PROUD: verifiable privacy-preserving outsourced attribute based signcryption supporting access policy update for cloud assisted IoT applications

Sana Belguith, Nesrine Kaaniche, Mohammad Hammoudeh, Tooska Dargahi

---

Article:
Belguith, S., Kaaniche, N. orcid.org/0000-0002-1045-6445, Hammoudeh, M. et al. (1 more author) (2020) PROUD : verifiable privacy-preserving outsourced attribute based signcryption supporting access policy update for cloud assisted IoT applications. Future Generation Computer Systems, 111. pp. 899-918. ISSN 0167-739X

https://doi.org/10.1016/j.future.2019.11.012

PROUD: Verifiable **Pr**ivacy-preserving **Ou**tsourced Attribute Based SignCryption Supporting Access Policy **Upd**ate for Cloud Assisted IoT Applications

Sana Belguith[a,*], Nesrine Kaaniche[b], Mohammad Hammoudeh[c], Tooska Dargahi[a]

[a] *School of Computing, Science and Engineering, University of Salford, Manchester, UK*
[b] *Department of Computer Science, University of Sheffield, Sheffield, UK*
[c] *Manchester Metropolitan University, Manchester, UK*

Abstract

The ever-growing number of Internet connected devices poses several cybersecurity risks. Most of the exchanged data between the Internet of Things (IoT) devices are not adequately secured due to resource constraints on IoT devices. Attribute Based SignCryption (ABSC) is a powerful cryptographic mechanism suitable for distributed environments, providing flexible access control and data secrecy. However, it imposes high designcryption costs, and does not support access policy update (user addition/revocation). This paper presents PROUD, an ABSC solution, to securely *outsource* data designcryption process to edge servers in order to reduce the computation overhead on the user side. PROUD allows end-users to offload most of the designcryption overhead to an edge server and verify the correctness of the received partially designcrypted data from the edge server. Moreover, PROUD provides the access policy update feature with neither involving a proxy-server, nor re-signcrypting the signcrypted message and re-distributing the users' secret keys. The access policy update feature in PROUD does not affect the size of the message received by the end-user which reduces the bandwidth and the storage usage. Our comprehensive theoretical and experimental analysis prove that PROUD outperforms existing schemes in terms of functionality, communication and computation overhead.

Keywords: Attribute Based Signcryption, Access Policy Update, Outsourced Designcryption, Cloud Assisted IoT, Privacy, Confidentiality, Access Control, Anonymous Data Origin Authentication.

1. Introduction

The Internet of Things (IoT) has sparked a significant shift in all aspects of our lives including business, industry and society. Todays availability of low-cost embedded sensors and actuators offer unprecedented opportunity for interconnecting smart buildings, factories, vehicles, power grids and other data infrastructures. The vast volume of connected devices feeding into networks introduced a wide spectrum of privacy risks and security vulnerabilities. In such hyper-connected environments, the identification of a specific object or its owner raises a crucial challenge that affects the system's security features such as privacy, confidentiality and access control. While combining various data sources from different data owners, devices, and applications enhances service quality and availability, it may also lead to severe data and privacy leakage, e.g., through data correlation techniques. For instance, user's purchase behaviors might be deduced through a smart card service, a smart home may disclose which appliances are used and a smart mobility application may be used to reveal users locations and infer their preferences.

Today, consumers' increased security awareness combined with regulatory reforms such as the General Data Protection Regulation (GDPR) calls for new measures to protect personal data and preserve citizen's privacy [1]. Often, not all privacy requirements can be satisfied only by authentication, but support for data confidentiality at different system levels is an essential property. Recently, the evolution of edge computing offered new capabilities to secure IoT ecosystems. Outsourcing computationally intensive cryptographic operations to the cloud offers an opportunity to equip IoT devices with powerful security solutions. Attribute Based SignCryption (ABSC) is a powerful cryptographic mechanism that provides data secrecy, flexible access control to enciphered data as well as data authentication [2, 3]. ABSC is an association of Attribute Based Encryption (ABE) and Attribute Based Signature (ABS). Using ABSC techniques, the data owner can encrypt and sign data w.r.t. an access policy, in only one step, before outsourcing to an untrusted third party. Data can be recovered only by authorised users whose access rights satisfy the signcryption access policy.

Attribute based signcryption has many properties that make it an attractive solution for several security and privacy concerns in decentralised and distributed architectures. However, in its standard form, ABSC suffers from severe drawbacks that limit their application to resource constrained systems. On the one hand, ABSC techniques incur high designcryption costs due to

*s.belguith@salford.ac.uk
Email addresses: s.belguith@salford.ac.uk (Sana Belguith), n.kaaniche@sheffield.ac.uk (Nesrine Kaaniche), m.hammoudeh@mmu.ac.uk (Mohammad Hammoudeh), t.dargahi@salford.ac.uk (Tooska Dargahi)

the execution of several pairing functions to verify and recover a plaintext. On the other hand, the update of access policies is not supported. Thus, the addition/revocation of users needs the re-signcryption of the signcrypted message and the re-distribution of users' secret keys. Please refer to [4] for a comprehensive survey on ABSC schemes in the cloud environment.

In this paper, we design a verifiable privacy-preserving outsourced attribute based signcryption scheme, so-called PROUD, to support access policy update for cloud assisted IoT applications without any ciphertext re-signcryption. In PROUD, the signcrypting entity generates a signcryption of the message, which includes the signcrypted data combined with additional components required to perform the access policy update. These additional components are simply a randomization of the signcryption elements w.r.t. the attributes universe. The cloud server uses these additional components to update the access policy on demand with neither designcrypting the data nor re-issuing users' secret keys. It is worth mentioning that the end-user will only receive "eight" elements of the ciphertext as the revocation/addition of attributes will be executed only on the cloud side. In other words, the end-user will only receive a constant-size signcrypted message without having information about the access policy updates (attribute addition and/or revocation). To the best of our knowledge, we are the first to propose an ABSC scheme enabling adding/removing attributes to/from the access policy in order to add/revoke users with neither involving a proxy server nor re-signcrypting data. Note that, the policy update feature has been only proposed in key policy attribute based encryption [5] and ciphertext policy attribute based encryption [6] which are two encryption schemes, not signcryption schemes.

