UNIVERSITY
OF TURKU

Privacy-by-Design Regulatory Compliance Automation in Cloud Environment

Cyber Security
Master’s Degree Programme in Information and Communication Technology
Department of Computing, Faculty of Technology
Master of Science in Technology Thesis

Author:
Massimo Morello

Supervisors:
Petri Sainio (University of Turku)
Antti Hakkala (University of Turku)
Mohammed B.M. Kamel (ELTE)

May 2023

Master of Science in Technology Thesis
Department of Computing, Faculty of Technology
University of Turku

Subject: Cyber Security
Programme: Master’s Degree Programme in Information and Communication Technology
Author: Massimo Morello
Title: Privacy-by-Design Regulatory Compliance Automation in Cloud Environment
Number of pages: 69 Pages
Date: May 2023

Abstract:

The proposed Master's thesis revolves around the development of a privacy-preserving Attribute Verifier for regulatory compliance, first designed cryptographically, and then implemented in a Cloud Environment. The Attribute Verifier makes use of the Attribute Verification Protocol and its underlying encryption scheme, composed of Decentralized Attribute-Based Encryption (DABE) combined with a Zero-Knowledge Proof (ZKP) approach. The contribution of this work was integrating a ticketing system, concerning tickets of compliance, with the existing protocol, and automating the whole workflow, simulating all the actors involved, in AWS Cloud Environment.
The major goal was to improve the security and privacy of sensitive data kept in the cloud as well as to comply with Cloud Regulatory, Standards, and different Data Protection Regulations. In particular, the use case covered in this Thesis refers to the General Protection Data Regulation (GDPR), specifically the compliance with Article 32. The word "Automation" in the title refers to the achievement of having automated in AWS Cloud Environment, through code, three main security objectives: Privacy, Identity and Access Management, and Attribute-based Access Control. A goal that was pursued because, in the majority of the cases, adherence to a Regulatory still requires heavy manual effort, especially when it's about pure Data Protection Regulations, i.e. in a legal setting. And when the manual effort is not required, confidentiality can be still heavily affected, and that's where the need for a privacy-by-design solution comes from.
The Attribute Verifier was developed to verify the attributes of a Prover (e.g. a company, an institution, a healthcare provider, etc.) without revealing the actual attributes or assets and to grant access to encrypted data only if the verification is successful. The proposed example, among many applicable, it's the one a National Bank attempting to demonstrate to a Verifier, i.e. the European Central Bank, compliance with Article 32 of the GDPR.

Keywords: Attribute Verification, Privacy, Cloud Security, Cloud Compliance, GDPR.

Acknowledgements

Poserò la testa sulla tua spalla

E farò un sogno di mare

E domani un fuoco di legna

Perché l'aria azzurra

Diventi casa

Chi sarà a raccontare? Chi sarà?

Sarà chi rimane.

Io seguirò questo migrare

Seguirò

Questa corrente di ali

A **mamma** e **papà**, con gli occhi lucidi. All'amore incodizionato, infinito e trascendentale che mi ha accompagnato ad ogni passo e non mi ha mai lasciato solo, contro le logiche della geografia. Alla loro anima pura, privilegio che mi ha concesso dei valori che, durante il mio percorso di vita, ho capito quanto siano non negoziabili. All'ambiente che hanno creato, che ha annullato ogni possibilità di futuri grigi, ed ha protetto sempre la fantasia e la purezza. A mamma e papà, pensiero costante ad vittoria e sconfitta. La vostra gioia è la mia, e lo sarà per sempre.

A **Stefano** e **Fabrizio**, essere fratelli maggiori è un lavoro, ed il vostro amore fraterno, in tutte le sue forme possibili, non è mai mancato. All'infanzia in cui abbi-

amo disegnato con tutti i colori possibili, colori che abbiamo preservato con gelosia e che non sono mai scomparsi, perchè sono lì ogni qualvolta ci sediamo insieme a tavola. Al vostro dono dell'intelligenza, frutto della natura e di sacrifici, che vi ha resi il mio vanto, e mi ha spinto sempre a non accontentarmi mai. Infine, alla vostra attitudine naturale ad essere fratelli maggiori. Grazie per avermi sempre protetto.
A **Chiara**, alla quale mi lega un bene istantaneo e grandissimo che soltanto i cuori puri, pochi ed in via di estinzione, possono innescare. Al suo essere tanto persona quanto figura, di equilibrio e genuinità. Grazie perchè la sera del 23 agosto 2021 mi hai regalato insieme a Fabrizio, sulle note di "A Thousand Years", un frammento di ricordo lucidissimo, che è la mia nuova definizione della parole "amore".

Ancora una volta, a **Zio Gaetano**, da sempre esempio da seguire sotto ogni punto di vista. Alla sua capacità di voler bene in modo assordante ed al suo credere in me come in pochi hanno fatto. Al tuo credere in me quando la sera del 19 settembre del 2019 ti dissi che mi sarei probabilmente fermato alla triennale ed alla tua reazione paterna, che aveva già visto qualcosa che io non potevo vedere. Oggi le lauree magistrali sono due. Grazie, Zio.

Alla **Calabria**, terra alla quale mi lega un amore viscerale, che mi fa sentire parte di un qualcosa di trascendentale e che ho trovato dentro di me, e che mi ha fatto trovare uno scopo nobile che perseguirò per tutta la mia vita: preservarne la bellezza e la sua integrità. La Calabria mi ha regalato dei valori e dei punti di vista che si sono rivelati essere senza prezzo, e spero, un giorno, di poter ripagarne anche soltanto un granello di una spiaggia immensa.

Finally, this thesis work would not have been even remotely conceivable without the tireless support of **Professor Kamel**. A professor whom I have admired since the first day of class for his knowledge, his way of delivering it to students, and his empathy. Thank you once again for conceding me this opportunity, it was an honor working with you. Another important thanks goes to **Professor Sainio**, another huge example of how teaching should be approached: listening, stimulating curiosity, and opening up debates. Thank you for accepting this assignment, also in this case, It's an honor to have your name on this Master's Thesis. And to conclude, thanks to **Ammar**, worker and family man, who still, inexplicably, decided to dedicate some time to me and motivate me. Thank you for your support and the empathy you showed me.

Contents

List of Figures

List of Codes

1 Introduction

This chapter delves into the motivations behind the work, the research questions objectives it addresses, with related contributions, and starts giving some background information that makes the reader aware of the protocols that were adopted.

1.1 Background and motivation

As companies depend on the scalability and accessibility of cloud-based services to store and manage their sensitive data, the cloud environment has grown more and more common. Nevertheless, because of the broad usage, there is now a number of security and privacy issues, many of them related to data security and regulatory compliance (data must be protected and kept private, especially in accordance with various regulations such as NIST, GDPR, SOC2, etc.). Using cutting-edge cryptographic methods like Zero-Knowledge Proof (ZKP) and Decentralized Attribute-Based Encryption (DABE) [1] is one way to overcome these issues.
ZKP is a cryptographic mechanism that enables a Prover to show a Verifier that they possess a certain piece of information or attribute without actually disclosing the information itself. The definition of a Zero-Knowledge Proof is one that conveys just the claim in issue is true and nothing more [2]. DABE, on the other hand, is a form of encryption technique that restricts access to encrypted data to individuals who have particular credentials or attributes while keeping the details of those credentials or attributes private. Ciphertexts are labeled with sets of attributes and

private keys are linked to access structures that have the role to control which ciphertexts a user can decrypt [3], and the key issuance is managed in a decentralized setting, allowing multiple issuers to become part of the scheme.
The solution proposed in this Master's Thesis leverages the Attribute Verification Protocol [4], composed in particular by DABE, and a ZKP approach to enhance the privacy and security of sensitive data stored in the cloud. Specifically, Attribute Verification Protocol will verify the attributes of a Prover without revealing the actual attributes and grant access to encrypted data only if the verification is successful [5]. The Attribute Verifier will be designed in a versatile way, applicable in many use cases, to verify the attributes of certain assets of a company (e.g. an article of a regulation, the passwords of the employees, the encryption algorithms they use, etc.) and ensure that they meet the requirements set by the regulations.
The findings of this Master's thesis will give a contribution to the field of Cloud Security, privacy-enhancing technologies, and mainly Regulatory Compliance (in particular, GDPR). First, it will provide a detailed overview of ABE, with the Decentralized version of it, and ZKP techniques and their application in cloud environments. Second, it will implement and evaluate the proposed solution, providing insights into the challenges and limitations of using DABE and ZKP in cloud environments. Finally, it will contribute to the development of a more privacy-preserving way of verifying compliance with regulations in the cloud.
The fundamental driver behind the study is the necessity for privacy-preserving technologies that guarantee confidentiality for the company that is being audited for regulatory compliance checks, while still allowing a smooth assessment of the company assets.

1.2 Research Objectives

The security objectives that have been identified throughout the study of the challenges and the conceptual design of a feasible solution to overcome them, are the followings:

1. Conducting a comprehensive literature review of cloud compliance and regulations, and providing an overview of Zero-Knowledge Proof and Decentralized Attribute-Based Encryption;

2. Investigating relevant studies on privacy-preserving Attribute Verification, and Attribute Verification Protocol;

3. Designing and developing an Attribute Verifier that incorporates DABE and ZKP protocols for privacy-preserving Attribute Verification;

4. Evaluating the proposed Attribute Verifier, comparing it with similar existing solutions, and discussing the results and implications of the study;

5. Demonstrating the effectiveness of the proposed system by conducting experiments in AWS cloud environment, and evaluating the system's security and privacy.

The outcomes of this research will result in a secure and privacy-preserving Attribute Verifier. The study will contribute to the development of privacy-by-design systems and will have implications for Regulatory Compliance Automation and Cloud Security.

1.3 Research Questions

The overall objective of the Thesis is to enhance privacy, and so the security of sensitive data stored in the cloud, which needs to be verified for cloud regulatory, data protection, or standard purposes. In this section, the research questions that this thesis aims to answer are outlined, together with the specific objectives of this research. The Research Questions for this Master's Thesis have been identified in the followings:

- How can Decentralized Attribute-Based Encryption and Zero-Knowledge Proof be used to implement an Attribute Verifier for Regulatory Compliance and Privacy purposes?
- What are the challenges in developing and integrating an Attribute Verifier in a Cloud Environment?
- How can the Attribute Verification Protocol be utilized to automate compliance verification of specific parts from certain regulations?
- How to ensure the privacy of the attributes being verified, while still allowing the Verifier to determine if the attributes are indeed held by the Prover?

1.4 Research Contributions

The main contributions of this research start with, first of all, working on the Attribute Verification Protocol, tailoring it to specific needs (Regulatory checks, Cloud Environment settings), integrating a ticketing system in its scheme, and testing its n-out-of-n type for compliance purposes (access is granted if, out of n compliant attributes, the Prover is in possession of at least one of them) [4] and applied to a set of attributes belonging to certain specific security controls taken from the regulations (e.g. GDPR).

The Attribute Verifier was developed in order to verify the attributes of a Prover without making this Prover reveal them, and to grant access to encrypted data only if the verification is successful. Another important contribution comes from the practical implementation of it: the proposed Attribute Verifier was practically implemented and evaluated in AWS cloud environment, to demonstrate its feasibility and effectiveness in supporting customized cryptographic solutions for enhancing the privacy of sensitive data. In particular, the three actors involved (Issuer, Prover, Verifier) were implemented as separate Lambda functions, and integrated in order to communicate with each other. The evaluation of the solution, in Chapter 3, will also highlight the implications of the study for cloud security and regulatory compliance.

1.5 Thesis Outlines

The outlines of this Master's thesis are organized as follows: the Thesis begins with an introduction, where the research topic, context, and motivation behind the development of a privacy-preserving Attribute Verifier for regulatory compliance in Cloud Environment are presented. This section also states the research objectives and questions, provides an overview of the main contributions of the research, and outlines the structure of the thesis.

Next, a literature review is presented, offering an overview of cloud computing security, compliance, and regulations. Zero-Knowledge Proof (ZKP) protocols and Attribute-Based Encryption (ABE) schemes are reviewed in deep, delving into the Decentralized version of ABE (DABE). It follows a review of relevant studies on privacy-preserving Attribute Verification.

In the following section, an adaptation of the Attribute Verification Protocol for regulatory compliance verification is explained. And by "adaptation" is intended a tailored and enhanced version of the skeleton of the protocol, with the addition of a ticketing system and an emphasis on the Decentralized version of ABE, for multiple Issuer. The enhancement concerned also the ability to deploy the Proposed Protocol and all its components (i.e. all the actors involved) in a cloud environment, testing the whole workflow behind it. The proposed modified protocol, which still preserves its underlying encryption scheme, is applied in the context of the banking systems, delving into a use case scenario involving any national Bank as Provers, and the European Central Bank as Verifier. This use case, though, doesn't limit at all the possibility to adopt the proposed solution to other scenarios and other fields, like healthcare, industry, and any setting where privacy-by-design solutions and confidentiality are a necessary requirement. The implementation of the Issuer, Prover, Verifier, and Orchestrator is then detailed, providing insights into the development of the system.

The Thesis then proceeds to the evaluation and analysis, where the proposed system and its effectiveness in privacy-preserving Attribute Verification are evaluated, and then, the proposed system is compared with existing solutions. The results and implications of the study in relation to cloud security, privacy, and regulatory compliance, are discussed.

Finally, the Thesis concludes with a summary of the main findings and contributions of the research. Potential future research directions in privacy-preserving Attribute Verification and cloud security are also discussed. This outline serves as a roadmap to guide readers through the main sections and subsections of the thesis.

2 Literature Review

This chapter contains a summary of the cloud-related objectives, like security, data protection, and compliance with regulations. Subsequently, it introduces a deep analysis of all the cryptographic primitives and algorithms necessary for implementing, and then integrating, the Attribute Verification Protocol.

