj}\}_j$);
$dek' \leftarrow$ get_second_coordinate($S_0^{'}$);

/* Resource decryption process */
$R' \leftarrow D(dek', c)$;

/* Verification */
assert_eq!($R$, $R'$)
```

**Algorithm 13:** Test 2 - Should fail

```
Data: Resource $R$, Policy tree $T$, Attribute public keys $\{Q_{ai}\}_i$ for every
      $i$, Attribute private keys $\{d_{aj}\}_j$ for some $j$ such that $T$
      requirements are **not** met
Result: Failing test
/* Resource encryption process */
$dek \leftarrow$ random();
$c \leftarrow E(dek, R)$;

/* DEK encryption process */
$S_0 \leftarrow (0, dek)$;
$T' \leftarrow$ generate_encrypt($T$, $S_0$, $\{Q_{ai}\}_i$);

/* DEK decryption process */
$S_0^{'} \leftarrow$ decrypt_recover($T'$, $T$, $\{d_{aj}\}_j$);        // output is None
$dek' \leftarrow$ get_second_coordinate($S_0^{'}$);

/* Resource decryption process */
$R' \leftarrow D(dek', c)$;

/* Verification */
assert_eq!($R$, $R'$);                                  // should fail
```

component and try to determine how many bytes are required to encode a cryptogram. A public key is nothing more than a point of the elliptic curve Curve25519, which is represented by 32 bytes. The tag from poly1303 requires 16 bytes. A Shamir's share is a pair of scalars, that is, elements of the cyclic subgroup produced by $G$, where $G$ is the point of the elliptic curve curve25519 with $x = 9$. A Shamir's share is represented by 64 bytes because each scale is encoded with 32 bytes. We need a total of $32 + 16 + 64 = 112$ bytes for each node of the tree (see Table 8.1).

| COMPONENT | BYTES |
|---:|:---:|
| sender's public key | 32 |
| tag | 16 |
| share | 64 |
| **cryptogram** | **112** |

Table 8.1: Number of bytes required for each cryptogram.

In this circumstance, we must remember that the table's tolopogy is significant. As a result, additional bytes will be required. However, it is difficult to assess this element right now because it is highly related to the type of database with which the solution is integrated.

Examine now the number of macro-operations required to obtain both a share tree and the DEK. Firstly, consider the macro-operations required to obtain the share tree. We need to process one ECIES encryption per node. If a node is a leaf, it also needs one "make share" function. Secondly, in order to obtain the DEK from a share tree, we have to perform one ECIES decryption per node, in addition to one "recover secret" function if the node is a leaf. The Table 8.2 summarizes this result.

| | # MACRO-OPERATIONS |
|:---:|:---:|
| make share / recover secret | $\#nodes$ |
| ecies encryption / ecies decryption | $\#branches$ |

Table 8.2: Number of macro-operations required for each ABE-SSS **encryption/decryption process**.

## 8.3 Solution evaluation

ABE-SSS is a new project at its very first stage. The proof of concept shown in Section 8.1 represent the first important achievement. The performance can undoubtedly be enhanced and optimized in terms of bytes as well as number of operation. Looking at the amount of bytes required to store, the overhead is quite big because it accounts for roughly 45% of the total number of bytes sought. It can almost certainly be reduced.

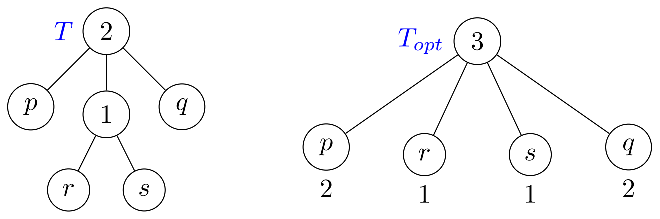

Figure 8.1: Original policy tree (left).
Optimized policy tree (right).

Also, branch nodes are the nodes that need the most computing effort while performing necessary macro-operations. The cryptographic operations themselves are already good and well optimized. Despite this, the number of macro-operations required can be decreased.

As we can see, the true heart of optimization at this stage is concerned with the structure of the tree connected with the resource policy. The greater the depth of the tree, the more operations and bytes are required to save the result.