PROUD also supports secure delegation of the designcryption algorithm to a Semi-Trusted Edge Server (STES). STES can partially designcrypt the signcrypted data[1] and forward the result to the requesting user. This latter can then retrieve the plaintext by executing low cost mathematical operations. To achieve a secure delegation, PROUD allows the user to verify the accuracy of the received partially designcrypted ciphertext to ensure that it has been honestly generated by the STES. These features make PROUD useful to be applied in dynamic environments requiring efficiently adding/removing users. PROUD ensures low storage and computation costs to suit resource-constrained end-users, such as in vehicular networks [7], e-health systems [8] and Publish and Subscribe systems (Pub/Sub) [9, 10].

The contributions of this work are as follows:

1. PROUD provides flexible access control over outsourced data to remote cloud servers. That is, thanks to the features of the attribute based signature, privacy of the signing data owner is preserved as the derived signature does neither reveal his identity nor the attributes used during the signing process.
2. PROUD ensures the privacy-preserving data origin authentication feature, which guarantees that outsourced contents are uploaded and modified by an authorised data owner, w.r.t. his granted privileges.
3. PROUD supports access policy update without requiring cipherext re-signcryption or re-issuing users' secret keys. PROUD does not rely on a proxy re-encryption server to execute policy update procedures.
4. We extend the fixed-size attribute based signcryption scheme proposed by Belguith et al. [11] to delegate the designcryption computation overhead to the STES. This latter performs most of the designcryption operations without accessing the plaintext, and returns a partially designcrypted ciphertext to the intended user. In return, the user can retrieve the plaintext by executing low cost mathematical operations.
5. PROUD allows the end-user to verify the accuracy of the partially designcrypted message received from the STES. This property is referred to as *verifiability*. Indeed, the user is able to check that the retrieved plaintext matches the ciphertext originally requested and downloaded from the cloud server.
6. Unlike most ABSC techniques, the size of the ciphertext that the end-user will receive is constant (only eight elements) and independent from the number of attributes involved in the access policy. Therefore, PROUD generates a constant-size ciphertext, w.r.t. the end-user, that reduces bandwidth utilisation and storage costs.

The reminder of the paper is organised as follows: Section 2 introduces the problem statement w.r.t. a real-world use case scenario, i.e., cloud assisted vehicular network, and details the security requirements that have to be fulfilled by the proposed scheme. In Section 3, a number of closely related ABSC schemes are reviewed and compared w.r.t. a set of features namely the support of policy updates and outsourced-decryption processing. Section 4 details the general architecture and involved actors, gives a general overview of the proposed scheme and presents the system and security models. Section 6 introduces the concrete construction and details main algorithms and phases. Section 7 gives a detailed security analysis of PROUD w.r.t the defined threat model. Finally, a comparative theoretical analysis of computation and communication costs for PROUD is discussed in Section 8 before concluding the paper in Section 9.

2. Problem Statement

In this section, we present a use-case example, i.e., secure communication in vehicular networks, to show how our proposal could be used in real-world scenarios. Then, we explain the main security requirements to be addressed by the proposed scheme.

2.1. Cloud Assisted Vehicular Networks

During the past decade, attribute based encryption has been increasingly used to secure communication in vehicular networks [12]. Due to its unique features in ensuring fine-grained access control, ABE has been considered as a good candidate

[1] For ease of presentation, ciphertext will be used to refer to signcrypted data

for designing secure and privacy preserving protocols for cloud assisted vehicular networks as well [13]. More recently, due to the emergence of autonomous vehicles (AVs), researchers took advantage of ABE for securing message exchange in AV platooning[2] [14].

As a use-case example we consider a cloud assisted vehicular network in which a message should be exchanged between an entity (which we call it data owner) and a group of vehicles (which we call them data users). A security requirement here would be confidentiality of the exchanged/broadcast message. Moreover, to ensure that the broadcast messages are genuine, the receiving vehicles should be able to verify the authenticity of the messages.

Let us consider a vehicular network composed of several transport service providers who manage different kinds of vehicles, such as Taxi cabs, buses, and trucks, in a city. Each vehicle could be assigned with a set of attributes used to identify the type of vehicle and the service provider (e.g., company *A* or *B*). The data owner in our use-case scenario could be a service provider or a vehicle. For example, a Taxi driver that works for the company *B* might need to share conventioneers pickup location information with his colleagues. In this scenario, the driver can broadcast an encrypted message defining an attribute based access policy (such as $\langle Company\ B \rangle\ AND\ \langle location\ x \rangle$) [12]. This will ensure the confidentiality of the exchanged message and helps in reducing the number of messages to be processed by those vehicles that do not satisfy the access policy. As another example consider an AV platoon, which could be composed of different vehicle types, such as cars, buses and trucks, that communicate with each other (i.e., V2V communication) and with the infrastructure (i.e., V2I communication). The service provider *A* might need to share specific traffic or route information only with the trucks managed by *A*. In this case, it can encrypt the message defining an access policy $\langle Company\ A \rangle\ AND\ \langle vehicle\text{-}type\ truck \rangle$. Therefore, only the trucks that belong to the company *A* will be able to decrypt the message.

In both scenarios, the usage of attribute based encryption will provide confidentiality and fine-grained access control. However, this would not ensure the authenticity of the message sender. To solve this issue, in [13] an identity-based sequential aggregate signature is used, while in [14] the message sender uses symmetric key encryption to sign the messages before encrypting it by ABE keys. We believe that ABSC could be a promising solution for our use-case scenario, as researchers [2, 11, 15] have shown that ABSC is an efficient technique to perform both encryption and signing at the same time.