2.1 Cloud Security, Compliance, and Regulations

Cloud computing is an abstraction of computing, storage, and network infrastructure put together as a platform that allows for speedy application deployment and dynamic scaling. A Key feature of cloud computing is its way of being self-service: the end user just fills in a form and it will already be up and running. Some clear advantages of this technology are: scalability, cost-savings, accessibility, flexibility, disaster recovery, environmental sustainability. The largest part of cloud users uses cloud computing services, which are housed in sizable, distant data centers that are kept up by the providers, through the internet [6]. This is the case of the public cloud model, and there are indeed three models: Public (the cloud service is provided by a third-party cloud service provider, i.e. CSP), Private, where internal users receive services from a company's data center (the user creates and manages its own underpinning cloud infrastructure), and Hybrid, that combines on-premises private clouds with public cloud services (businesses can use the public cloud to accommodate workload peaks or demand rises while running mission-critical workloads or sensitive applications on the private cloud)

The most important distinction in technical terms is about the services the cloud provider offers [6]:

1. IaaS (Infrastructure as a Service): Providers of IaaS, like Amazon Web Services (AWS), offer virtual server instances, storage, and application programming interfaces (APIs) that let customers move workloads to virtual machines (VM). Users are given a certain amount of storage space and for different workload requirements, IaaS providers provide small, medium, big, extra-large, and memory- or compute-optimized instances in addition to providing instance customization. For commercial customers, the IaaS cloud model is the most similar to a remote data center;

2. PaaS (Platform as a Service). The PaaS concept places development tools on the infrastructure of cloud providers. Using APIs, web portals, or gateway software, users may access these tools online. PaaS is utilized for the creation of all types of software, and several PaaS service providers host the finished product. Salesforce's Lightning Platform, Amazon Elastic Beanstalk, and Google App Engine are examples of popular PaaS platforms;

3. SaaS (Software as a service): SaaS is a way of distributing software via the internet, and that's why it's many times referred to as "web service". Users can use a PC or mobile device with internet connectivity to have access to SaaS apps and services from any place, and get access to databases and applications. Microsoft 365, with its productivity and email capabilities, is a clear example of SaaS application features.

Cloud Security

To safeguard cloud-based systems, applications, data, and infrastructure against unauthorized access, theft, damage, or data loss, a collection of rules, technologies, and controls is used in the cloud security framework. Making sure cloud security is a priority for every IT strategy has grown more and more important as more businesses move their data and workloads to the cloud. A variety of security domains are included in cloud security, including identity and access management, data protection, threat detection and response, network security, and compliance management. Making sure that sensitive data is protected from unwanted access is one of the main goals of cloud security. A common method for protecting data both at rest and while it is transiting, is encryption, and to guarantee that encryption keys are adequately secured, secure key management procedures must also be adopted: access controls, firewalls, intrusion detection and prevention systems, automated security monitoring and alerting are a few of the security features that cloud providers include.
Identity and access management (IAM) policies must be implemented correctly in order to ensure cloud security, as these policies leverage authentication and authorization protocols, multi-factor authentication, and role-based access control to regulate user access based on specified rules and policies [7]. Access controls, firewalls, intrusion detection and prevention systems, automatic security monitoring, and alerts are just a few of the security features that cloud providers grant to end users.
Many technological obstacles must be overcome in order to secure cloud systems. Because cloud computing is shared, conventional perimeter-based security procedures are not sufficient, and so a defense-in-depth strategy that combines controls at many infrastructure tiers is required. Data availability, confidentiality, integrity, and compliance with rules and standards must all be guaranteed. In addition, effective countermeasures are required for new threats, including supply chain attacks,

insider threats, and zero-day vulnerabilities.

Cloud Regulations/Regulatory

As cloud computing continues to gain traction as a delivery model for IT services, regulatory frameworks have emerged, in order to ensure the safety and privacy of data stored and processed in the cloud. These regulations draw guidelines and requirements for cloud service providers and their customers to adhere to ethical, legal, and operational standards, like the General Data Protection Regulation (GDPR), the Health Insurance Portability and Accountability Act (HIPAA), the Payment Card Industry Data Security Standard (PCI-DSS), and the Federal Risk and Authorization Management Program (FedRAMP).

For the use case of this Thesis, the attention was dedicated to GDPR [8]: launched on May 25, 2018 by the European Union (EU), GDPR is a comprehensive data protection and privacy policy. With the GDPR, companies that collect, handle, and keep personal data are subject to stringent regulations for data controllers and processors, such as the need for explicit consent, data minimization, data portability, and the right to be forgotten, aiming to protect the privacy of EU citizens and residents [8]. Organizations both inside and outside of the EU that provide products or services to, or keep an eye on, EU data subjects' conduct must comply with the regulation. Failure to comply with GDPR regulations can result in substantial fines of up to 4% of global annual revenue or €20 million, whichever is greater.

Overall, regulatory frameworks such as these are critical in ensuring the safety and privacy of data stored and processed in the cloud. By defining clear guidelines and requirements for cloud service providers and their customers, these regulations foster trust in the cloud and protect sensitive data from unauthorized access, disclosure, and misuse.

Cloud Compliance

Ensuring cloud compliance is a crucial aspect of cloud computing, and it may require cloud service providers (CSPs) to meet various industry standards and regulatory frameworks, such as NIST, GDPR, SOC2, HIPAA, and PCI DSS, depending on the type of data they process, store, or transmit via the cloud.

One of the most extensively used compliance frameworks is the NIST Cybersecurity Framework, which provides guidance for improving the security and resilience of critical infrastructure [9]. This framework includes instructions for locating, protecting, spotting, reacting to, and recovering from cybersecurity events, and it also incorporates particular controls and specifications for cloud computing, like data encryption, access control, and monitoring.

In terms of GDPR, "cloud compliance" refers to ensuring that businesses employing cloud-based infrastructure and services adhere to the regulations set forth in the GDPR while handling and keeping personal data. Data protection and privacy must be upheld in the cloud on behalf of both cloud service providers and their clients [8].

Aside from these compliance frameworks, CSPs must comply with other regulations depending on the industry they serve. For example, healthcare organizations must comply with HIPAA, while payment card processors must adhere to the PCI DSS.

CSPs must conduct ongoing monitoring, assessment, and audit of their security controls and processes to ensure compliance with these regulations. Adherence to several legal frameworks and industry standards it's required for cloud compliance, which is a complicated and ongoing process. To comply with these requirements, CSPs must install strong security measures and submit to routine audits, which can assist to increase customers' and stakeholders' trust.

2.2 Zero-Knowledge Proof (ZKP) Protocols

ZKP is a cryptographic protocol that allows one party (i.e. the Prover) to demonstrate to another party (i.e. the Verifier) that they know a secret, without actually disclosing that secret. This is achieved by employing probabilistic calculations and complex mathematical operations like those inherent to public key cryptography. ZKP is a notion that was initially proposed in 1985 by Goldwasser, Micali, and Rackoff [10] with the following definition: *A zero-knowledge proof system for a language L is a probabilistic interactive proof system with the property that there exists an efficient algorithm which, given any input $x \in L$, produces a transcript which convinces the verifier of the truth of the statement x, revealing no additional information about x.*

Since then, it has been extensively researched and used in the fields of cryptography, privacy, and authentication. The Prover receives a challenge that is generated randomly as part of the protocol, it proceeds to compute an answer and send it back to the Verifier using their understanding of the secret. The Verifier, in turn, will compare the response to the challenge, sent by the Prover, in order to verify the Prover's truthfulness without actually knowing the secret. Depending on whether the proof is established in a single message exchange or needs a back-and-forth exchange of messages, ZKP protocols can be interactive or non-interactive.

Regardless of the interactivity or non-interactivity, A zero-knowledge protocol must meet the following requirements [11]:

- Completeness: The zero-knowledge protocol always returns "true" if the input is valid. Therefore, if the underlying claim is true, and both the parties (Prover and Verifier) act honestly, the proof can be accepted;

- Soundness: It is theoretically infeasible to trick the zero-knowledge protocol into returning "true" if the input is invalid. Therefore, a dishonest Prover cannot deceive a truthful Verifier into accepting a false claim as true (except with a tiny margin of probability);

- Zero-knowledge: The Verifier gains no further information about a claim other than whether it is true or false (they have "zero knowledge of the claim"). Thanks to this condition, the Verifier cannot derive the original input (the contents of the statement) from the proof.

Now, from the mathematical point of view [12], let's consider a prover, P, who wants to convince a Verifier, V, that they know a secret value x, without revealing anything about x to V :

P generates a random value r.

P computes a "commitment" value $c = g^x * h^r$, where g and h are fixed publicly known values.

P sends the commitment value c to V.

V generates a random challenge value e, and sends e to P.

P computes a "response" value $s = r + ex$, and sends s to V.

V checks that $c = g^x * h^r$ and $c' = g^s * h^e$. If both equations hold, V accepts the proof.

To see why this protocol works, it needs to be considered that the commitment value c can be thought of as a "locked box" containing the value x and r. P can generate any c value they want, but once they have done so, they cannot change x or r without breaking the equation $c = g^x * h^r$.

When V sends the challenge value e, P is forced to compute the response value s that correctly satisfies the equation $c' = g^s * h^e$. Let's note that that s can be computed as $s = r + ex$, since e is known to both P and V.

If P knows the value x, then they can compute r and s that satisfy the equations, and V can verify the proof by checking that both equations hold. On the other hand, if P doesn't know x, then they will not be able to compute s correctly, and V will reject the proof.

ZKP can be differentiated in Interactive and Non-Interactive [13]:

- In interactive ZKP, the Prover and the Verifier engage in a series of interactive rounds, where the Prover sends some information to the Verifier, and the Verifier sends back a challenge based on the information received. This continues for a number of rounds until the Verifier is convinced that the Prover knows the secret information.
- In contrast, non-interactive ZKP (NIZK) is a proof system that requires only one round of communication between the Prover and the Verifier. In NIZK, the Prover creates a proof without any interaction with the Verifier, and sends the proof to the Verifier. The Verifier checks the proof and accepts or rejects it based on the validity of the proof. The advantage of NIZK is that it requires less communication between the Prover and Verifier, making it more efficient for certain applications. However, constructing NIZK is generally more difficult than constructing interactive ZKP, and there are fewer known constructions of NIZK compared to interactive ZKP. In addition, NIZK may suffer from a problem known as the "random oracle model" (ROM) assumption, which assumes the existence of a random oracle, i.e. a function that outputs random values for each unique input. The use of random oracles in NIZK may weaken the security of the protocol, and there is ongoing research on constructing NIZK without relying on the ROM assumption [14].

Password-based authentication, electronic payment systems, digital signatures, secure calculations, and anonymous transactions are just a few of the many uses for ZKP. It is especially helpful with blockchain technology since it may provide transaction privacy and anonymity while upholding the safety and integrity of the blockchain.

2.2.1 Schnorr Protocol

The Schnorr protocol [12] allows a Prover to prove to a Verifier that they know the discrete logarithm of a given element in a group, without revealing any information about the logarithm. The protocol proceeds as follows:

1. The prover chooses a random value $r \in \mathbb{Z}_q$, where q is a large prime number. They compute $t = g^r$, where g is a generator of the group;
2. The prover sends t to the verifier;
3. The verifier chooses a random challenge value $c \in \mathbb{Z}_q$ and sends it to the prover;
4. The prover computes $s = r + cx \bmod q$, where x is the discrete logarithm they are trying to prove knowledge of;
5. The prover sends s to the verifier;
6. The verifier checks whether $g^s = tx^c$. If the equation holds, the verifier accepts the proof;

This protocol is secure under the discrete logarithm assumption in the group, meaning that an attacker who does not know the discrete logarithm cannot forge a proof. However, the protocol is interactive, meaning that it requires multiple rounds of communication between the prover and verifier.

2.2.2 Fiat-Shamir Heuristic

The Fiat-Shamir heuristic [13] is a method for converting interactive ZKP protocols into non-interactive ones. The basic idea is to replace the random challenge value chosen by the verifier with a hash function of the prover's previous messages and a random seed value. The resulting protocol has only one round of communication, and the prover's messages are generated solely based on their secret knowledge.

For example, to apply the Fiat-Shamir heuristic to the Schnorr protocol, we would modify steps 3-5 as follows:

1. The prover chooses a random seed value $s \in \mathbb{Z}_q$ and computes $h = H(g, t)$, where H is a hash function. They send h to the verifier
2. The verifier computes $c = H(h, x)$ and sends c to the prover
3. The prover computes $s = r + cx \bmod q$, and sends s to the verifier
4. The verifier checks whether $g^s = th^c$. If the equation holds, the verifier accepts the proof

The security of the resulting non-interactive protocol depends on the collision resistance and random oracle properties of the hash function used. If the hash function is secure, then the resulting protocol is also secure, and is often more practical and efficient than the original interactive protocol.

2.2.3 zk-SNARK

zk-SNARK stands for Zero-Knowledge Succinct Non-Interactive Argument of Knowledge, and it's a type of non-interactive zero-knowledge proof system that allows for the verification of a statement without revealing any information beyond the validity of the statement itself [15]. The main idea behind it is to represent the statement to be proven as a Boolean circuit. The circuit is then transformed into a polynomial representation, using a process called "arithmetization". The polynomial is then used to create a proof that can be verified in constant time and space, regardless of the size of the circuit. The security of zk-SNARKs depends on the difficulty of specific mathematical challenges, such as the discrete logarithm problem and the elliptic curve discrete logarithm problem. The polynomial representation can be made safe against attackers with sufficient computational power by selecting the proper parameters. The use of zk-SNARKs in bitcoin transactions is one of its main applications.

Without disclosing the sender's and recipient's addresses or the transaction's value, zk-SNARKs can be used to demonstrate a transaction's legitimacy. This protects the transaction's integrity while offering a high level of secrecy.

A zk-SNARK consists of three algorithms [16]:

- A Key Generation algorithm G that takes as input a security parameter k and generates a public and private key pair (pk, sk)
- A Proving Algorithm P that takes as input a statement x and a witness w, and generates a proof π that can be verified
- A Verification Algorithm V that takes as input a statement x, a proof π, and a public key pk, and outputs either "accept" or "reject

Let's denote a statement by x, a witness by w, and a proof by π. We can represent the Proving Algorithm P as $P(x, w) = \pi$ and the verification algorithm V as $V(x, \pi, pk) = 0, 1$.

2.3 Attribute-Based Encryption (ABE)

Attribute-Based Encryption (ABE) is a type of public-key cryptography that enables users to encrypt and decrypt data based on attributes or characteristics rather than using specific identities or keys. In ABE, a user's secret key is associated with a set of attributes, like age, gender, job title, or any other characteristic that may be used to define a group of users. While ciphertexts are encrypted with access policies that define which attributes are required to decrypt the data. A user can decrypt the ciphertext if their attributes satisfy the access policy [17]. ABE can be divided into two primary types:

- Key-Policy Attribute-Based Encryption (KP-ABE): In KP-ABE, the access policy is embedded in the user's private key, and the ciphertext is associated with a set of attributes. A user can decrypt the ciphertext if their private key's access policy is satisfied by the attributes associated with the ciphertext.

- Ciphertext-Policy Attribute-Based Encryption (CP-ABE): In CP-ABE, the access policy is embedded in the ciphertext, and the user's private key is associated with a set of attributes. A user can decrypt the ciphertext if the attributes in their private key satisfy the access policy associated with the ciphertext.

The ABE protocol, overall, works as follows:

1. Setup: A trusted authority initializes the system by generating public parameters and a master secret key. Public parameters include information about the bilinear group, generators, and a bilinear map. The master secret key is kept secret by the trusted authority.

2. Key Generation: The trusted authority generates a private key for each user based on their attributes. In KP-ABE, the private key embeds an access policy, while in CP-ABE, the private key is associated with a set of attributes.