A viable solution would be to award a variable amount of shares to each node dependent on its level, and hence on the threshold connected with it. The intuitive idea is shown in Fig. 8.1. The policy tree $T$ on the left is the original, which we discussed in Section 7.1. For each node in this tree $T$, one Shamir's share is counted. The policy tree $T_{opt}$ on the right is an optimized tree with varying numbers of Shamir's shares per node. Under each node of the tree, the number of shares corresponding to that node is given. The reader is exhorted to think about how the two trees convey the same policy. This improvement may be part of the future developments.

Furthermore, a second limit of the solution concern the collision resistance. Indeed, according to this model, people with varying attributes might combine their private keys and gain access to resources that do not belong to them. In this regard, this approach does not meet the ABE standard. However, this can be enhanced by preventing people from accessing confidential information if they do not match the required criteria right away. It is also part of future work so that ABE-SSS can be call an actual implementation of ABE standard. It is also part of future work to make ABE-SSS a proper implementation of the ABE standard.

# Conclusion

Data protection is becoming increasingly important, especially in recent years in which several and important cyber attacks have occurred. Data saved on digital devices is frequently private and/or sensitive. Therefore as a consequence, its protection is critical and can really determine our safety, security, and health. Security must come not just from the outside, but also from within systems, whether Information Technology (IT) or not. Structured systems significantly minimize the effects of both undesired errors and malicious behavior. Cryptography, especially that based on roles, has been highlighted as one of the most relevant data protection strategies. Although you can access a system, the resources stored within it is not viewable unless you have the appropriate role. Different roles provide you access to different resources.

LumoSQL, an open source project, also deals with data protection and security. Its aim is to create a library for embedded data storage that permits granular encryption of resources. The encryption process is based on the requirements held, also called attributes, rather than the position occupied by the resource (a certain table, rather than a certain column or row).

The ABE-SSS project developed at this point. The name origins from the core technique of this work, Attribute-Based Encryption (ABE), in conjunction with Shamir's Secret Sharing (SSS) scheme. While ABE is responsible for bring attributes into encryption and decryption process, SSS deals with splitting and recovering the components that keep a system secret, namely the encryption keys. The encryption and decryption themselves are provided by the hybrid scheme Integrated Encryption Scheme (IES) for elliptic curves, namely ECIES.

The results suggest that the algorithm works. When the prerequisites are satisfied, you can encrypt and decode DEKs; otherwise, you cannot. Despite this, this approach does not offer collusion resistance since various users with different secret keys can collect their secrets and decode resources that they are not permitted to access. This improvement will be part of future work. Also, because of the encryption procedure, the bytes necessary to save the solution have a significant overhead. The number of operations required grows in proportion to the depth of the tree. Optimizing the policy representation with a less deep tree would improve both bytes and operations required. A solution has already been provided.

# Future work

This work's next advances include a more effective and collision-resistant utilization of the ABE. In fact, as described in Section 8.3, ABE-SSS is not now collision resistant, but it can be.

Furthermore, optimization in terms of bytes and number of processes is regarded an impending future advancement. A viable approach to achieving this result is already illustrated in Section 8.3.

Finally, one of the upcoming initiatives is the integration of ABE-SSS with the LumoSQL project.