Since Onboard Units (OBUs) are generally resource-constrained [13], other challenges in our use-case scenarios would be: i) the number of exchanged messages, as well as ii) the computation overhead that is imposed by ABE operations. A solution for the first issue could be the usage of cloud storage to upload the encrypted messages instead of broadcasting each message in the network. For the latter issue, a possible solution would be the usage of edge servers, e.g., Road Side Units (RSUs), to perform a part of the computation (which we will explain in Section 4).

In the following section, we discuss the security and functional requirements that have to be fulfilled by the proposed mechanism.

2.2. Security Requirements

To design a secure attribute based signcryption scheme that supports the access policy update, the following requirements are considered in the design of PROUD:

- ***flexible fine-grained access control –*** provides flexible security policies among dynamic groups of users to the outsourced data contents.
- ***data authenticity –*** ensures that data has been generated by authorised users.
- ***data confidentiality –*** protects the content of the outsourced data against unauthorised entities.
- ***privacy –*** preserves the privacy of data owners and data users when creating and authenticating or accessing data.
- ***low processing and storage overheads –*** incur low computation and storage costs at end-users' side, mainly considered as resource-constrained devices.

3. Attribute Based Techniques: Related Work

In this section, we first review ABSC schemes with constant ciphertext size. Then, we present ABSC schemes supporting oustourced designcryption before introducing policy update mechanism applied to attribute based schemes.

3.1. Constant Size Attribute Based Signcryption

Attribute based SignCryption (ABSC) is considered as a flexible cryptographic primitive that provides confidentiality, fine-grained access control, data authentication and privacy preservation [16]. As detailed above, data is encrypted and signed w.r.t. an access policy, in one single logical step, before being outsourced to an untrusted third party. Afterwards, data contents can be recovered only by authorised users whose access rights satisfy the access policy embedded in the signcrypted data.
However, the main drawback of the most proposed ABSC schemes in the literature is the size of the generated ciphertext that is proportional to the number of attributes used in the access policy; i.e., the greater number of attributes the bigger ciphertext size. To address this limitation, several schemes have been proposed in the literature, which we review some of the prominent and recent ones that aim to achieve fixed-size ciphertext.

In 2016, Rao et al. designed the first key-policy attribute based signcryption scheme (KP-ABSC) [17]. The length of the derived signcrypted message is limited to 6 group elements that are independent from the size of the access policy, i.e., the number of attributes included in the access structure.

[2] A transportation system which allows vehicles to travel close to each other with constant speed

In 2017, a ciphertext policy attribute based signcryption scheme (CP-ABSC) introduced in [11]. The proposed scheme supports threshold access policies and constant-size ciphertext features. Indeed, the data owner signcrypts his data w.r.t. a defined threshold access policy, before outsourcing to remote cloud servers. Thus, an authorised end-user is able to both verify if the received data is uploaded by an authorised data owner, and decipher the received data content w.r.t. granted privileges.

Later, Rao et al. [18] introduced a CP-ABSC scheme, that permits to ensure the security of personal health records sharing among different groups of cloud users. The proposed scheme exhibits a constant ciphertext size, adapted to several resource-constrained devices.

Most of the attribute based signcrytion schemes, such as [17, 11, 18, 19], mainly suffer from high designcryption cost. This is generally due to the intensive mathematical computations and the complexity of access structures associated with ciphertexts or users' private keys. To address this concern, outsourcing the designcryption process in attribute based singcryption to an edge server or cloud has been proposed in the literature, which will be discussed in the following section.

3.2. Outsourcing Attribute Based SignCryption

Outsourcing decryption in ABE has been first introduced by Green et al. [20]. This technique is based on deriving a public key form the user's private key that is shared with a semi-trusted server. This latter is able to partially decrypt a particular ciphertext using the generated key. This outsourcing mechanism has been applied recently to attribute signcryption schemes. For instance, Chen et al. [15] have designed a KP-ABSC scheme that supports outsourced de-signcryption. In this scheme, the user shares a transformation key with the cloud server to allow the partial designcryption of the ciphertext. After receiving the result from the cloud server, the user proceeds to totally decrypt the data file. In [21] all the key generation, signing, encryption, decryption and verification are offloaded to cloud service providers.

The basic outsourcing techniques applied in the aforementioned schemes [20, 15] requires the user to fully trust the server performing the partial decryption. However, a *lazy* server may return a previously computed designcrypted ciphertext. To make outsourcing technique more secure, the *verifiability* notion has been introduced in [22]. This concept enables the user to verify that the partially designcrypted ciphertext has been genuinely generated from the ciphertext forwarded to a semi-trusted server. Several ABE schemes supporting decryption outsourcing have adopted the verifiability technique, e.g., [23, 24, 25]. For instance, in [24], the authors present a multi-authority ciphertext-policy attribute based encryption scheme that supports the delegation of the decryption process to an untrusted server. Their proposal provides a verifying mechanism enabling users to check either the partially deciphered content was correctly generated, based on randomly generated tokens.

Recently, Deng et al. [25] have introduced an ABSC scheme that supports verifiable outsourced designcryption. The proposed construction enables the end-user to verify the correctness of the partially designcrypted ciphertext relying on some elements of the ciphertext.

Liu et al. [26] propose a KP-ABSC scheme which ensures outsourced designcryption. However, in this scheme, the end-user trusts the edge server and shares his secret keys to enable this server to partially designcrypt the ciphertext.

None of the aforementioned constructions consider the highly dynamic aspect of IoT environments, i.e., they do not provide efficient policy-update mechanisms to support the frequent enrolment and revocation of end-users.

3.3. Attribute Based SignCryption supporting Policy Updates

One of the limitations of the current attribute based signcryption techniques is the cost of updating access policies after generating a ciphertext, i.e., the addition or deletion of attributes from existing access policies requires re-encrypting data. Moreover, policy update often relies on proxy re-encryption mechanism [27, 28, 26, 29]. The re-encryption process is expensive in terms of communication cost and is particularly inefficient when the number of ciphertexts grows. Although some of the proposed algorithms, such as [26, 29], are capable of revocation/addition of users, they still require sharing re-encryption keys with the proxy server which adds extra communication costs. In [6], attributes can be added or removed efficiently without sharing re-encryption keys. The first KP-ABE schemes supporting policy update are recently presented in [5, 30, 31]. Both schemes presented in [6] and [5] enable a cloud server to update the access policy associated with a ciphertext without relying on a proxy server to execute re-encryption algorithm neither re-issuing users' keys.