3. Encryption: The data owner encrypts the data using the public parameters and an access policy (in CP-ABE) or a set of attributes (in KP-ABE). The encrypted data is called ciphertext.

4. Decryption: A user can decrypt the ciphertext using their private key if their attributes (in CP-ABE) or access policy (in KP-ABE) satisfy the requirements specified during the encryption process. If the requirements are not met, the user cannot decrypt the ciphertext.

Let M be the message to be encrypted and let S be the set of attributes required to decrypt the message. The key generation algorithm takes as input a set of attributes A and an access policy Γ, and generates a secret key SK that can be used to decrypt any ciphertext that satisfies the access policy. The encryption algorithm takes as input the message M and the set of attributes S, and produces a ciphertext C that can be decrypted by a user with a secret key SK that has attributes that satisfy the access policy S.

ABE can be used in many different applications, such as Access Control systems, secure communication systems, and cloud storage.
The main advantage of ABE is that it allows for fine-grained access control to data, allowing users to encrypt and decrypt only the data they are authorized to access, based on their attributes. This makes ABE an ideal encryption scheme for applications where data must be securely shared among a large number of users with varying levels of authorization. Additionally, ABE can be used to enforce privacy policies, where sensitive data can be encrypted and shared only with users who have specific attributes, such as those who work in a particular department or have a certain security clearance.

2.3.1 Key-Policy Attribute-Based Encryption (KP-ABE)

KP-ABE is a type of Attribute-Based Encryption where a user's private key is associated with an access structure, typically defined by a monotonic access tree, and ciphertexts are associated with sets of attributes. A user can decrypt a ciphertext if the attributes associated with the ciphertext satisfy the access structure of their private key.

The KP-ABE scheme consists of the following algorithms:

Setup: This algorithm takes a security parameter as input and outputs the public parameters and a master secret key. The public parameters include a bilinear group, a bilinear map, random generators, and a hash function.

Key Generation: This algorithm takes the master secret key, the public parameters, and an access structure as input. It generates a private key for a user, which is associated with the given access structure. The access structure is usually defined by a monotonic access tree, where the internal nodes represent threshold gates and the leaf nodes represent attributes.

Encryption: This algorithm takes the public parameters, a message, and a set of attributes as input. It encrypts the message and outputs a ciphertext associated with the given set of attributes.

Decryption: This algorithm takes the public parameters, a user's private key (with its associated access structure), and a ciphertext as input. If the user's access structure is satisfied by the set of attributes associated with the ciphertext, the algorithm decrypts the ciphertext and retrieves the original message.

The main idea behind KP-ABE is to allow fine-grained access control over encrypted data by associating access policies with private keys and attributes with ciphertexts. This enables data owners to enforce access policies without having to know the exact set of users that can access their data, and users only need to possess the appropriate attributes to decrypt the data. The security of KP-ABE relies on the hardness of certain mathematical problems, such as the Decisional Bilinear Diffie-Hellman (DBDH) assumption [18]. Specifically, the security of KP-ABE is based on the assumption that it is computationally infeasible to derive any information about the secret key or the plaintext given the ciphertext and the access policy.

2.3.2 Ciphertext-Policy Attribute-Based Encryption (CP-ABE)

CP-ABE is a type of Attribute-Based Encryption where a user's private key is associated with a set of attributes, and ciphertexts are associated with an access structure, typically defined by a monotonic access tree. A user can decrypt a ciphertext if their private key's attribute set satisfies the access structure associated with the ciphertext.

The CP-ABE scheme consists of the following algorithms:

Setup: This algorithm takes a security parameter as input and outputs the public parameters and a master secret key. The public parameters include a bilinear group, a bilinear map, random generators, and a hash function.

Key Generation: This algorithm takes the master secret key, the public parameters, and a set of attributes as input. It generates a private key for a user, which is associated with the given set of attributes.

Encryption: This algorithm takes the public parameters, a message, and an access structure as input. It encrypts the message and outputs a ciphertext associated with the given access structure. The access structure is usually defined by a monotonic access tree, where the internal nodes represent threshold gates and the leaf nodes represent attributes.

Decryption: This algorithm takes the public parameters, a user's private key (with its associated set of attributes), and a ciphertext as input. If the user's attribute set satisfies the access structure associated with the ciphertext, the algorithm decrypts the ciphertext and retrieves the original message.

The main idea behind CP-ABE is to allow fine-grained access control over encrypted data by associating access policies with ciphertexts and attributes with private keys. This enables data owners to enforce access policies directly on the encrypted data, ensuring that only users with the appropriate attributes can decrypt the data.

2.3.3 Decentralized Attribute-Based Encryption (DABE)

Decentralized Attribute-Based Encryption (DABE), i.e. the protocol that is part of the underlying encryption scheme of the Attribute Verification Protocol adopted throughout this Thesis, is an extension of ABE schemes that allows for the decentralized management of attributes and access policies. In fact, in DABE, multiple authorities can issue attributes independently and no central authority is required. This enhances privacy and scalability in comparison to centralized ABE schemes. DABE consists of four main algorithms [19]: Setup, Key Generation, Encryption, and Decryption.

1. Setup: Each authority generates its public parameters and a master secret key. The public parameters are shared among all the authorities and users in the system.

2. Key Generation: Users obtain secret keys from multiple authorities based on their attributes. Each authority issues a partial secret key corresponding to the user's attributes under its jurisdiction. The user then combines the partial secret keys to obtain a complete secret key.

3. Encryption: The data owner encrypts the data using a policy that defines the attributes required to decrypt the data. The policy can be expressed as an access structure, such as a threshold gate or a monotonic boolean formula.

4. Decryption: A user can decrypt the encrypted data if their secret key satisfies the access policy associated with the data. The user combines the components of their secret key that correspond to the attributes in the access policy to recover the decryption key and decrypt the data.

In mathematical terms [1], the logic is the following:
Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic groups of prime order p, and let g be a generator of $\mathbb{G}$. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map. We use a set of attributes $\mathcal{A}$ and a set of authorities $\mathcal{N}$.

1. Setup: Each authority $i \in \mathcal{N}$ generates its public parameters and master secret key as follows:

   - Choose random $y_i, t_i \in \mathbb{Z}_p$.
   - Compute $Y_i = g^{y_i}$ and $T_i = g^{t_i}$.

   The public parameters are (g, Y_i, T_i), and the master secret key is (y_i, t_i).

2. Key Generation: For a user with attributes $A_u \subseteq \mathcal{A}$, the user obtains a secret key as follows:

   - For each authority $i \in \mathcal{N}$ and each attribute $a \in A_u$, the authority computes a partial secret key: $K^i_{u,a} = T_i^{r_{u,a}} \cdot Y_i^{r_{u,a} \cdot H(a)}$, where $r_{u,a} \in \mathbb{Z}_p$ is a random value and $H : \mathcal{A} \to \mathbb{Z}_p$ is a hash function.
   - The user combines the partial secret keys to obtain the complete secret key: $K_{u,a} = \prod_{i \in \mathcal{N}} K^i_{u,a}$

3. Encryption: The data owner encrypts a message $m \in \mathbb{G}_T$ using an access policy $\mathcal{P}$ as follows:

   - Choose a random $s \in \mathbb{Z}_p$.
   - Compute the ciphertext components: $C_0 = m \cdot e(g, g)^s$ and for each attribute $a \in \mathcal{A}$ in the access policy, $C_a = g^{s \cdot H(a)}$.

   The ciphertext is $(C_0, \{C_a\}_{a \in \mathcal{A}})$.

4. Decryption: A user with a secret key K_u can decrypt the ciphertext if their attributes satisfy the access policy $\mathcal{P}$:

- The user computes the decryption key $D = \prod_{a \in \mathcal{A}_u} e(K_{u,a}, C_a)$, where $\mathcal{A}_u$ is the set of attributes that satisfy the access policy.
- The user recovers the message as $m = \frac{C_0}{D}$.

2.4 Attribute Verification Protocol

The Attribute Verification Protocol is a cryptographic protocol used to verify the possession of certain attributes by a user without disclosing the actual attribute values. This protocol is particularly useful for privacy-preserving applications where revealing the user's attributes might lead to privacy breaches. The protocol generally involves an attribute holder (user) and an Attribute Verifier (service provider). The Attribute Verification Protocol aims to achieve the following security properties: Completeness (a legitimate Prover who possesses the required attributes can always convince the Verifier), Soundness (a malicious Prover who does not possess the required attributes cannot convince the Verifier), Zero-Knowledge (the Verifier learns nothing about the user's attributes except that they satisfy the verification requirements), and Privacy (the Verifier cannot link the user's verification requests to their actual identity, ensuring unlinkability and anonymity) [5].

Several cryptographic primitives can be used to build Attribute Verification Protocols, such as ZKPs, group signatures, and anonymous credentials. The choice of the underlying primitive depends on the specific requirements and security assumptions of the application.

An Attribute Verification protocol can be built using a combination of Decentralized Attribute-Based Encryption (DABE) and Zero-Knowledge Proofs (ZKP): DABE provides fine-grained access control based on attributes, while ZKP allows users to prove possession of attributes without revealing the actual values. Here's an outline of how an Attribute Verification Protocol can be composed of ABE and ZKP:

- DABE for attribute management and encryption: DABE is used to manage attributes and control access to encrypted data. Prover's private decryption keys, managed in a decentralized way (among multiple Issuers), are associated with their attributes, and data is encrypted based on access policies. A Prover can decrypt the ciphertext if its attributes satisfy the access policy.

- ZKP for attribute verification: Zero-Knowledge Proofs are used to prove the possession of attributes required by the access policy without revealing the actual attribute values. The user engages in an interactive protocol with the Verifier, and the Verifier can be convinced that the user possesses the required attributes without learning any additional information.

The combination of DABE and ZKP in an Attribute Verification Protocol provides several advantages:

1. Fine-grained access control: DABE enables the data owner to define complex access control policies based on the attributes of the users.

2. Privacy-preserving attribute verification: ZKP allows Provers to prove they possess the required attributes without revealing the actual attribute values, thus preserving privacy.

3. Collision resistance: The combination of DABE and ZKP ensures that users cannot combine their decryption keys to decrypt data that they are individually unauthorized to access.

4. Scalability: The combination of DABE and ZKP can handle a large number of attributes and complex access policies, making it suitable for applications with diverse and dynamic user bases.

In this protocol, we assume a setup that involves an Issuer, a Prover, and an

Attribute Verifier. Here's a description of an Attribute Verification protocol using DABE and ZKP:

1. Setup: A trusted authority initializes the system by generating public parameters and master secret keys for the DABE scheme. The authority also provides the required cryptographic primitives and parameters for the ZKP protocol.

2. Attribute Issuance: Provers obtain their attributes from the attribute Issuer, and because of DABE, the Issuers can be multiple. These attributes are associated with users' private decryption keys for the DABE scheme. The trusted authority can generate these keys based on the user's attributes and the master secret key.

3. DABE Encryption: A Prover requests access to encrypted data, and the Verifier, who is responsible for controlling access to the encrypted data, encrypts the data using an access policy (defined over attributes) and the public parameters of the DABE scheme. In particular, the Verifier is responsible for the following sub-steps:

   - Challenge: The attribute Verifier sends a random challenge to the Prover. This challenge ensures that the user cannot predict the Verifier's queries beforehand, preventing cheating.
   - Verification: The attribute Verifier verifies the user's Zero-Knowledge Proof using the public parameters. If the verification is successful, the Verifier concludes that the Prover indeed possesses the required attributes.

4. Data Decryption: If the attribute verification is successful, the Prover receives the encrypted data. They can use their private decryption key associated with the attributes to decrypt the data, as their attributes satisfy the access policy defined by the Verifier. In particular, the Prover is responsible for the following sub-steps:

- Commitment: The Prover commits to their attributes using a secure commitment scheme, generating a commitment value that binds them to the attributes without revealing the actual values.
- Proof Construction: The Prover constructs a Zero-Knowledge Proof using their committed attributes, the challenge from the Verifier, and the public parameters. This proof demonstrates that the Prover possesses the required attributes without revealing the actual attribute values.

5. Access Decision: Based on the Attribute Verification result, i.e. on the eventual successful decryption performed from the Prover, the attribute Verifier concludes whether the Prover fulfills the access policy that was previously set, or not.

In summary, an Attribute Verification Protocol composed of DABE and ZKP can achieve fine-grained access control and privacy-preserving attribute verification. The combination of these cryptographic techniques allows for secure and efficient access control in privacy-sensitive applications.

2.5 Privacy-Preserving Attribute Verification

The paper "Decentralizing Attribute-Based Encryption" Lewko and Waters [1] took care of introducing in detail the concept of a Decentralized ABE scheme, which built the foundations for the Decentralized Attribute-Based Encryption (DABE). The authors proposed a new ABE scheme that eliminates the need for a central authority to manage attributes and keys, so any party or node could become an Issuer (or Attribute Authority) and issue secret keys for their attributes, making the system more flexible and scalable compared to previous centralized ABE schemes. One of the emphasized key features of this DABE scheme was its collusion-resistant design, meaning that users cannot combine their secret keys to gain unauthorized access to encrypted data. Another important aspect of the proposed scheme was its support for expressive access policies. This means that the DABE scheme can handle a wide range of conditions for granting access to encrypted data. Specifically, it can support any access policy that can be represented by a monotonic access structure, which allows for complex and flexible policy definitions.

Another very important paper for this work, written by Kamel [4] introduced a Decentralized Attribute Verification (relying on DABE) system for IoT applications that employs a challenge-response technique. By addressing privacy and scalability issues often associated with centralized models, the system provides two verification modes: 1-out-of-n and n-out-of-n, enabling participants to demonstrate possession of one or all specified target attributes. The model includes three types of participants: Provers, Issuers, and Verifiers. Provers confirm attribute ownership by responding to challenges sent by Verifiers, Issuers supply attribute proofs (secret keys associated to the attributes) to Provers, and Verifiers validate Prover attribute ownership via challenges. A node may join the system as Prover, while organizations can become independent Issuers for certain attributes, providing secret keys for confirmed attributes to users.
In the 1-out-of-n mode, Provers must demonstrate ownership of at least one of the provided attributes, while in the n-out-of-n mode, they must possess all given attributes. The model's Provers are independent nodes with a Prover Attribute Vector (PAV), a private collection of attributes issued by different issuers [4]. The system seeks soundness, which prevents the successful verification of malicious provers, while also satisfying unlinkability and untraceability properties. Soundness guarantees that a Prover can only provide a verifiable PAV for forged attributes with negligible probability. Unlinkability prevents a Verifier from determining whether a pair of attribute Verifier tokens belong to the same Prover, while untraceability ensures an issuer cannot trace an issued attribute to a Prover. Issuers participate solely

in providing attribute secret keys to Provers and not in the Attribute Verification process, preventing them from learning about the future use of issued attributes.