# Glossary

**ABE** Attribute-Based Encryption. Public-key cryptographic encryption technique wherein users use attributes as keys. Attributed initially to Sahai and Waters, and Bethencourt, Sahai, and Waters thereafter. v, 4, 49–51, 53

**API** Application programming interface. A standardized software interface that allows to applications to communicate with each other. 3, 53

**LumoSQL** An extension of SQLite that aims to add privacy and security to embedded databases. `https://lumosql.org/` v, 3, 4, 51, *see also* SQLite

**SQLite** The most used embedded SQL database engine. `https://sqlite.org/` v, 4

**SSS** Shamir's Secret Sharing. Cryptographic technique to split a secret into shares [9], based on polynomial interpolation. v, 4, 5, 24, 25, 34, 35, 44, 50, 53

# Abbreviations

**ABE-SSS** Attribute-based Encryption Shamir’s Secret Sharing v, 46, 48–51

**ABE** Attribute-Based Encryption v, 4, 49–51, *Glossary:* ABE

**API** application programming interface 3, *Glossary:* API

**DEK** Data Encryption Key 29–31, 34–37, 40, 41, 46, 48, 50

**ECC** Elliptic-curve Cryptography 6, 16, 19, 23

**ECDH** Elliptic-curve Diffie–Hellman 16, 17, 23

**ECDLP** Elliptic Curve Discrete Logarithm Problem 15, 17, 18

**ECIES** Elliptic-curve Integrated Encryption Scheme 19–21, 23, 35, 39, 41, 48, 50

**IES** Integrated Encryption Scheme 50

**IT** Information Technology 50

**KA** Key Agreement 19–23

**KDF** Key Derivation Function 19–22

**KEK** Key Encryption Key 37, 38

**KEM** Key Encapsulation Mechanism 19

**MAC** Message Authentication Code 19, 20, 22, 23, 35

**RSA** Rivest–Shamir–Adleman 6, 16, 19

**SEE** SQLite Encryption Extension 4

**SSS** Shamir’s Secret Sharing v, 4, 5, 24, 25, 34, 35, 44, 50, *Glossary:* SSS

**TDE** transparent database encryption 5

# Bibliography

[1] *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*, Apr. 27, 2016.

[2] *Consumer data right (CDR)*, Nov. 26, 2017. [Online]. Available: https://www.accc.gov.au/focus-areas/consumer-data-right-cdr-0 (visited on 08/29/2022).

[3] L. H. Newman, *Atlanta spent $2.6m to recover from a $52,000 ransomware scare*, Apr. 2018. [Online]. Available: https://www.wired.com/story/atlanta-spent-26m-recover-from-ransomware-scare/.

[4] C. Smith. "TikTok exploited an Android privacy loophole to track users," BGR. (Aug. 12, 2020), [Online]. Available: https://bgr.com/tech/tiktok-tracking-users-android-app-collected-mac-addresses/ (visited on 04/05/2022).

[5] Lapowsky Issie, "Facebook Exposed 87 Million Users to Cambridge Analytica," *Wired*, Aug. 3, 2020, ISSN: 1059-1028. [Online]. Available: https://www.wired.com/story/facebook-exposed-87-million-users-to-cambridge-analytica/ (visited on 08/03/2020).

[6] *Lumosql*. [Online]. Available: https://github.com/LumoSQL/lumosql.

[7] *Sqlite*. [Online]. Available: https://www.sqlite.org/index.html.

[8] *The sqlite encryption extension (see)*. [Online]. Available: https://sqlite.org/com/see.html.

[9] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/359168.359176. [Online]. Available: https://dl.acm.org/doi/10.1145/359168.359176 (visited on 03/31/2022).

[10] Archiveddocs, *Transparent data encryption (tde)*. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/sql/sql-server-2012/bb934049(v=sql.110)?redirectedfrom=MSDN.

[11] [Online]. Available: https://www.ibm.com/docs/en/db2/10.5?topic=windows-fix-pack-summary.

[12] *Oracle advanced data security*. [Online]. Available: https://www.oracle.com/it/security/database-security/advanced-security/.

[13] *Mysql enterprise transparent data encryption (tde)*. [Online]. Available: `https://www.mysql.com/products/enterprise/tde.html#:~:text=MySQL%5C%20Enterprise%5C%20TDE%5C%20enables%5C%20data,decrypted%5C%20when%5C%20read%5C%20from%5C%20storage.`.

[14] *Column-level encryption*. [Online]. Available: `https://www.ibm.com/docs/en/informix-servers/14.10?topic=data-column-level-encryption`.