Table 1 presents a comparison between PROUD and most closely related schemes in the literature, in terms of considered features. It shows the type of attribute based technique in use, i.e, ciphertext policy or key-policy, as well as the type of access policy (monotone, or threshold). It also depicts the support for outsourced decryption, policy update, verifiability and constant-size ciphertext features. As it can be seen in Table 1, PROUD is the only ABSC scheme that covers all the four considered features. Moreover, PROUD and [11] are the only *threshold* ABSC schemes. This feature provides more flexibility in terms of the number of attributes satisfying the access policy. In addition, other schemes study the security aspects in a theoretical framework and overlook the issues related to the highly dynamic nature of IoT environments.

In this paper, we design a new privacy-preserving protocol, that supports the requirements listed in Section 2, and we provide a detailed performance analysis to highlight the advantage of PROUD compared to the state-of-the-art in terms of processing and storage overheads. As it can be seen in Table 1, PROUD is actually a CP-ABSC scheme. The reason behind this choice is that CP-based schemes are more appropriate for access control applications compared to KP-based schemes [32]. This is due to the fact that in CP-based schemes the data owner has more control on deciding the access policy and who can access the data. However in KP-based schemes the users' primitives are associated with the access policy and the data owner only decides a set of attributes to be associated to the encrypted data.

Table 1: Comparison of the PROUD features and the existing ABE/ABSC schemes.

Scheme	*Type*	*Access Policy*	*Outsourced Decryption*	*Policy Update*	*Verifiability*	*Constant Ciphertext*
[2] (2010)	CP-ABE	Threshold	✗	✗	✗	✗
[33] (2012)	KP-ABSC	Monotone	✗	✗	✗	✗
[34] (2015)	CP-ABSC	Monotone	✗	✗	✗	✗
[17] (2016)	KP-ABSC	Monotone	✗	✗	✗	✓
[15] (2016)	CP-ABSC	Monotone	✓	✗	✗	✗
[11] (2017)	CP-ABSC	Threshold	✗	✗	✗	✓
[18] (2017)	CP-ABSC	Monotone	✗	✗	✗	✓
[6] (2017)	CP-ABE	Threshold	✗	✓	✗	✓
[26] (2017)	CP-ABSC	Monotone	✓	✗	✓	✗
[25] (2018)	CP-ABSC	Monotone	✓	✗	✓	✓
[29] (2018)	CP-ABSC	Monotone	✓	✗	✓	✗
[5] (2018)	KP-ABE	Monotone	✗	✓	✗	✓
[30] (2018)	KP-ABE	Monotone	✗	✓	✗	✓
PROUD	CP-ABSC	Threshold	✓	✓	✓	✓

4. PROUD Specification

In this section, we first present PROUD system model, explaining the involved actors, core building blocks and phases of the system initialization and usage in Subsection 4.1. We detail the considered security model in Subsection 4.2.

4.1. System Model

In Figure 1 we show the five core entities of PROUD. We detail the main responsibility of each actor and the interaction between different entities in the following.

The main actors of our system model are:

- The Central Trusted Authority (CTA), is considered as trusted by all the involved entities in our system model. It is responsible for both deriving entities' private keys and initialising the system by public global parameters.
- The Cloud Service Provider (CSP) manages a set of remote servers and data centers and permits to store and share outsourced data contents among authorised users w.r.t. their granted privileges. CSP is also responsible of executing the ciphertext access policy update algorithm according to the data owner's recommendations.
- The Semi-Trusted Edge Server (STES) is responsible for partially decrypting a ciphertext using a transformation key received from the user. Indeed, the user derives a couple of public and private transformation keys from his secret keys. The user shares the public key with STES to allow the partial decryption of ciphertext, while keeping secret the private transformation key to be used for the final local data retrieval.
- The data owner (O) is responsible for defining the deciphering access policy. Before outsourcing data to remote cloud servers, O has to first signcrypt data w.r.t. defined access policies. In order to ensure the policy update feature, the data owner generates additional ciphertext components which are mainly a randomization of the ciphertext elements w.r.t. the attributes universe. These additional components are used by the cloud server to update the access policy on demand without designcrypting any ciphertext neither re-issuing any secret keys.
- The data user (U) requests access to the outsourced data. U delegates a part of designcryption process to the STES. Specifically, U is responsible for using his secret keys to derive transformation keys used by the STES to partially designcrypt the ciphertext. Then, U designcrypts and verifies the partially designcrypted ciphertext that is received from the STES.

The PROUD scheme is composed of four phases, i.e., SYS_INIT, STORAGE, UPDATE and RETRIEVAL, defined with respect to seven randomized algorithms detailed in Subsection 4.1.
The SYS_INIT phase is executed once by the CTA. It permits to generate and publish system public parameters to *all* involved entities and derives users' private keys associated to their attributes. The SYS_INIT phase is based on two algorithms, denoted by Setup and KeyGen.