To conclude, while centralized Attribute Verification models provide enhanced computational capacity and lower complexity, the centralized entity may become vulnerable to single points of failure or similar attacks. The paper introduced a decentralized Attribute Verifier offering two verification modes, capable of verifying attributes in a decentralized and Zero-Knowledge way.

3 Regulatory Compliance Verification Protocol

This chapter will delve into the conceptual mathematical design of the Attribute Verification Protocol with the addition of a ticket generation feature. Additionally, a practical use case scenario concerning banking systems will be provided.

3.1 Adaptation of the Attribute Verification Protocol

In order to address in-deep the challenge of developing a privacy-preserving Attribute Verifier for assessing cloud regulatory compliance, a mathematical analysis of the Attribute Verification Protocol [4] needed to be performed first, in order to tailor this broad and versatile protocol to fit specific needs. In comparison to the already existing solutions, i.e. previously proposed model seen in Section 2.5, the adaptation of it for this Thesis integrates a ticketing system, that managed to fit perfectly in a decentralized logic. In particular, any trusted node can join the system and be an Issuer, issuing tickets for the attributes that are held by the Prover. Moreover, as it will be seen in section 3.3, the enhancement concerned also the ability to deploy the Proposed Protocol and all its components (i.e. all the actors involved) in a cloud environment, testing the whole workflow that compposes it. Here is the outcome of the study, i.e. the Attribute Verification Protocol with the previously mentioned modifications (See Figure 3.1), that can be adapted versatilely to many use cases.

1. Global Setup (performed once during the system setup): The Attribute Authority will perform this step just once, by getting the security parameter λ as input, and performing the global setup algorithm with which it will generate the global parameters, which include:

- Two cyclic groups G, G_T
- A generator g in G
- A bilinear mapping $e : G \times G \rightarrow G_T$
- A hash function $H : \{0,1\}^* \rightarrow \{0,1\}^d$

2. Attribute Setup (performed by the Issuer(s)): An Issuer joins the system, and takes the global parameters as input, in order to produce its private key pair $\alpha_i, \beta_i \in Z_p$, which will be kept private. Then it will compute its public key pair as follows:

$$e(g,g)^{\alpha_i}, g^{\beta_i}$$

3. Tickets Generation (performed by the Issuer(s)): A Prover with identity I_u and a set of attributes, requesting a ticket of compliance sk for each attribute i in its set, contacts the relevant Issuers. The relevant Issuers, in turn, generate the user's corresponding tickets, i.e. the secret keys related to those attributes, leveraging the equation below. For each attribute i:

$$sk(i,u) = g^{\alpha_i} H(I_u)^{\beta_i}$$

A ticket related to a certain attribute will be nothing else than a proof of possession of that attribute. The integrated ticketing system inside the Attribute Verification Protocol that was implemented in this Thesis, shapes the role of a secret key issued for a certain attribute, to take the value of proof of compliance of a certain aspect of a regulatory.
The decentralized nature of the protocol will make an Issuer node be responsible for the issuance of a certain number of tickets, while other Issuers are responsible for others, distributing the power or authority across multiple nodes and decreasing significantly the issues revolving around the centralized scenarios.

4. Encryption (performed by the Verifier):

Before performing the encryption step, the Verifier needs to perform some setup steps: first of all, it defines the set of target attributes $T_v = \{t_0, t_1, \ldots, t_n\}$ for verification. Subsequently, it also randomly generates a challenge key $R \in G_T$, and prepares a challenge ciphertext for the Prover, by first hashing the challenge key R that will be used as the key to the symmetrically encrypted challenge, and right after that, encrypting the challenge by using the key $k = H(R)$, which includes a nonce $r \in Z_p$, the timestamp ts, and the public key of the Verifier PK_v to be used later to secure the returned response, where $\|$ defines the concatenation:

$$\text{challenge} = \text{Enc}_{H(R)}(r \| ts \| PK_v)$$

The verifier then generates a random number $s \in Z_p$, and converts the access policy Γ to the equivalent linear secret sharing scheme (LSSS) matrix $M(\Gamma)$ for encryption. The access policy can be set to either require from the Prover the possession of just one of the attributes in T_v, which represents an n-out-of-n policy (boolean operator OR), or require the possession of all the attributes in T_v (boolean operator AND), which corresponds to an n-out-of-n policy [4]. The proposed model by this Thesis enforces an n-out-on-n policy for the covered use case, where the Prover, in order to be considered GDPR Article 32 [20] compliant by the Verifier, would need to prove the possession of four out of four required attributes.

The Verifier then gets from the Issuers the public keys, pair of $e(g,g)^{\alpha_i}$ and g^{β_i}. These public keys will be based on the target attributes in T_v. Up to this point, based on the number of columns in the $LSSS$ matrix, two vectors γ and ω are generated, where their first elements are set to s and 0, respectively, and the remaining elements are randomly chosen from Z_p. The randomly generated challenge key R will be encrypted using DABE [1].

In order to perform the DABE encryption algorithm, the Verifier generates three parameters r_i, γ_i, and ω_i, based on the number of rows in the $LSSS$ matrix $M(\Gamma)$, and for each of the attributes in T_v. r_i, taken as a parameter from the algorithm, is a random value that is chosen from Z_p, and γ_i and ω_i are computed using the equation below, where $M(\Gamma)_i$ indicates the ith row in $M(\Gamma)$.

$$\gamma_i = M(\Gamma)_i\gamma, \qquad \omega_i = M(\Gamma)_i\omega$$

The challenge key R will be then encrypted using the following formula:

$$C_0 = Re(g,g)^s$$

In the DABE encryption algorithm, three components C_{i1}, C_{i2} and C_{i3} are computed, and this is performed for each attribute i in T_v, using the equations below:

$$C_{i1} = e(g,g)^{\gamma_i}e(g,g)^{\alpha_i r_i}, \quad C_{i2} = g^{r_i}, \quad C_{i3} = g^{\beta_i r_i}g^{\omega_i}$$

The resulting ciphertext contains the single parameter C_0, which includes the actual encrypted message, and a number of parameters (C_{i1}, C_{i2}, C_{i3}) for each used attribute i in the defined access policy Γ. These components will be used by the Prover to attempt to get the component C_0, which represents nothing else than the encrypted challenge key R.

5. Decryption (performed by the Prover):

The Prover can prove the claimed attributes if the defined Γ returns true In the n-out-of-n mode, which is the one applied to the use case developed in this Thesis, the Prover needs all the secret keys, for each attribute in T_v, in order to get R. While in the 1-out-of-n mode, having any defined attributes in T_v already satisfies the Boolean formula [4]. In order to decrypt C_0, the Prover computes an intermediate value for attribute i using its secret key $sk(i,u)$ and parameter $C_i = (C_{i1}, C_{i2}, C_{i3})$.

Intermediate values are computed as follows:

$$\frac{C_{i1} \cdot e(H(I_u), C_{i3})}{e(sk(i,u), C_{i2})} = e(g,g)^{\gamma_i} e(H(I_u), g)^{\omega_i}$$

In the n-out-of-n mode, all computed intermediate values will be used to compute $e(g,g)^s$, while in the 1-out-of-n mode, just one single computed intermediate value.

The final Decryption is performed as follows:

$$e(g,g)^s = \begin{cases} e(g,g)^{\gamma_i} & \text{1-out-of-n mode} \\ \prod_{i=1}^{|\Gamma|} e(g,g)^{\gamma_i} e(H(I_u), g)^{\omega_i} & \text{n-out-of-n mode} \end{cases}$$

The challenge key R is recovered from C_0 as $R = \frac{C_0}{e(g,g)^s}$. By recovering R, the prover can decrypt the challenge and send back the response encrypted with the verifier's public key PK_v.

It's important to emphasize the potential of the proposed Attribute Verification Protocol that, by definition, takes advantage of the underlying DABE protocol in order to naturally support multiple nodes joining the system as Issuers. Section 3.3 of this work covers the implementation of the proposed protocol, and subsequent deployment in a pure Cloud scenario, giving the Cloud provider the role of an Issuer node devoted to issuing tickets of compliance.

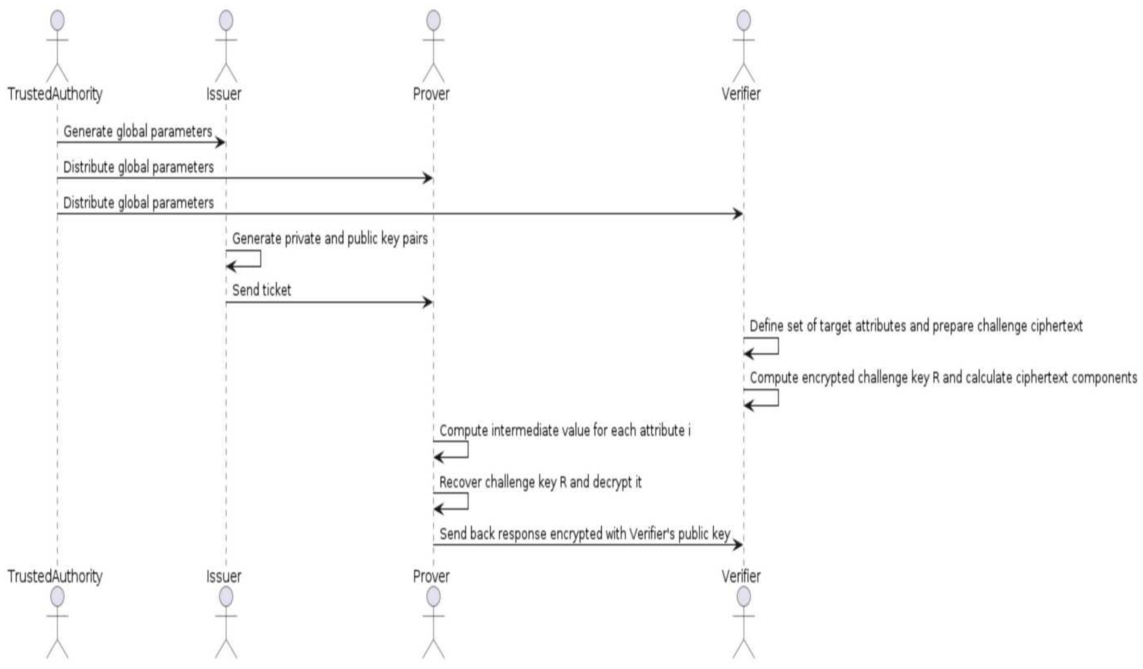

TrustedAuthority
Issuer
Prover
Verifier
Generate global parameters
Distribute global parameters
Distribute global parameters
Generate private and public key pairs
Send ticket
Define set of target attributes and prepare challenge ciphertext
Compute encrypted challenge key R and calculate ciphertext components
Compute intermediate value for each attribute i
Recover challenge key R and decrypt it
Send back response encrypted with Verifier's public key
TrustedAuthority
Issuer
Prover
Verifier

Figure 3.1: Sequence diagram of the proposed model

3.2 Proposed Protocol in Banking System

The use case explored for the proposed protocol, as already mentioned briefly in the previous sections, is the following: a National Bank requesting a certain resource or service from a higher European Institution like the European Central Bank (which will act as the Verifier or Auditor), and in order to access that resource/service, the prerequisite or necessary condition to be met, is being compliant with Article 32 of the GDPR (See Figure 3.2). For the proposed use case, the considered Prover was a National bank, in order to give consistency to the whole scenario, but it could have been any other institution that needs to pass a Verification check on GDPR requirements, performed by the Verifier. Out of the scope of this use case, is establishing whether the Prover belongs to the European Union, so a scenario where GDPR compliance needs to be just confirmed, because it's supposed to be already in place, but often there are rare cases of full compliance, or outside of the

EU, which would mean assessing from scratch. The proposed model could adapt to both cases, as it performs an in-deep check of all the necessary attributes related to Article 32.

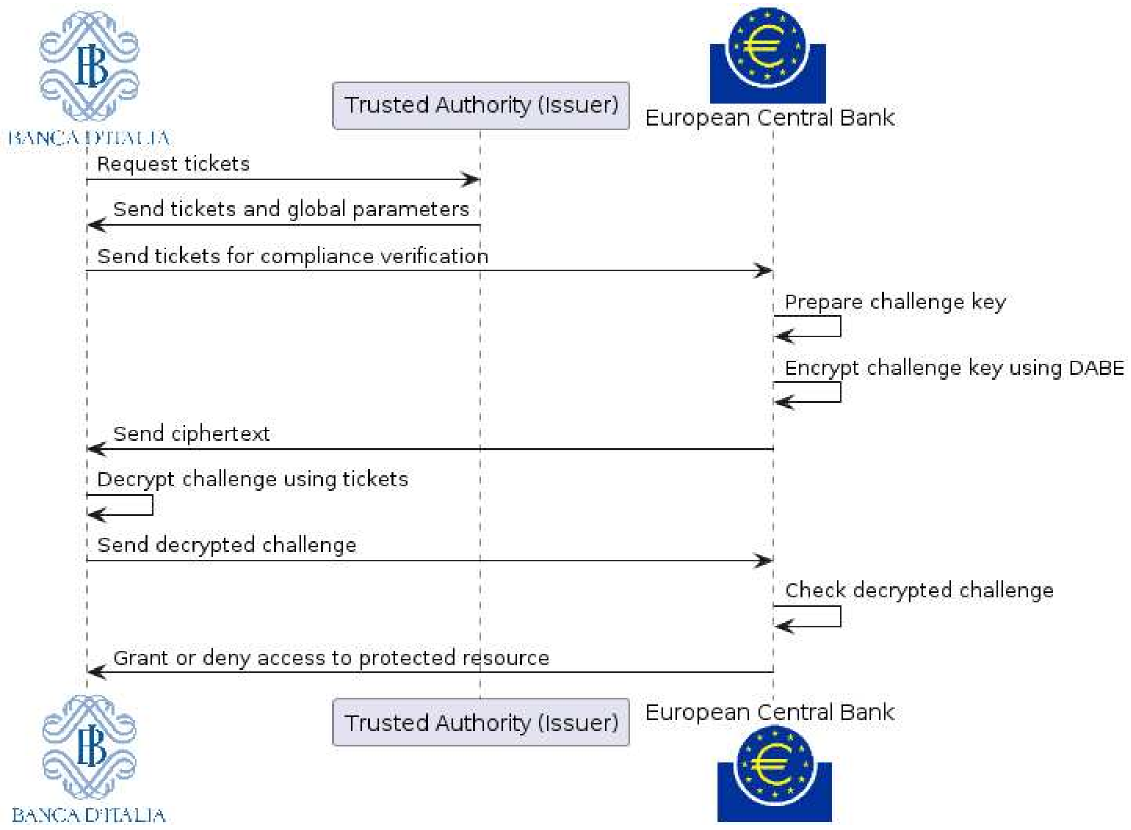

BANCA D'ITALIA
Trusted Authority (Issuer)
European Central Bank
Request tickets
Send tickets and global parameters
Send tickets for compliance verification
Prepare challenge key
Encrypt challenge key using DABE
Send ciphertext
Decrypt challenge using tickets
Send decrypted challenge
Check decrypted challenge
Grant or deny access to protected resource
BANCA D'ITALIA
Trusted Authority (Issuer)
European Central Bank