[15] C. Braund, *Mongodb releases queryable encryption preview: Mongodb blog*, Jun. 2022. [Online]. Available: `https://www.mongodb.com/blog/post/mongodb-releases-queryable-encryption-preview`.

[16] J. H. Silverman, *The arithmetic of elliptic curves*. Springer, 2009, vol. 106.

[17] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976, ISSN: 0018-9448. DOI: `10.1109/tit.1976.1055638`.

[18] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[19] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987. DOI: `10.1090/S0025-5718-1987-0866109-5`.

[20] V. S. Miller, "Use of Elliptic Curves in Cryptography," in *Advances in Cryptology — CRYPTO '85 Proceedings*, H. C. Williams, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 417–426, ISBN: 978-3-540-39799-1.

[21] L. C. Washington, *Elliptic curves: number theory and cryptography*. Chapman and Hall/CRC, 2008.

[22] N. Koblitz, *A course in number theory and cryptography*. Springer Science & Business Media, 1994, vol. 114.

[23] W. M. Baldoni, C. Ciliberto, and G. P. Cattaneo, *Aritmetica, crittografia e codici*. Springer Science & Business Media, 2007.

[24] D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *International Workshop on Public Key Cryptography*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 207–228, ISBN: 978-3-540-33852-9.

[25] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted Edwards Curves," in *International Conference on Cryptology in Africa*, Springer, 2008, pp. 389–405.

[26] H. de Valence and I. Lovecruft, *Curve25519-dalek: A pure-Rust implementation of group operations on Ristretto and Curve25519*, 2016–2018. [Online]. Available: `https://github.com/dalek-cryptography/curve25519-dalek` (visited on 06/17/2018).

[27] Y. Nir and S. Josefsson, "Curve25519 and Curve448 for the Internet Key Exchange Protocol Version 2 (IKEv2) Key Agreement," Internet Engineering Task Force, Request for Comments RFC 8031, Dec. 2016, 8 pp. DOI: `10.17487/RFC8031`. [Online]. Available: `https://datatracker.ietf.org/doc/rfc8031` (visited on 10/12/2022).

[28] M. Marlinspike. "Advanced cryptographic ratcheting," Signal Messenger. (Nov. 26, 2013), [Online]. Available: `https://signal.org/blog/advanced-ratcheting/` (visited on 04/27/2021).

[29] M. Hamburg, "Decaf: Eliminating Cofactors Through Point Compression," in *Advances in Cryptology – CRYPTO 2015*, ser. Lecture Notes in Computer Science, R. Gennaro and M. Robshaw, Eds., vol. 9215, Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 705–723, ISBN: 978-3-662-47988-9. DOI: `10.1007/978-3-662-47989-6_34`. [Online]. Available: `http://link.springer.com/10.1007/978-3-662-47989-6_34` (visited on 04/30/2020).

[30] H. de Valence, I. Lovecruft, and T. Arcieri, *The Ristretto Group*. [Online]. Available: `https://ristretto.group` (visited on 05/03/2019).

[31] M. Abdalla, M. Bellare, and P. Rogaway, "The oracle diffie-hellman assumptions and an analysis of dhies," in *Cryptographers' Track at the RSA Conference*, Springer, 2001, pp. 143–158.

[32] D. J. Bernstein, "ChaCha, a variant of Salsa20," in *Workshop Record of SASC*, vol. 8, 2008, pp. 3–5.

[33] D. J. Bernstein, "The Poly1305-AES Message-Authentication Code," in *Fast Software Encryption*, H. Gilbert and H. Handschuh, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2005, pp. 32–49, ISBN: 978-3-540-31669-5. DOI: `10.1007/11502760_3`.

[34] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," IETF, RFC 8439, Jun. 2018. [Online]. Available: `https://tools.ietf.org/html/rfc8439` (visited on 10/09/2020).

[35] *Rust*. [Online]. Available: `https://www.rust-lang.org/`.

[36] Zkcrypto, *Zkcrypto/curve25519-dalek-ng: A pure-rust implementation of group operations on ristretto and curve25519*. [Online]. Available: `https://github.com/zkcrypto/curve25519-dalek-ng`.

[37] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2005, pp. 457–473. DOI: `10.1007/11426639_27`.

[38] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-Policy Attribute-Based Encryption," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, May 2007, pp. 321–334. DOI: `10.1109/SP.2007.11`.