During the STORAGE phase, the data owner (O) who has already received a predefined signing access policy (t, S_s), has to first define the deciphering policy, referred to as S_e. To this end, O has to create a subset S_e from the attributes universe $\mathbb{U}$ where $|S_e| = |S_s|$. O chooses a threshold t such that $1 \leq t \leq |S_e|$ to define the encryption access policy (t, S_e). Recall that the encryption access policy (t, S_e) defines which certified attributes (i.e., private keys associated to attributes) are needed to be able to decipher the signcrypted content, while the signing policy (t, S_s) defines the credentials that need to be fulfilled by the signing data owner. The STORAGE phase includes one randomized algorithm, denoted by SignCrypt, to signcrypt the data content w.r.t. the access policies (t, S_s) and (t, S_e), while pointing out

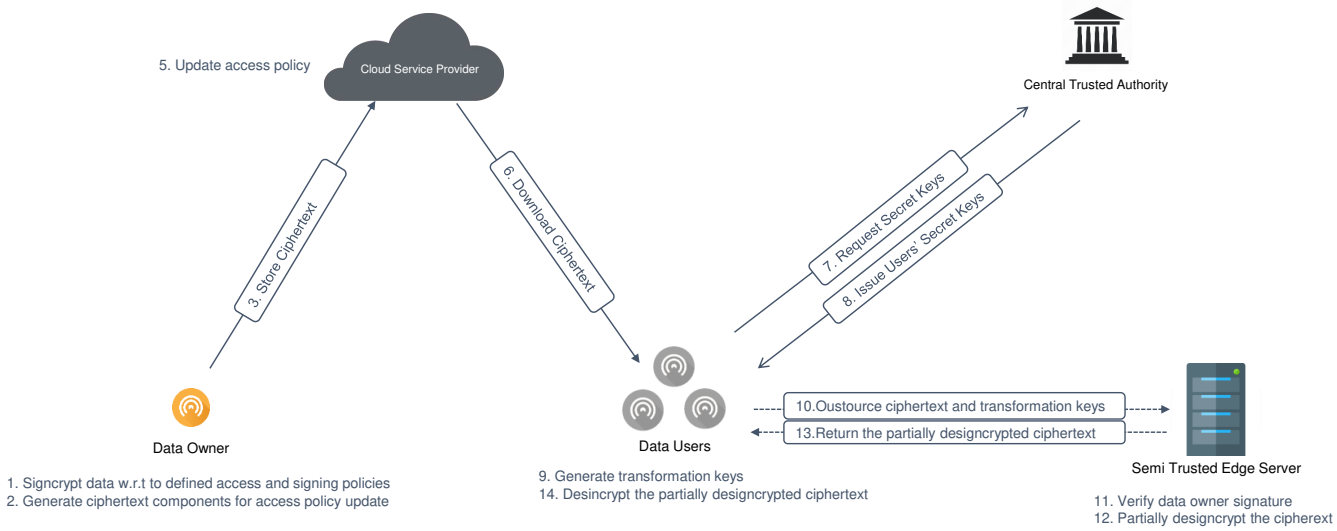

Figure 1: PROUD architecture and key interactions between different entities. The data owner takes advantage of cloud servers to share his data with a set of users, specifying access control policies and signcrypting the data. We consider data users to be resource-constrained IoT devices. The data users delegate a part of data designcryption to edge servers, which is more powerful than the IoT devices.

either the data content can be updated or not.

The UPDATE phase is executed by the cloud provider, upon the request of the data owner. It is based on one algorithm, denoted by Update that supports both the addition and removal of attributes from access policies.

During the RETRIEVAL phase, the user (U) has first to authenticate with the cloud provider and request access to a particular data content. In this paper, we skip the details of the authentication process, as several solutions have been already proposed in the literature which could be used, such as [35, 36]. Once authenticated, U runs an interactive protocol with the STES, to recover the original data content. The RETRIEVAL phase relies on three different algorithms, referred to as Transform, $\mathsf{DesignCrypt}_{\mathsf{out}}$ and DesignCrypt. Indeed, the data user U executes the Transform algorithm to derive a *transformation* key, relying on his private keys that satisfy the encryption access policy (t, S_e) associated to the requested data content. The transformation key is then sent to the STES. This latter performs the $\mathsf{DesignCrypt}_{\mathsf{out}}$ algorithm and generates a partially decrypted data content. Finally, based on the partially deciphered data file, U is able to finalise the designcryption process. Recall that U can check whether the received partially deciphered content was correctly generated, thanks to the support of the *verifiability* property.

It is worth mentioning that, in order to reduce the communication costs, one could consider a scenario in which the data user delegates the ability of downloading the ciphertext to the STES. However, the user still needs to download a part of the ciphertext to be able to check whether the received partially designcrypted content from the STES was correctly generated, to be compliant with the *verifiability* property. In addition, the user has to communicate with the STES to share the transformation public keys (tpk) required to execute the $(DesignCrypt_{out})$ algorithm.

PROUD involves seven randomized algorithms, w.r.t. four phases defined respectively as follows:

- SYS_INIT phase:

  $\mathsf{Setup}(\xi) \rightarrow (\mathtt{pp}, \mathtt{msk})$ – performed by a CTA. Given the security parameter ξ, this algorithm generates the public parameters pp and the secret master key msk.

  $\mathsf{KeyGen}(\mathtt{pp}, \mathtt{msk}, i, A_i) \rightarrow sk_i$ – run by CTA in order to generate the private keys of the user i, where $i \in \{O, U\}$. It takes as input pp, a user's attribute set $A_i \subset \mathbb{U}$ and the secret master key msk. The KeyGen algorithm returns the user's secret key sk_i w.r.t. A_i.

- STORAGE phase:

  $\mathsf{SignCrypt}(\mathtt{pp}, sk_O, A_O, (t, S_e), (t, S_s), M) \rightarrow \Sigma$ – executed by the data owner. Given the public parameters pp, the secret key of the data owner sk_O and their related set of attributes A_O, the access policies (t, S_e) and (t, S_s) and the message M, where $M \in \mathbb{M}$ (i.e., $\mathbb{M}$ refers to the message

space), the SignCrypt algorithm returns the signcrypted message Σ.

- UPDATE phase:

  $\mathsf{Update}(\mathsf{pp}, \Sigma, \mathsf{ind}, \mathcal{U}) \rightarrow \Sigma_{up}$– runned by a cloud server upon the demand of the data owner. This algorithm takes as input pp, a signcrypted message Σ which contains the set of enciphering attributes $S_e = \{a_i\}_{i=1..s}$ where $|S_e| = s$, an operation indicator ind such that ind may refer to either attributes' addition, i.e., ind = add or attributes' removal, i.e., ind = revoke and $\mathcal{U}$ a new set of attributes where $\mathcal{U} \cap S_e = \emptyset$ if ind = add or $\mathcal{U} \subset S_e$ if ind = revoke. This randomized algorithm generates a new/updated signcrypted message Σ_{up} based on the new encrypting set of attributes $\mathcal{S}'_e$ such as $S'_e = S_e \cup \mathcal{U}$ or $S'_e = S_e \setminus \mathcal{U}$ w.r.t. ind value.