Figure 3.2: Sequence diagram of the proposed model in Banking Systems

Getting back to the Regulatory, Article 32 of the GDPR concerns the security of processing personal data, and it has probably the highest relevance among all the others, in terms of pure cyber security, as it mandates organizations that process personal data to implement appropriate technical and organizational measures to ensure a level of security appropriate to the risk [20]. Scope of the thesis was to analyze the article, spot the specific security requirements that it enforces, to be transposed then into attributes (to be used for the Attribute Verification Protocol), and implement the related security controls, that were developed in AWS Cloud Environment, through the use of Lambda functions written in Python. The addressed requirements of Article 32, transposed then into attributes, are four:

- Pseudonymization and encryption of personal data (Attribute 1);
- Ensuring the ongoing confidentiality, integrity, availability, and resilience of processing systems and services (Attribute 2);
- The ability to restore the availability and access to personal data in a timely manner in the event of a physical or technical incident (Attribute 3);
- A process for regularly testing, assessing, and evaluating the effectiveness of technical and organizational measures for ensuring the security of the processing (Attribute 4);

Starting from these four requirements enforced by Article 32, a logic to assess compliance with them, seen as attributes, needed to be established, in order to assess whether a Prover (in the use case, a National Bank) fulfills each of them. Moreover, in order to be compliant with the whole article 32, the Prover would need to pass the check of all four of them (n-out-of-n access policy), so the necessary condition would be receiving four tickets (secret keys associated with each attribute). The developed logic to assess compliance of each of the four attributes, is the following:

1. Pseudonymization and encryption of personal data (Attribute 1):
    - Pseudonymization technique
    - Encryption algorithm
    - Key length

   To be compliant with Attribute 1:

    - The pseudonymization technique must be either “tokenization” or “masking.” (Non-compliant example: “reversible_hashing”);

   - The encryption algorithm must be one of the approved algorithms, e.g., “AES”, “RSA”, “ChaCha20.” (Non-compliant example: “DES”);
   - The encryption key length must be equal to or greater than a certain value, e.g., 128 bits for symmetric encryption or 2048 bits for asymmetric encryption. (Non-compliant example: 56 bits for DES).

2. Ensuring the ongoing confidentiality, integrity, availability, and resilience of processing systems and services (Attribute 2):

   - Security measures
   - Assessment frequency
   - Vulnerabilities count
   - Vulnerabilities severity

   To be compliant with Attribute 2:

   - The security measures must include certain mandatory items (e.g., a firewall). (Non-compliant example: using only an intrusion detection system without a firewall);
   - The frequency of security assessments should be at least quarterly. (Non-compliant example: “yearly”);
   - The number of vulnerabilities found in the most recent assessment must be below a certain threshold (e.g., 5). (Non-compliant example: 7 vulnerabilities);
   - The severity of vulnerabilities found should not exceed a certain level (e.g., “low” or “medium”). (Non-compliant example: “high”).

3. The ability to restore the availability and access to personal data in a timely manner in the event of a physical or technical incident (Attribute 3):

- Backup type
- Backup frequency
- Recovery Time Objective (RTO) in hours
- Recovery Point Objective (RPO) in hours

To be compliant with Attribute 3:

- The backup type must be either “incremental” or “full.” (Non-compliant example: “differential”);
- The backup frequency must be either “daily” or “weekly.” (Non-compliant example: “monthly”);
- The RTO should be less than or equal to a maximum value, e.g., 8 hours. (Non-compliant example: 24 hours);
- The RPO should be less than or equal to a maximum value, e.g., 4 hours. (Non-compliant example: 12 hours).

4. A process for regularly testing, assessing, and evaluating the effectiveness of technical and organizational measures for ensuring the security of the processing (Attribute 4):

   - Assessment frequency
   - Assessment type
   - Assessment status
   - Last assessment date

   To be compliant with Attribute 4:

   - The assessment frequency should be one of the valid options, e.g., “monthly”, “quarterly”, “yearly.” (Non-compliant example: “every two years”);

- The assessment type must be one of the valid types, e.g., “vulnerability scanning”, “penetration testing”, “security audit.” (Non-compliant example: “informal review”)
- The assessment status must be “passed.” (Non-compliant example: “failed”);
- The time elapsed since the last assessment date must be less than or equal to a maximum value, e.g., 18 months. (Non-compliant example: 24 months since the last assessment).

In order to test both compliance and not compliance of a user, in regards to Attribute 32 of the GDPR, two different records, with different user_id (used as the partition key), were created in the DynamoDB database. So the table from which the data is being retrieved contains a complaint user and a not compliant one. The not-compliant user has the "vulnerability count" column equal to 6, when the threshold is 5. This would already make it not suitable to receive the ticket to Attribute 2 ("Ensuring the ongoing confidentiality, integrity, availability, and resilience of processing systems and service"), and consequently already not compliant overall with Article 32, as the Verifier requires an n-out-of-n policy over 4 attributes.

4 Implementation and Analysis

This chapter states the implementation and full deployment of the proposed protocol [21] in a cloud setting, i.e. AWS Cloud Environment of the practical use case scenario concerning banking systems. Additionally, the evaluation of the outcomes produced by the implemented solution will be addressed, and a comparison with existing solutions will be provided. Towards the end, the implication of the study will be highlighted.

4.1 Implementation of the Proposed Protocol

4.1.1 Implementation of the Issuer:

1. Definition of the DecentralizedABE class (see Code 4.1):

A DecentralizedABE class was defined, and tailored for the actor (in this case, the Issuer), with just the relevant methods for covering the logic of it, and the same was done for the other two actors involved (Prover and Verifier). An instance of this class was created in order to be able to invoke the following functions:

- `__init__(self, groupObj)`: Constructor of the DecentralizedABE class. Takes a pairing group object as a parameter and initializes the base class (ABEnc-MultiAuth). It also sets the global variables `util` and `group`:
  - `util`: SecretUtil object for working with secret sharing

  - `group`: pairing group object for cryptographic operations

- `setup(self)`: For the scope of this use case, the power of being an Attribute Authority generating the global parameters (GP) for the DABE scheme, was given to the cloud environment itself, i.e. the Issuer. This setup would take place just once, as soon as the system needs to be set up, and the Attribute Authority creates a random group element `g` in G1, and a hash function `H` that maps inputs to G1. The GP is a dictionary containing 'g' and 'H'.

- `issuersetup(self, GP, attributes)`: Sets up the issuer's secret key (SK) and public key (PK) for each attribute. Takes the global parameters GP and a list of attributes as input. It initializes empty dictionaries for SK and PK. For each attribute, it generates random `alpha_i` and `beta_i` values, computes `e(g,g)^alpha_i` and `g^beta_i`, and stores them in the respective SK and PK dictionaries. Returns a tuple containing SK and PK.

- `keygen(self, GP, SK, i, gid)`: Generates a key for a specific user (GID) and attribute (i). Takes the global parameters GP, secret key SK, attribute i, and user GID as input. It computes the hash of the GID using the H function, and then generates the key k using the formula: `k = (g^alpha_i) * (h^beta_i)`, where h is the hashed GID. Returns a dictionary containing the key k and the GID.

Listing 4.1: DecentralizedABE class tailored for the Issuer

```
class DecentralizedABE(ABEncMultiAuth):
    def __init__(self, groupObj):
        ABEncMultiAuth.__init__(self)
        global util, group
        util = SecretUtil(groupObj, verbose=False)
        group = groupObj

    def setup(self):
        # Generate the global parameters (GP)
        g = group.random(G1)
        H = lambda x: group.hash(x, G1)
        GP = {'g': g, 'H': H}
        return GP

    def issuersetup(self, GP, attributes):
        SK = {} # dictionary of {attribute i: {alpha_i, beta_i}}
        PK = {} # dictionary of {attribute i: {e(g,g)^alpha_i, g^
           beta_i}}
        for i in attributes: # done for each attribute that this AA
             handles
            alpha_i, beta_i = group.random(), group.random()
            # first part of the PK
            e_gg_alpha_i = pair(GP['g'], GP['g']) ** alpha_i
            g_beta_i = GP['g'] ** beta_i  # second part of the PK

            # The random group elements are the SK
            SK[i.upper()] = {'alpha_i': alpha_i, 'beta_i': beta_i}
            PK[i.upper()] = {'e(gg)^alpha_i': e_gg_alpha_i, 'g^
               beta_i': g_beta_i}

        return (SK, PK)

    def keygen(self, GP, SK, i, gid):
        #Generate a key for a specific user (GID) and attribute (i)
        h = GP['H'](gid)
        g = GP['g']
        k = (g ** SK[i.upper()]['alpha_i']) * (h ** SK[i.upper()]['
           beta_i'])
        return {'k': k, 'gid': gid}
```

2. Evaluating fulfillment of attributes related to GDPR Article 32:

Four functions were developed, so one for each attribute, in order to evaluate the fulfillment of the Prover to it, i.e. the possession of that particular attribute:

- "Pseudonymization and encryption of personal data (Attribute 1);"
- "Ensuring the ongoing confidentiality, integrity, availability, and resilience of processing systems and services (Attribute 2);"
- "The ability to restore the availability and access to personal data in a timely manner in the event of a physical or technical incident (Attribute 3);"
- "A process for regularly testing, assessing, and evaluating the effectiveness of technical and organizational measures for ensuring the security of the processing (Attribute 4)."

These functions return a boolean value indicating whether the National Bank, or any other Prover in any other field, meets that attribute or not.

1. `evaluate_attribute_1(attributes):`

This function evaluates the compliance of the organization based on "The pseudonymization and encryption of personal data" criteria. It performs the following checks:

- Pseudonymization Technique Check: It checks if the provided pseudonymization technique is among the valid techniques, i.e., 'tokenization' or 'masking'. Hence, if the Prover is using reversible hashing, then it would already be not compliant (as it is not in the valid techniques that were set), because it doesn't even fulfill Attribute 1.
- Encryption Algorithm Check: It checks if the provided encryption algorithm is among the valid algorithms, i.e., 'AES', 'RSA', or 'ChaCha20' (so, using DES would already make the user not compliant).
- Key Length Check: It checks if the provided key length is equal to or greater than the minimum required key length for the specified encryption algorithm: 128 bits for AES, 256 bits for ChaCha20, 2048 bits for RSA.

If all these checks are passed, the function returns True, indicating that the Prover fulfills Attribute 1, i.e. it is compliant with this attribute of Article 32. Otherwise, it concludes already that the user is not compliant.

2. `evaluate_attribute_2(attributes):`

This function evaluates the compliance of the Prover based on "Ensuring the ongoing confidentiality, integrity, availability, and resilience of processing systems and services" criteria. It performs the following checks:

- Security Measures Check: It checks if the organization has implemented all the mandatory security measures, such as a firewall. A not-compliant example would be having an IDS without a firewall,
- Assessment Frequency Check: It checks if the organization conducts assessments at valid frequencies, i.e., 'monthly' or 'quarterly'.
- Vulnerabilities Count Check: It checks if the number of vulnerabilities in the organization's system is less than or equal to the maximum allowed vulnerability count (5 in this use case).
- Vulnerabilities Severity Check: It checks if the severity of the vulnerabilities is within the acceptable range, i.e., 'low' or 'medium'.

If all these checks are passed, the function returns True, indicating that the Prover fulfills Attribute 2, i.e. it is compliant with this aspect of Article 32.

3. `evaluate_attribute_3(attributes):`

This function evaluates the compliance of the organization based on "The ability to restore the availability and access to personal data in a timely manner in the event of a physical or technical incident" criteria. It performs the following checks:

- Backup Type Check: It checks if the provided backup type is among the valid backup types, i.e., 'incremental' or 'full' (not compliant example: differential).
- Backup Frequency Check: It checks if the provided backup frequency is among the valid backup frequencies, i.e., 'daily' or 'weekly'.
- Recovery Time Objective (RTO) Check: It checks if the organization's RTO, in hours, is less than or equal to the maximum allowed RTO (8 in this case).
- Recovery Point Objective (RPO) Check: It checks if the organization's RPO, in hours, is less than or equal to the maximum allowed RPO (4 in this case).

If all these checks are passed, the function returns True, indicating that the Prover fulfills attribute 3, i.e. it is compliant with this aspect of Article 32.

4. `evaluate_attribute_4(attributes):`

This function evaluates the compliance of the organization based on "A process for regularly testing, assessing, and evaluating the effectiveness of technical and organizational measures for ensuring the security of the processing" criteria. It performs the following checks:

- Assessment Frequency Check: It checks if the organization's assessment frequency is among the valid frequencies, i.e., 'monthly', 'quarterly', or 'yearly'.
- Assessment Type Check: It checks if the organization's assessment type is among the valid assessment types, i.e., 'vulnerability scanning', 'penetration testing', or 'security audit' (not compliant example: informal review).
- Assessment Status Check: It checks if the organization's assessment status is 'passed'.
- Assessment Age Check: It checks if the time since the organization's last assessment is less than or equal to the maximum allowed assessment age (18 months in this case).

If all these checks are passed, the function returns True, indicating that the Prover fulfills attribute 4, i.e. it is compliant with this aspect of Article 32. Otherwise, it returns False.

These four functions mentioned above are part of the policy evaluation and compliance verification process.

3. lambda_handler function for the Issuer (portion 1):

The `lambda_handler(event, context)` for the Issuer extracts first the `user_id` from the event, and retrieves the corresponding user attributes from DynamoDB. If the user is found, a pairing group object `groupObj` is initialized, with the chosen security setting 'SS512'. This object is used to work with bilinear pairings. Next, a DecentralizedABE object, `dabe`, is instantiated using the `groupObj`. The function then calls the `dabe.setup()` method to generate the global parameters (`GP`) required for the Decentralized Attribute-Based Encryption scheme.

4. lambda_handler function for the Issuer (portion 2):

At this point, an `attribute_mapping` dictionary gets defined, to map numerical keys to descriptive attribute names (four attributes, as previously mentioned) that represent specific requirements of GDPR Article 32. Then,the Issuer's secret key (`SK`) and public key (`PK`) are generated for each attribute, using the `dabe.issuersetup()` method and the generated global parameters.

For each attribute, the function checks whether the user's attributes fulfill the corresponding GDPR requirement. If the Prover fulfills it, then a ticket is generated for that attribute using the `dabe.keygen()` method, which takes the global parameters, Issuer's secret key, attribute name, and user identity as input. The resulting ticket's 'k' element is then serialized and added to the `generated_tickets` dictionary.

Finally, the Issuer's public key is serialized into a dictionary, `serialized_public_key`, by iterating through each attribute and it's then returned as part of the response.

5. Json response from the Issuer:

The Issuer Lambda function's final part constructs a JSON response, which is then sent to both the Prover and the Verifier. The response body contains the generated `tickets`, `public_key`, and `global_parameters`. The `tickets` dictionary includes serialized secret keys for each compliant attribute, serving as evidence of the Prover's GDPR Article 32 compliance. The `public_key` field holds the serialized public keys for each attribute. Lastly, the `global_parameters` contain the serialized generator `g` of the pairing group, which is required, for both the Prover and the Verifier, to perform DecentralizedABE operations.

Together, these components enable the Prover to perform the necessary cryptographic operations and provide the Verifier with the required information to determine the Prover's compliance (see Figure 4.1). Based on this information, the Verifier can then decide whether to grant or deny access to the protected resource or service.

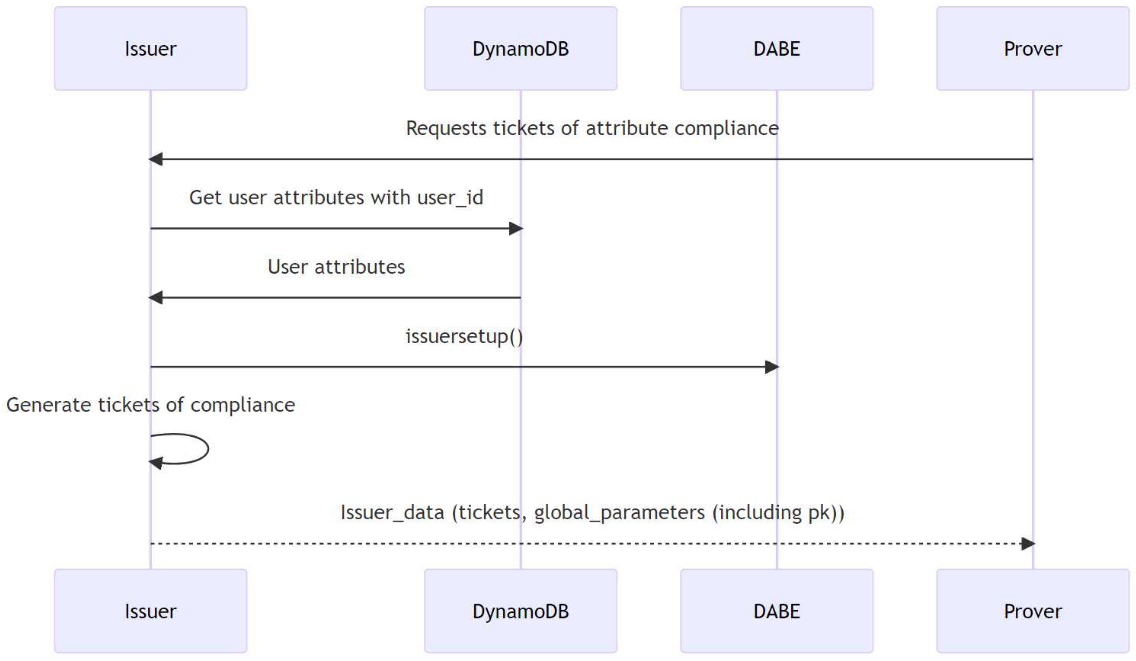

Issuer
DynamoDB
DABE
Prover
Requests tickets of attribute compliance
Get user attributes with user_id
User attributes
issuersetup()
Generate tickets of compliance
Issuer_data (tickets, global_parameters (including pk))
Issuer
DynamoDB
DABE
Prover

Figure 4.1: Sequence diagram of the Issuer's Lambda function

4.1.2 Implementation of the Prover:

1. Definition of the DecentralizedABE class (see Code 4.2):

The `DecentralizedABE` class is an extension of the `ABEncMultiAuth` class that has been tailored to the specific requirements of the Prover's Lambda function. It provides the necessary methods for handling decryption in a decentralized attribute-based encryption (DABE) scheme, allowing the Prover to demonstrate possession of required attributes without revealing their actual values.

`original_decrypt(self, GP, sk, ct)`: This method is a key component of the `DecentralizedABE` class, responsible for decrypting a challenge ciphertext (`ct`) using the user's secret keys (`sk`) for specific attributes. It takes the global parameters (`GP`), the user's secret keys (`sk`), and ciphertext (`ct`) as input. The method first creates a policy based on the ciphertext's policy and then prunes it based on the user's attributes. If the user does not possess the required attributes, the decryption process is terminated.

Next, the method computes coefficients for the pruned policy and calculates an intermediate value (`egg_s`) using a loop over the pruned policy's attributes. Within the loop, it computes the numerator (`num`) and denominator (`dem`) of the decryption formula and raises the result to the power of the corresponding policy coefficient. The intermediate value (`egg_s`) is updated by multiplying it by the current result.

Finally, the method returns the decrypted challenge by dividing the ciphertext's `C0` component by the intermediate value (`egg_s`). The decrypted challenge is then used by the Prover to demonstrate the possession of the required attributes, enabling the secure execution of the Prover Lambda function.

Listing 4.2: DecentralizedABE class tailored for the Prover

```
class DecentralizedABE(ABEncMultiAuth):
    def __init__(self, groupObj):
        ABEnc.__init__(self)
        global util, group
        util = SecretUtil(groupObj, verbose=False)
        group = groupObj

    # Original encrypt function
    def original_encrypt(self, GP, M, policy_str):
        policy = util.createPolicy(policy_str)
        p_list = util.getAttributeList(policy)
        s = group.random()
        print("M:", M, "type:", type(M))
        print("GP['g']:", GP['g'], "type:", type(GP['g']))
        print("s:", s, "type:", type(s))

        #C0 = M * (GP['g'] ** s)
        C0 = pair(M, GP['g'] ** s)

        C1, C2, C3 = {}, {}, {}
        for i in p_list:
            r = group.random()
            C1[i] = (GP['g'] ** r)
            C2[i] = (GP[i] ** r)
            C3[i] = (GP[i] ** s) * (GP['g'] ** -r)
        return { 'C0':C0, 'C1':C1, 'C2':C2, 'C3':C3, 'policy':
            policy_str }
```

2. lambda_handler function for the Prover (first portion)

The `lambda_handler(event, context)` for the Prover starts by invoking the Issuer with the `user_id` extracted from the event. It receives a response that includes the "user attribute secret keys" (i.e. the tickets) and the public key associated with them. The function proceeds to deserialize the pairing group object, the tickets, the public key (using the `PairingGroup` object), and the global parameters (`GP`).

3. lambda_handler function for the Prover (second portion):

In the second portion of the `lambda_handler` function for the Prover, the ciphertext (i.e., the encrypted challenge) is deserialized from the event. The function iterates through the ciphertext, and for each nested key, it deserializes the corresponding values using the `PairingGroup` object. After deserializing the ciphertext, the `DecentralizedABE` object is instantiated, and the challenge is decrypted using the `original_decrypt` method. This method takes the global parameters (`GP`), user attribute secret keys (tickets), and the deserialized ciphertext as input arguments. The result is the decrypted challenge (challenge key R), which is also deserialized using the `PairingGroup` object.

4. Json response from the Prover:

The final part of the Prover Lambda function prepares and returns the JSON response for the Verifier. The body of the response contains the `tickets` and the `decrypted_challenge`. The `tickets` are a dictionary of attribute names and their corresponding serialized secret keys, serving as evidence that the Prover is compliant with GDPR Article 32. The `decrypted_challenge`, obtained after decrypting the ciphertext, proves that the Prover possesses the necessary secret keys to meet the Verifier's access policy. Both elements together enable the Verifier to determine the Prover's compliance and decide whether to grant or deny access to the protected resource based on the provided information (see Figure 4.2).

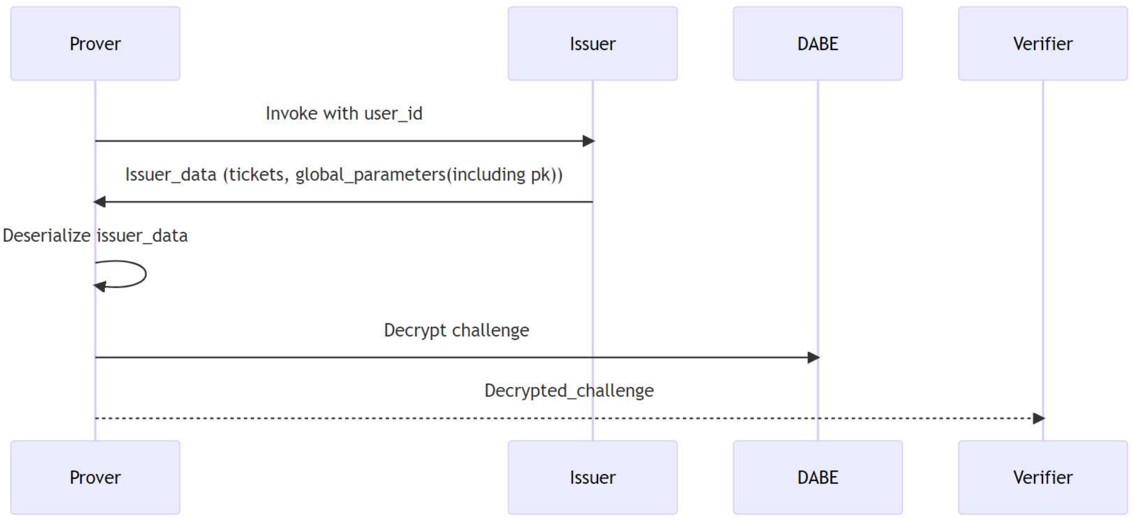


Figure 4.2: Sequence diagram of the Prover's Lambda function

4.1.3 Implementation of the Verifier:

1. Definition of the DecentralizedABE class (see Code 4.3):

The DecentralizedABE class is an extension of the ABEncMultiAuth class tailored for the Verifier Lambda function. It provides the necessary methods for handling encryption in a decentralized attribute-based encryption (DABE) scheme, allowing the Verifier to encrypt a challenge message based on the required access policy.

original_encrypt(self, GP, M, policy_str): this method is a central component of the DecentralizedABE class, responsible for encrypting a challenge message (M) based on the given access policy string (policy_str). The method first creates a policy based on the access policy string and obtains the attribute list from the policy. It then generates a random value (s) and initializes the ciphertext components (C0, C1, C2, and C3).

Next, the method calculates the C0 component of the ciphertext by pairing the challenge message with the global parameter g raised to the power of the random value s. It then iterates through the policy's attribute list and generates random values (r) for each attribute. The C1, C2, and C3 components of the ciphertext are updated with corresponding calculations for each attribute using the random values r and the global parameters.

Finally, the method returns a dictionary containing the ciphertext components and the access policy. The encrypted challenge is then used by the Verifier to securely execute the Verifier Lambda function, allowing only users with the required attributes to decrypt the challenge and obtain access.

Listing 4.3: DecentralizedABE class tailored for the Verifier

```python
class DecentralizedABE(ABEncMultiAuth):
    def __init__(self, groupObj):
        ABEnc.__init__(self)
        global util, group
        util = SecretUtil(groupObj, verbose=False)
        group = groupObj

    # Original encrypt function
```

```
def original_encrypt(self, GP, M, policy_str):
    policy = util.createPolicy(policy_str)
    p_list = util.getAttributeList(policy)
    s = group.random()
    print("M:", M, "type:", type(M))
    print("GP['g']:", GP['g'], "type:", type(GP['g']))
    print("s:", s, "type:", type(s))

    #C0 = M * (GP['g'] ** s)
    C0 = pair(M, GP['g'] ** s)

    C1, C2, C3 = {}, {}, {}
    for i in p_list:
        r = group.random()
        C1[i] = (GP['g'] ** r)
        C2[i] = (GP[i] ** r)
        C3[i] = (GP[i] ** s) * (GP['g'] ** -r)
    return { 'C0':C0, 'C1':C1, 'C2':C2, 'C3':C3, 'policy':
       policy_str }
```

2. lambda_handler function for the Verifier (first portion):

In the first portion of the Verifier's `lambda_handler(event, context)`, the function initializes the pairing group object and sets up a connection to the AWS Lambda service. It then invokes the Issuer's Lambda function by passing the necessary `user_id` and receiving the JSON response from the Issuer's Lambda function. The function proceeds to extract the response's body, which includes the serialized global parameters and public keys for the attributes. Next, the global parameters and public keys are deserialized, converting them back into their original formats.

To define the access policy, the Verifier uses a dictionary named `attribute_mapping`, which maps integers to attribute names. The access policy is created as a string representing the logical AND of all the attributes, so a n-out-of-n policy, indicating that all of them must be satisfied for the access to be granted. This policy will be used later in the process to encrypt the challenge key R.

3. lambda_handler function for the Verifier (second portion):

In the second portion of the `lambda_handler` function, the Verifier prepares the challenge key R by first creating an instance of the DecentralizedABE class and then generating a random element from the target group GT. The challenge key is then converted to a string, subsequently encoded into bytes, and a symmetric cipher is initialized using the SHA256 hash of the challenge key bytes.

The challenge key R is symmetrically encrypted as a challenge and subsequently encrypted again using the Attribute Verification Protocol from the Decentralized-ABE class. This double encryption generates a ciphertext containing the encrypted challenge key R, and this ciphertext is then serialized, and each of its elements is encoded using base64.

The Prover is invoked by passing the `user_id`, public keys, and the encrypted challenge (ciphertext). The Verifier, then, receives the JSON response from the Prover and extracts the decrypted challenge. And finally, it will be able to compare the decrypted challenge with the original challenge key R, to determine whether the Prover has the necessary attributes to satisfy the access policy or not. The result of this comparison is either "Access granted" or "Access denied" (see Figure 4.3), which will be included in the JSON response returned by the Verifier's Lambda function.

4. Json response from the Verifier:

In the Verifier's Lambda function, the JSON response's body contains three key-value pairs: `statusCode`, `ciphertext`, and `result`. The `statusCode` indicates the HTTP status code. The `ciphertext` is the serialized encrypted challenge key (R) generated by the Verifier, which is sent to the Prover for decryption. Lastly, the `result` field holds the outcome of the verification process, either "Access granted" or "Access denied," based on whether the Prover's decrypted challenge matches the original challenge key. The JSON response in the Verifier Lambda function, similar to those in the Issuer and Prover Lambda functions, facilitates communication between the involved parties by providing the necessary information to complete the protocol.