- RETRIEVAL phase:

  $\mathsf{Transform}(\mathsf{pp}, sk_U, (t, S_e)) \rightarrow tk_U$ – performed by U having a set of attributes A_U and their related secret keys sk_U. Transform takes as input pp, sk_U and the encryption access policy (t, S_e). It generates the transformation key $tk_U = (tpk_U, tsk_U)$ related to sk_U, where tpk_U and tsk_U are the public and private transformation keys respectively.

  $\mathsf{DesignCrypt}_{\mathsf{out}}(\mathsf{pp}, tpk_U, (t, S_e), (t, S_s), \Sigma) \rightarrow \Sigma_{part}$ – is executed by the Semi-Trusted Edge Server (STES). To retrieve the partially designcrypted message Σ_{part}, this algorithm takes as input pp, the transformation key tpk_U, access policies (t, S_e) and (t, S_s) and the signcrypted message Σ. Note that Σ can be Σ_{up} if the Update algorithm has been executed.

  $\mathsf{DesignCrypt}(tsk_U, \Sigma_{part}) \rightarrow M - U$ executes DesignCrypt to retrieve M. This algorithm takes tsk_U and the partially designcrypted message Σ_{part} as input and outputs M.

4.2. Security Model

In this section, we explain the considered attack model based on which we discuss the security and privacy properties of our proposed scheme.

- *An honest but curious cloud server provider (CSP).* CSP is honest as it generates accurate inputs or outputs, during the different steps of the protocol, and performs calculations properly. However, it is curious to gain extra data from the protocol, such as obtaining credentials/attributes of a data user, retrieving the plaintext, or distinguishing the data owner based on the signcrypted content. As such, we consider the honest but curious attack model against the confidentiality and privacy requirements w.r.t adaptive replayable chosen-ciphertext attacks (IND-RCCA2) (c.f, 4.2.1) and the privacy of the data owner (c.f, 4.2.3).

- *An unauthorised data user.* This attacker could be a data user (or an external entity), whose attributes do not satisfy the access policy associated with the ciphertext, or could be a revoked user. We also consider a set of colluding users on the attributes, who do not satisfy the access policy associated with the ciphertext and try to merge their attributes to retrieve the plaintext, in this attack model. These unauthorised users attempt to designcrypt the content and access the plaintext. As such, we consider this attack model against the confidentiality requirements w.r.t adaptive replayable chosen-ciphertext attacks (IND-RCCA2) (c.f, 4.2.1).

- *A non-authentic data owner.* This attacker can be a malicious data owner or an external malicious entity aiming to sign and upload a non-genuine ciphertext to the CSP to be shared among users. A non-authentic data owner is an entity whose attributes do not satisfy the signing policy associated to the ciphertext. As such, we consider the non-authentic data owner security model mainly against the unforgeability requirement considering chosen message attacks (CMA) (c.f, 4.2.2).

- *A lazy STES.* This attacker tries to forge a non-authentic partial designcryption of a ciphertext, or returns an old previously computed ciphertext. A lazy STES is also curious to distinguish the data owner based on the encrypted content. A security scheme against this attacker should satisfy verifiability feature (c.f, 4.2.4), and the computational privacy of the data owner (c.f, 4.2.3).

4.2.1. Confidentiality

The confidentiality property ensures that non-authorised entities cannot decipher a singcrypted -updated- message. As detailed in our attack model, a non-authorized entity can be an honest but curious CSP or a malicious data user who try to override their rights to retrieve the plaintext.

For this purpose, we define Exp^{conf}, a security game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The adversary tries to decipher a signcrypted message, i.e; to distinguish between two randomly generated signcrypted messages, without having sufficient deciphering credentials. In their standard definition, ABSC schemes are proved to be secure based on Chosen Ciphertext Attack (CCA2) security games. For instance, CCA2 is a security game that captures the resistance of a public key encryption scheme against adversaries. In CCA2, the adversary has access to a decryption oracle, however, this access is limited until the challenge ciphertext is known. Further details about these security notions can be found in [37]. However, CCA2's definition does not allow any alteration to the ciphertext which makes CCA2 games not suitable with added functional features such as outsourcing and policy updates. Therefore, in this security game, we apply the relaxation notion proposed by Canetti et al. [38, 20] called Replayable Chosen Ciphertext Attack (RCCA2) which allows minor modifications to the ciphertext during the game without changing the plaintext in

a meaningful way. This newly defined security game has been already adopted by state of the art ABSC schemes supporting outsourced designcryption [29, 15] and ABE schemes ensuring access policy updates [6, 5]. That is, we define a challenger which is responsible for simulating the system procedures. We also define an adversary which can query to get the designcryption of several chosen signcrypted messages. Indeed, Exp^{conf} starts by an initialisation phase such that $\mathcal{A}$ sets a challenge signing and encrypting access policies, defined as $\Psi_s^* = (t, S_s^*)$ and $\Psi_e^* = (t, S_e^*)$ respectively. Both access policies are then sent to $\mathcal{C}$, which later generate the public parameters, then forwards them to $\mathcal{A}$.