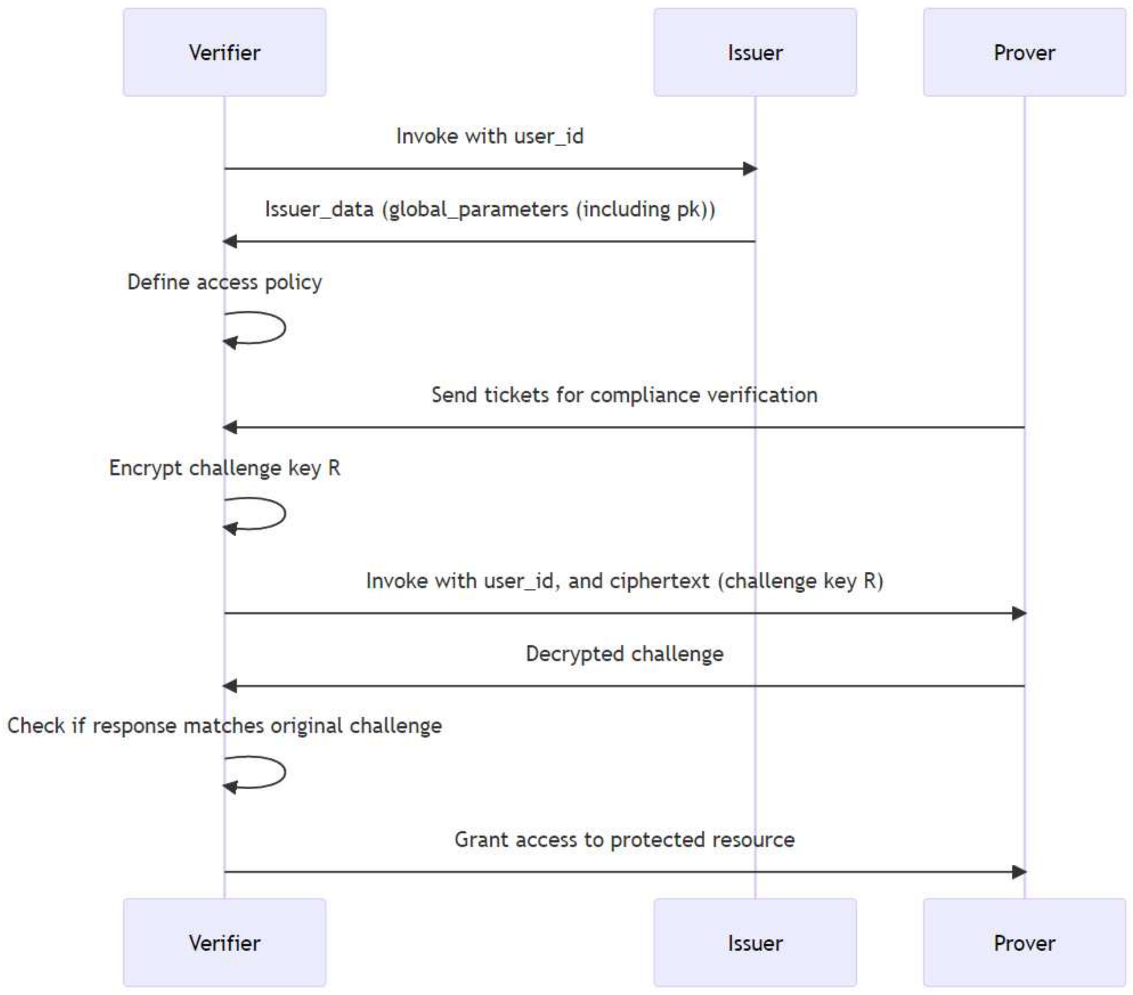

Verifier
Issuer
Prover
Invoke with user_id
Issuer_data (global_parameters (including pk))
Define access policy
Send tickets for compliance verification
Encrypt challenge key R
Invoke with user_id, and ciphertext (challenge key R)
Decrypted challenge
Check if response matches original challenge
Grant access to protected resource
Verifier
Issuer
Prover

Figure 4.3: Sequence diagram of the Verifier's Lambda function

4.1.4 Implementation of the Orchestrator:

The purpose of this Lambda function is to coordinate all the other three, i.e. the three actors involved that we have seen so far, and calling them in a chronological way according to the workflow of the Attribute Verification Protocol.

1. **lambda_handler (first portion) - Prover's request for ticket:**

In the first part of the `lambda_handler` function, the Prover requests a ticket from the Issuer by invoking the `issuer_thesis3` Lambda function. The `lambda_handler` sends the user's ID as input to the Issuer function using the `Payload` parameter in a

JSON format. After receiving the Issuer's response, it extracts the JSON response body from the Issuer's function using the `Payload` key and stores the result in the `issuer_response_body` variable. The Orchestrator then prints the response and further extracts the Issuer's data from the `issuer_response_body` by accessing the `body` key and loading it as a JSON object. This data includes the user's attribute secret keys and other relevant information necessary for the subsequent steps in the protocol.

2. lambda_handler (second portion) - Challenge key generation:

In this portion of the `lambda_handler` function, the Verifier checks if the user is compliant with GDPR Article 32 and a ticket has been issued. If the ticket is present, the Verifier proceeds to generate a challenge key. To do this, it invokes the `verifier_thesis` Lambda function with the `prepare_challenge` action and the user's ID as input. The Verifier's response is extracted as a JSON object and stored in the `verifier_response_body` variable. The ciphertext of the encrypted challenge key, which will be sent to the Prover later, is extracted from the response and stored in the `ciphertext` variable.

3. lambda_handler (third portion) - Prover sending the ticket for verification:

In this part of the `lambda_handler` function, the Prover sends the ticket, the public key, and the encrypted challenge key (ciphertext) to the Verifier for verification. To do this, the `prover_thesis` Lambda function is invoked with the user's ID, ticket, public key, and ciphertext as input in a JSON format. The Prover's response is extracted as a JSON object and stored in the `prover_response_body` variable. The Orchestrator then prints the Prover's response.

4. lambda_handler (fourth portion) - Attribute Verification step: In the final portion of the `lambda_handler` function, the Verifier checks the decrypted challenge to grant or deny access to the Prover (See Figure 4.4). The `verifier_thesis` Lambda function is invoked again, but with the `verify_challenge` action, the user's ID, the decrypted challenge from the Prover's response, and the ticket as input. The Verifier's response is extracted as a JSON object and stored in the `verifier_response_body` variable. The Orchestrator then prints the Verifier's response, which contains the result of the verification process.

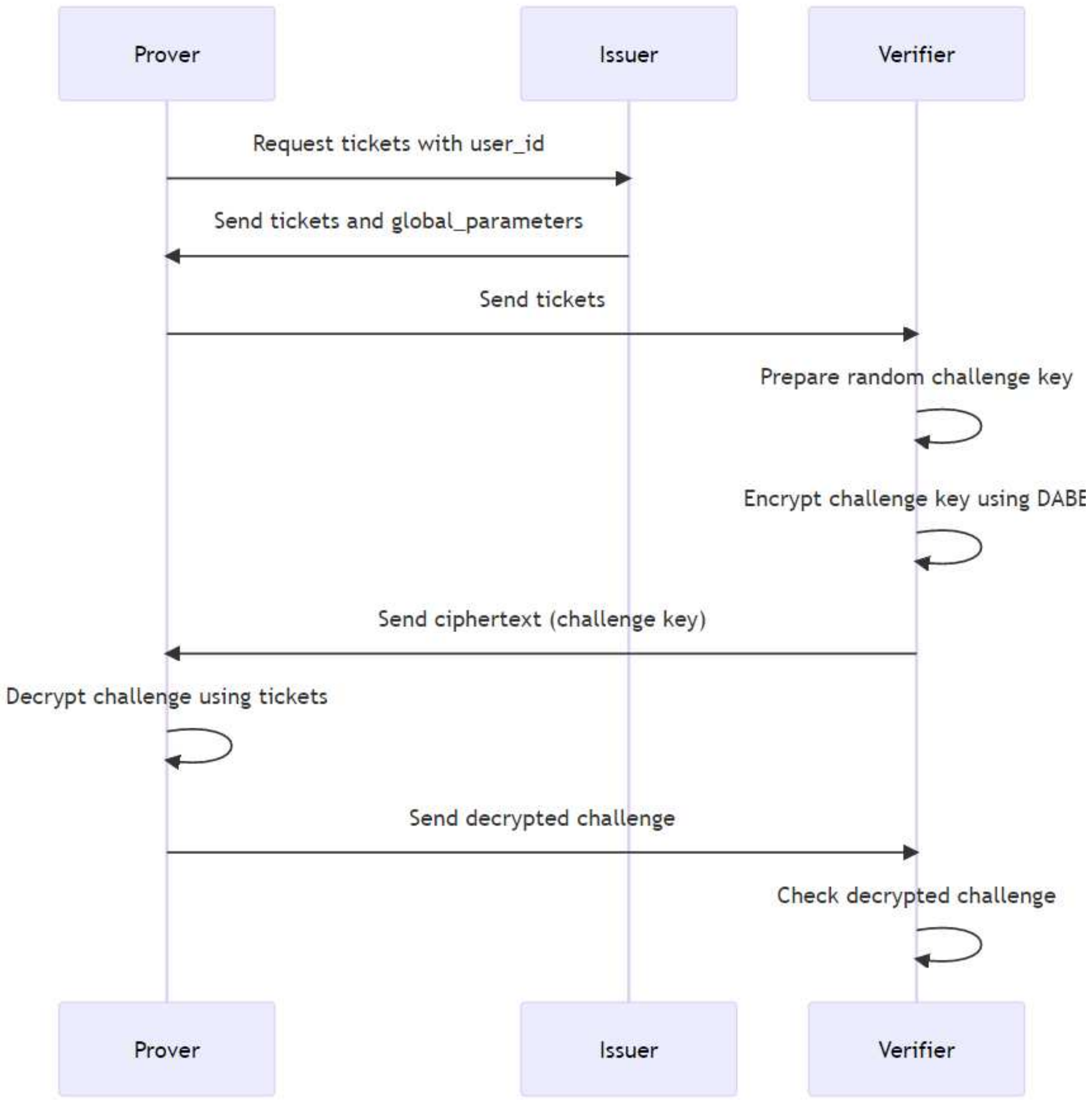

Prover
Issuer
Verifier
Request tickets with user_id
Send tickets and global_parameters
Send tickets
Prepare random challenge key
Encrypt challenge key using DABE
Send ciphertext (challenge key)
Decrypt challenge using tickets
Send decrypted challenge
Check decrypted challenge
Prover
Issuer
Verifier

Figure 4.4: Sequence diagram of the Orchestrator Lambda function

4.2 Evaluation of the Proposed System

Evaluating the proposed system involved successfully integrating the modifications to the existing Attribute Verification protocol, transposing the protocol into code, assessing the successful implementation of all the actors in the Cloud Environment, their automation, the automation of regulatory compliance verification, and the system's overall performance in achieving privacy-preserving regulatory compliance.

The most important thing to highlight, before start analyzing the results, is that for the implementation of this use case, the resource-related constraints in having just one Cloud provider, and one account for it, made natural testing the solution with just one supported Issuer, and they made this Issuer coincide also with the Attribute Authority responsible for the global parameter generation (Global Setup phase). However, the way in which the solution was successfully implemented, already guarantees smoothness in the support of multiple Issuers, as it is intended, in case of possession of the necessary resources to simulate them.

The ticketing system for regulatory compliance check was successfully integrated into the skeleton of the Attribute Verification Protocol, and so the transposition of it into code, by having a Lambda function written in Python for all the three actors, plus another one orchestrating chronologically their interactions and the whole workflow.

The Issuer has been successfully implemented, generating and distributing attribute-related tickets of proof in the correct format, and ensuring that each ticket is unique to the user. This demonstrates the feasibility of using the Attribute Verification Protocol for secure ticket issuance. The use case related to the attributes belonging to Article 32 of the GDPR was successfully tested, showing the fulfillment of two goals: translation of the attributes (populating the database) into code, and automation of the compliance check.

The Orchestrator, i.e. the Lambda function that has the duty of integrating and coordinating all three actors and simulating their workflow in the chronological, intended way, has also been successfully implemented,

However, some limitations were encountered along the way: the Prover and the Verifier's Lambda functions faced some package/module issues (that will need to be further investigated as future work), preventing them to reach the end of their life-cycle, i.e. the final output. However, their semantic design is correct, and the debug process for both of them shows that there's no issue with the way the encryption and decryption processes were implemented, and not even with the related, integrated, ticketing system (that already allows the Issuer to successfully issue tickets of compliance related to the Prover's attributes). Moreover, the correct design and implementation of the orchestrator function are promising for the overall system's functionality.

In terms of performance, the Lambda function of the Issuer was taken as a sample for assessing the overall efficiency of the artifact, being the most complex one and so the most computationally expensive.

In order to assess the performance of the considered Lambda function, there was first the need to understand the complexity of the code and the operations it performs. The complexity of the code was, in fact, estimated considering that the Issuer's Lambda function:

- Interacts with the database DynamoDB to retrieve user attributes
- Instantiates a pairing group object for bilinear pairings
- Instantiates a DecentralizedABE object using the pairing group object
- Sets up the global parameters for the DecentralizedABE object
- Generates the issuer's secret key and public key for each attribute
- Creates a user identity based on the `user_id`
- Evaluates compliance for each attribute using the defined rules
- Generates tickets for each compliant attribute

- Serializes the public key and the generated tickets, so performing the DABE Encryption step for each of the fulfilled attributes: n fulfilled attributes, n encryption steps performed.

- Returns the generated tickets, public key, and global parameters as a JSON object, the be captured in the event by Prover, Verifier, and Orchestrator

To perform the assessment, two metrics have been retrieved from CloudWatch: Duration Time, providing information about the execution time of the Lambda function during different invocations, and Success Rate, providing insights about the successful operation of the Lambda function within a given period of time. These metrics help understand the performance and efficiency of the Issuer's Lambda function, and they can be useful for identifying any potential bottlenecks or areas that require optimization. Analyzing the Duration time, the following results were shown (See Figure 4.5), showing the Maximum, Average, and Minimum Duration on an overall number of 7 invocations:

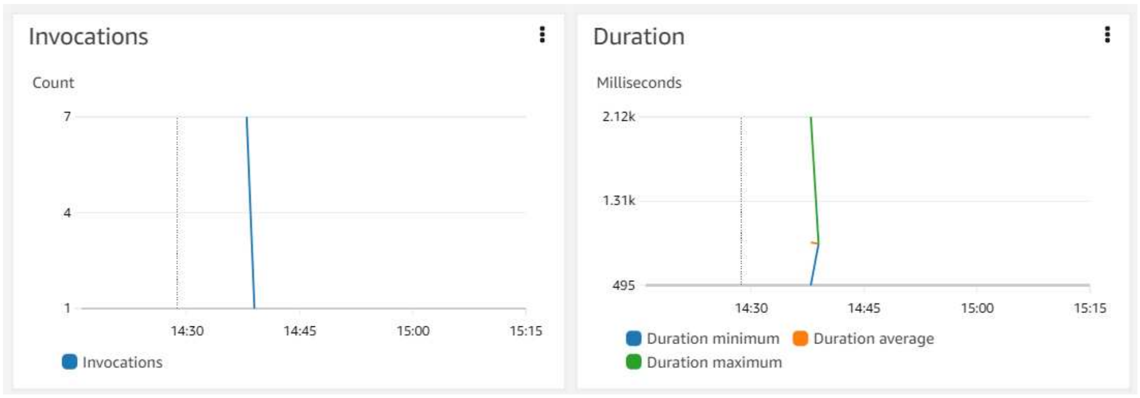
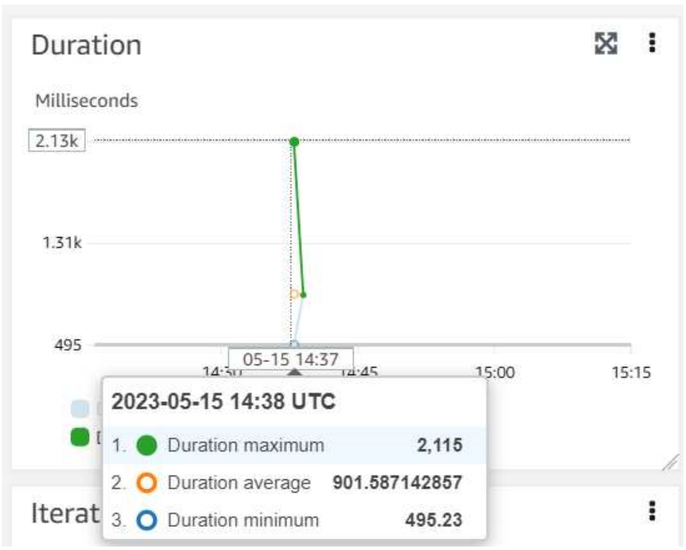


Figure 4.5: Duration Time and Number of Invocations

In order to interpret the result, the following definitions need to be taken into account:

- Duration maximum: This is the longest time (in milliseconds) that the Lambda function took to execute among all its invocations within the given period.
- Duration average: This is the average time (in milliseconds) that the Lambda function took to execute across all its invocations within the given period.
- Duration minimum: This is the shortest time (in milliseconds) that the Lambda function took to execute among all its invocations within the given period.

Considering the operations performed by the Lambda function, which involves also cryptographic operations that can be computationally expensive, the execution times (See Figure 4.6) seemed to be even better than the most optimistic expectations:



2023-05-15 14:38 UTC

1. ●	Duration maximum	2,115
2. ○	Duration average	901.587142857
3. ○	Duration minimum	495.23

Figure 4.6: Maximum, Average, and Minimum Duration

While Regulatory Compliance check contexts don't really require high-speed and optimal verification time in comparison to other scenarios like electronic payments, high-frequency trading (HFT), real-time gaming, and video streaming, the achievement in terms of performances that were reached, established the foundations for creating awareness about this possibility.

Finally, the second and last criterion that was taken into consideration for evaluating the efficiency, but mostly effectiveness of the implemented solution, is the Success Rate. In particular, the chart drawn by CloudWatch based on the logs, confirmed what has already been tested: the Issuer, no matter if the output was a compliant or not compliant record, always succeeded to reach the end of its lifecycle, resulting in a Success Rate of 100% (See Figure 4.7).

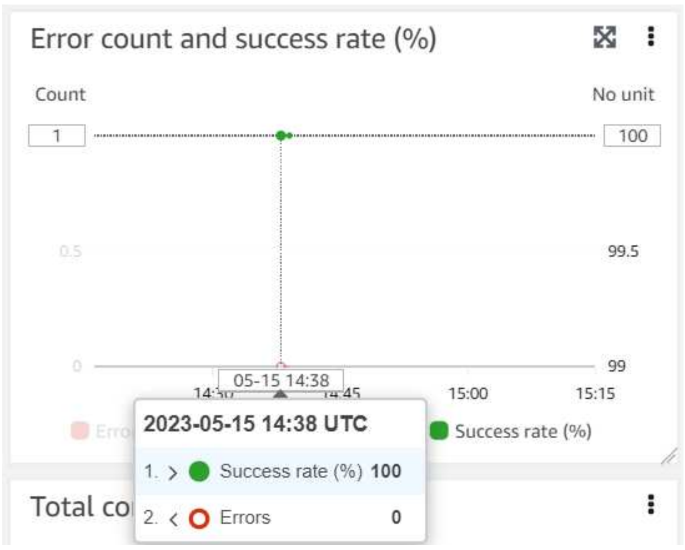


Figure 4.7: Success Rate of the Issuer's Lambda function

4.2.1 Comparison with Existing Solutions

A significant portion of the research objectives behind this work, was taken and extended from a previous Master's Thesis [22]. In particular, the just mentioned Thesis, highlighted the issue caused by the need for manual effort in checking regulatory compliance, and developed an artifact to automate some security objectives taken from several regulations, into code. One of the goals of my Master's Thesis was taking the same concept of dealing with regulations through code, but adding a privacy layer to the proof of compliance check, thus following a privacy-by-design approach. The added value that my work achieved, is indeed having integrated an

Attribute Verifier into the compliance check, ensuring confidentiality by enhancing privacy during the compliance verification process.

An additional existing solution, which represented the biggest inspiration behind my work, was the already mentioned paper that addressed the development of an Attribute Verifier for the Internet of Things [4]. The paper delved into the Attribute Verification Protocol and the related underlying encryption scheme, with an emphasis on the decentralized nature allowed by DABE. My thesis took the proposed Attribute Verification Protocol as a layout and integrated in it a ticketing system for proving regulatory compliance. Another addition that was achieved, was the implementation of the proposed protocol, through code, in a full Cloud Environment setting, demonstrating its feasibility and effectiveness in enhancing privacy.

4.2.2 Implications of the Study

The proposed Attribute Verification Protocol provides a novel solution for privacy-preserving regulatory compliance in Cloud Environment. The design and implementation of the suggested protocol aim to enhance the privacy of sensitive data stored in the cloud, while ensuring adherence to regulatory compliance. Even though the covered use case was developed around GDPR and its Article 32, the work has shown the versatility of the proposed solution, which will be able to be adopted for other regulations and multiple security objectives. In addition to that, the decentralized nature of the protocol, capable of supporting multiple Issuers, will reduce the power held by a single Issuer and will contribute to eliminating single points of failure issues, and related risks linked to centralized settings.

The outcome of this study will have implications in the field of privacy, in particular, privacy-preserving technologies, as the whole solution was motivated by a strong and targeted privacy-by-design overall goal. The implications will be substantial also for the field of Cloud Regulatory, and Data Protection Regulations, as it was demonstrated a possible way to automate compliance checks, without mining the confidentiality of the assets belonging to the entity involved in the audit process. Finally, some implications will concern also the pure Cloud Computing field. That's because the deployment of a heavy cryptographic solution and the implementation of the whole solution, i.e. all the actors involved, was implemented fully in the AWS Cloud Environment, without needing local machine solutions or other applications different from the cloud.

5 Conclusion and Future Work

In this chapter the conclusions related to the achievements reached during the pursued work, and related limitations, will be discussed. Towards the end of it, future improvements and research directions will be pointed out.

5.1 Summary and Conclusions

This Master's Thesis aimed to develop a privacy-preserving Attribute Verifier for regulatory compliance in a Cloud Environment, that would enhance automation for compliance checks, and privacy for the verification of the requirements verification. The proposed system successfully integrated the Attribute Verification Protocol with a ticketing system for regulatory compliance checks. The use case of GDPR Article 32 was implemented and tested, showcasing the feasibility of automating compliance checks while preserving privacy. Moreover, the study demonstrated the protocol's potential to be extended to other regulations and security objectives.

The evaluation of the proposed system revealed a successful implementation through code of the Issuer and Orchestrator, while highlighting some limitations with the Prover and Verifier's Lambda functions. Despite these issues, the overall design and functionality of the system show promise in achieving the intended privacy-preserving regulatory compliance.

A comparison with existing solutions indicated that the proposed Attribute Verification Protocol expands on the previous work [22], adding a privacy layer to the compliance check. It also built upon the IoT-oriented Attribute Verifier presented in [4], which was translated into code, and fully deployed in Cloud Environment.

The study has implications for privacy, Cloud Regulatory, and Data Protection Regulations, as well as the field of Cloud Computing. By offering a privacy-preserving solution for regulatory compliance, it contributes to the advancement of privacy-enhancing technologies, automation of compliance checks, and cloud-based deployment of cryptographic solutions.

In conclusion, this Master's thesis demonstrated the feasibility and effectiveness of a privacy-preserving Attribute Verifier for regulatory compliance in a Cloud Environment, achieved by integrating a ticketing system for attribute compliance checks. Finally, the proposed model provides a novel solution that enhances both privacy and automation in regulatory compliance verification.

5.2 Future Research Directions

The findings and lessons learned along with the development of the artifact (the overall privacy-preserving Attribute Verifier) will not stop to the state of the art of this Master's Thesis, but broader and more granular contexts will be considered. Here are some future improvements, and improvements that are already in place, but were not leveraged, for non-compatibility with the considered use case:

- In terms of Regulatory: the Master's Thesis considered just the GDPR, but the same artifact can be tailored to NIST, SOC2, ISO27001, HIPAA, or in terms of data protection, like in the case of GDPR, also regulations like CCPA (California Consumer Privacy Act);
- In terms of attribute verification, the Issuer can be tailored to a specific security control or article, like in the considered use case, with Article 32, or issuing tickets for multiple articles (like in the case of Data Protection Regulations) or

security objectives (like in the case of standards, e.g. NIST). With the latter, the Prover will be able to request proof of compliance (to send to the Attribute Verifier) for multiple articles/security objectives, with just a single interaction with the Issuer;

- In terms of the underlying encryption scheme and its applicability, the Attribute Verification Protocol relies on DABE, for supporting multiple Issuers in cloud environment (in this case: multiple cloud providers). This will make the DABE protocol increase even more the security of this privacy-preserving solution, as it will allow the Verification of attributes even in case of participation of multiple Issuers (decentralized scenarios), and this will overcome potential single point of failure issues, and targeted attack scenarios.

References

[1] A. Lewko and B. Waters, "Decentralizing Attribute-Based Encryption", in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, Tallinn, Estonia, 2011, pp. 568–588.

[2] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge Complexity of Interactive Proof-Systems", *Journal of the ACM (JACM)*, vol. 35, no. 1, pp. 67–88, 1988.

[3] A. Sahai and B. Waters, "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data", in *Proceedings of the 2005 ACM Conference on Computer and Communications Security*, ACM, Alexandria, Virginia, USA, 2005, pp. 89–98.

[4] M. B. M. Kamel, Y. Yan, P. Ligeti, and C. Reich, "Attribute Verifier for Internet of Things", in *2022 32nd International Telecommunication Networks and Applications Conference (ITNAC)*, Wellington, New Zealand, 2022, pp. 1–3. DOI: `10.1109/ITNAC55475.2022.9998348`.

[5] A. Sahai, B. Waters, and H. Wee, "Attribute-Based Encryption with Verifiable Outsourced Decryption", in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ACM, Scottsdale, Arizona, USA, 2014, pp. 129–140. DOI: `10.1145/2660267.2660296`.

[6] P. Mell and T. Grance, *The NIST Definition of Cloud Computing* (NIST Special Publication). National Institute of Standards and Technology, 2011, vol. 800-145. DOI: `10.6028/NIST.SP.800-145`.

[7] Palo Alto Networks, *Cloud IAM Security: How to Make Sure Your Cloud Data Stays Safe*, `https://www.paloaltonetworks.com/blog/2020/02/cloud-iam-security/`, Accessed: 2023-03-11, 2020.

[8] *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)*, Accessed: 2023-02-26, Luxembourg: European Parliament and Council, 2016. [Online]. Available: `https://eur-lex.europa.eu/eli/reg/2016/679/oj`.

[9] Joint Task Force Transformation Initiative, ”Security and Privacy Controls for Federal Information Systems and Organizations”, National Institute of Standards and Technology, Tech. Rep. NIST Special Publication 800-53, Revision 5, 2020, Accessed: 2023-03-15.

[10] S. Goldwasser, S. Micali, and C. Rackoff, ”The knowledge complexity of interactive proof-systems”, *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pp. 291–304, 1985.

[11] Ethereum. ”Zero-Knowledge Proofs”. (2021), [Online]. Available: `https://ethereum.org/en/zero-knowledge-proofs/` (visited on 03/17/2022).

[12] C.-P. Schnorr, ”Efficient Identification and Signatures for Smart Cards”, in *Advances in Cryptology—CRYPTO'91*, Springer, 1991, pp. 239–252.

[13] A. Fiat and A. Shamir, ”How to Prove Yourself: Practical Solutions to Identification and Signature Problems”, in *Advances in Cryptology — CRYPTO '86*, Santa Barbara, California, USA, 1987, pp. 186–194.

[14] J. Groth and M. Kohlweiss, ”Non-Interactive Zero-Knowledge from (Indented) Simulation Extractable Subversion-Resistant NIZK Proofs”, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9453, Springer Verlag, 2015, pp. 41–60. DOI: `10.1007/978-3-662-48797-6_3`.

[15] E. Ben-Sasson, A. Chiesa, C. Garman, *et al.*, ”Zero-Knowledge Proofs for Circuit Evaluation”, in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 826–837.

[16] ConsenSys, *Introduction to zk-SNARKs*, Accessed: 2023-03-19, 2019. [Online]. Available: `https://consensys.net/blog/developers/introduction-to-zk-snarks/`.

[17] V. Goyal, O. Pandey, A. Sahai, and B. Waters, ”Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data”, in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, Alexandria, Virginia, USA, 2006, pp. 89–98.

[18] D. Boneh, B. Lynn, and H. Shacham, ”Short Signatures from the Weil Pairing”, in *Advances in Cryptology — ASIACRYPT 2001*, Springer, Berlin, Heidelberg, 2001, pp. 514–532. DOI: `10.1007/3-540-45682-1_30`.

[19] J. Hur and D.-K. Noh, ”Attribute-Based Access Control with Efficient Revocation in Data Outsourcing Systems”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, pp. 1214–1221, 2011. DOI: `10.1109/TPDS.2010.183`.

[20] *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)*, Accessed: 2023-

02-26, Luxembourg: European Parliament and Council, 2016. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj?uri=celex:32016R0679#d1e1685-1-1.

[21] M. Morello, *Attribute Compliance Verifier*, Accessed: 2023-05-15, 2023. [Online]. Available: https://github.com/Massimore/attribute-compliance-verifier.

[22] A. Shareiyat and M. K. Hasan, ”Machine-readable Regulatory and Compliance in Cloud Computing”, Degree project 30 HE credits, Computer and Systems Sciences, Spring term, Degree project at the master level, Stockholm University, Department of Computer and Systems Sciences, 2022.