Exp^{conf} includes four different phases, such that **Phase 1 Queries** and **Phase 2 Queries** can be repeated as many times as the adversary wants:

Phase 1 Queries – first, $\mathcal{C}$ sets an empty table T. Then, $\mathcal{A}$ is able to request the following queries, for each session i:

- **Private Key Query** – $\mathcal{A}$ queries a signcryption attribute set $A_{\mathcal{A},i}$ w.r.t. a threshold value t_i such that $|A_{\mathcal{A},i} \cap S_e^*| < t_i$ and $|A_{\mathcal{A},i} \cap S_s^*| < t_i$. Then, $\mathcal{C}$ runs the KeyGen algorithm and forwards $sk_{\mathcal{A},i}$ to the adversary $\mathcal{A}$.
- **Transformation Key Query** – $\mathcal{A}$ queries the secret transformation keys $tk_{\mathcal{A},i}$, w.r.t. $A_{\mathcal{A},i}$. $\mathcal{C}$ searches the entry $(A_{\mathcal{A},i}, sk_{\mathcal{A},i}, tk_{\mathcal{A},i})$ in table T. It returns the transformation key if it exists in table T, otherwise $\mathcal{C}$ executes the Transform algorithm to generate $tk_{\mathcal{A},i}$ and forwards them to the adversary.
- **SignCryption Query** – $\mathcal{A}$ requests the signcryption of a message M_i w.r.t. t_i, associated to both signing and encrypting access policies Ψ_s, Ψ_e. Thus, the challenger executes the SignCrypt algorithm and returns a ciphertext Σ_i to $\mathcal{A}$.
- **DesignCryption Query** – $\mathcal{A}$ queries the designcryption of a ciphertext Σ_i w.r.t. t_i and attribute set $A_{\mathcal{A},i}$. For this purpose, $\mathcal{C}$ executes the DesignCrypt[3] algorithm that returns M_i or a reject symbol $\perp$.

Challenge Phase – $\mathcal{A}$ forwards to $\mathcal{C}$ two randomly selected equal-length messages ${M_0}^*$ and ${M_1}^*$ and an attribute set $\mathcal{U}^*$ where $\mathcal{U}^* \cap S_e = \emptyset$ if ind = add or $\mathcal{U}^* \subset S_e$ if ind = revoke. The challenger first chooses a random bit b from $\{0,1\}$ and derives the signcryption secret keys $sk_{\mathcal{C},i}$ associated to the challenge access policies. Then, he checks if $\mathcal{U}^* = \emptyset$, so that he derives ${\Sigma_b}^*$, by executing the SignCrypt algorithm. Otherwise, he generates ${\Sigma_{b_{up}}}^*$ by applying the Update algorithm based on the ind value.

Phase 2 Queries – Having received ${\Sigma_b}^*$ or $\Sigma_{b_{up}}^*$, the adversary is allowed to query a polynomial number of queries as in **Phase 1**, except that he is unable to request the designcryption of the received challenge signcrypted message ${\Sigma_b}^*$ or $\Sigma_{b_{up}}^*$.

Guess – during this phase, the adversary $\mathcal{A}$ attempts to guess which message M_i, such that $i \in \{0,1\}$ corresponds to the received signcryption ${\Sigma_b}^*$ or $\Sigma_{b_{up}}^*$. To do so, $\mathcal{A}$ outputs a bit b' of b and wins the game if $b = b'$. The advantage of the adversary $\mathcal{A}$ in the Exp^{conf} game is defined as $Adv_{\mathcal{A}}[Exp^{Conf}(1^{\xi})] = |Pr[b = b'] - \frac{1}{2}|$.

Definition 1. *PROUD fulfills the confidentiality property by being indistinguishable against replayable chosen ciphertext attacks (IND-RCCA2) if there is no adversary that can succeed in the* Exp^{conf} *security game with non-negligible advantage.*

4.2.2. Unforgeability

The unforgeability property ensures that a non-authentic data owner cannot generate a verifiable correct signature. As detailed in our attack model, a non-authentic data owner can be a malicious/compromised data owner or an external malicious entity.

PROUD is unforgeable against chosen message attack (EUF-CMA) if there is no adversary that can succeed in the Exp^{unf} security game with non-negligible advantage. Similar to the confidentiality game detailed in subsection 4.2.1, we define a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ that interactively conduct Exp^{unf}. Note that Exp^{unf} starts with an initialisation process that permits to set up the challenge access policies as well as generate public parameters. Exp^{unf} includes 2 different phases, where **Phase 1** can be repeated as many times as the adversary wants:

Phase 1 Queries – $\mathcal{C}$ sets an empty table T. Then, for each session i, $\mathcal{A}$ is allowed to issue the following queries:

- **Private Key Query** – $\mathcal{A}$ queries a signcryption attribute set $A_{\mathcal{A},i}$ w.r.t. a threshold value t_i such that $|A_{\mathcal{A},i} \cap S_e^*| < t_i$ and $|A_{\mathcal{A},i} \cap S_s^*| < t_i$. Then, $\mathcal{C}$ runs the KeyGen algorithm and forwards $sk_{\mathcal{A},i}$ to the adversary $\mathcal{A}$.
- **SignCryption Query** – $\mathcal{A}$ requests the signcryption of a message M_i w.r.t. t_i, associated to both signing and encrypting access policies Ψ_s, Ψ_e. $\mathcal{C}$ runs the SignCrypt algorithm and returns a ciphertext Σ_i to $\mathcal{A}$.

Forgery Phase – $\mathcal{A}$ generates a ciphertext Σ^* w.r.t. (Ψ_e^*, Ψ_s^*) access policies (i.e; Ψ_e^*, Ψ_s^* were not queried during **Phase 1 Queries** and where $t_i < t^*$). Then, $\mathcal{A}$ forwards Σ^* to $\mathcal{C}$. This latter runs DesignCrypt algorithm while taking Σ^* as input. $\mathcal{A}$ wins the security game if the signcrypted message Σ^* is accurate and the DesignCrypt algorithm returns a valid message M^* which was not queried in the **SignCryption Query** phase. The advantage of $\mathcal{A}$ is defined as $Adv_{\mathcal{A}}[Exp^{unf}(1^{\xi})] = |Pr[Exp^{unf}(1^{\xi})] = 1$.

Definition 2. *PROUD is unforgeable against chosen-message attacks (EUF-CMA), if the advantage* $Adv_{\mathcal{A}}[Exp^{unf}(1^{\xi})]$ *is negligible for all adversaries.*

[3] As $\mathcal{C}$ executes both $DesignCrypt_{out}$ and $DesignCrypt$ algorithms, we use $DesignCrypt$ to refer to both algorithms

4.2.3. Privacy

We consider the privacy property for two different entities: i) the data user, and ii) the data owner. In the first case, we need to guarantee that a curious entity (which could be the CSP, STES, or an external entity) is not able to gain any information about the attributes of a data user. In fact, this property holds due to the construction of the PROUD. The data user's attributes are neither sent to the CSP nor to the STES. Instead, the data user uses his attributes (i.e., secret keys associated to his attributes) locally to finalise the designcryption process.

In the second case, privacy property ensures that a curious entity (which could be the CSP, STES, data user or even an intruder) is unable to distinguish between two correctly-generated signcrypted messages that have been derived w.r.t. the same set of attributes. That is, an adversary cannot link a signcrypted message to a specific signing entity. To capture the malicious behavior of these entities, in the following we present the Exp^{Priv} security game, that is an interactive game between an honest data owner and a curious entity.

The Exp^{Priv} security includes two phases and starts by an initialisation phase where challenge access policies and public parameters are set up.

Challenge Phase – $\mathcal{A}$ chooses the two required access policies $\Psi_s^* = (t, S_s^*)$ and $\Psi_e^* = (t, S_e^*)$ such that $1 \leq t \leq |S_s^*| = |S_e^*|$, two attribute sets $A_{\mathcal{A},1}$ and $A_{\mathcal{A},2}$ such that $|A_{\mathcal{A},1} \cap S_e^*| = |A_{\mathcal{A},1} \cap S_s^*| = |A_{\mathcal{A},2} \cap S_e^*| = |A_{\mathcal{A},2} \cap S_s^*| = t$, and a message M. $\mathcal{A}$ forwards the tuple $(\Psi_s^*, \Psi_e^*, A_{\mathcal{A},1}, A_{\mathcal{A},2}, M)$ to $\mathcal{C}$. $\mathcal{C}$ chooses a random bit $b \in \{0,1\}$ and runs the KeyGen algorithm to derive a secret key $sk_{\mathcal{A},b}$. Finally, $\mathcal{C}$ derives Σ_b by running the SignCrypt algorithm and sends it to the adversary $\mathcal{A}$.

Guess – $\mathcal{A}$ picks a bit b' and wins the game if $b' = b$. The advantage of $\mathcal{A}$ in $Exp^{Priv}$is defined as $Adv_{\mathcal{A}}[Exp^{Priv}(1^{\xi})] = |Pr[b = b'] - \frac{1}{2}|$.

Definition 3. *PROUD is said to be computationally private if there is no adversary that can succeed in the security game Exp^{Priv} with non-negligible advantage.*

4.2.4. Verifiability

The verifiability property ensures that a lazy STES cannot reply to an outsourcing request coming from an honest data user with a compromised partially designcrypted ciphertext. That is, an honest data user should be able to find out whether the designcrypted data content with the STES assistance is matching the ciphertext downloaded from the CSP or not.

To capture the behavior of the STES, we formally define the Exp^{verif} security game, between a lazy server and a honest data user. The adversary attempts to generate a valid partially designcrypted message without processing the challenge ciphertext. That is, Exp^{verif} starts by an initialisation process that permits to set up the challenge access policies as well as generate public parameters, similarly as detailed in sub-section 4.2.1. Exp^{verif} includes four different phases, where **Phase 1 Queries** and **Phase 2 Queries** can be repeated as many times as the adversary wants:

Phase 1 Queries – first, $\mathcal{C}$ sets an empty table T. Then, for each session i, $\mathcal{A}$ is allowed to issue the following queries:

- **Private Key Query** – $\mathcal{A}$ queries a signcryption attribute set $A_{\mathcal{A},i}$ w.r.t. a threshold t_i where $|A_{\mathcal{A},i} \cap S_e^*| < t_i$ and $|A_{\mathcal{A},i} \cap S_s^*| < t_i$. Then, $\mathcal{C}$ runs the KeyGen algorithm and forwards the generated secret key $sk_{\mathcal{A},i}$ to $\mathcal{A}$.
- **Transformation Key Query** – $\mathcal{A}$ queries the secret transformation keys $tk_{\mathcal{A},i}$, associated to a set of attributes $A_{\mathcal{A},i}$. $\mathcal{C}$ searches the entry $(A_{\mathcal{A},i}, sk_{\mathcal{A},i}, tk_{\mathcal{A},i})$ in table T. It returns the transformation key if it exists in table T, otherwise $\mathcal{C}$ executes the Transform algorithm to generate $tk_{\mathcal{A},i}$ and forwards them to the adversary.
- **SignCryption Query** – $\mathcal{A}$ requests the signcryption of a message M_i with respect to a threshold t_i, while considering the access policies Ψ_e, Ψ_s. Thus, $\mathcal{C}$ executes SignCrypt and returns a ciphertext Σ_i.
- **DesignCryption Query** – $\mathcal{A}$ requests the designcryption of a ciphertext Σ_i with respect to a threshold t_i, while considering the signcryption attribute set $A_{\mathcal{A},i}$. $\mathcal{C}$ replies to the request by executing the DesignCrypt algorithm that returns M_i or a reject symbol $\perp$.

Challenge Phase – $\mathcal{A}$ chooses a challenge message M^* to be sent to $\mathcal{C}$. This latter derives the challenge signcrypted message Σ^*, by running the SignCrypt algorithm and returns it to $\mathcal{A}$.

Phase 2 Queries – Having received Σ^*, $\mathcal{A}$ is allowed to query a polynomial number of queries as in **Phase 1**, except that he is unable to request the designcryption of the received challenged signcrypted message Σ^*.

Forge – $\mathcal{A}$ generates an attribute set $\{A_{\mathcal{A},i}^*\}$ and a random partially designcrypted ciphertext Σ_{part}^* without executing the $\mathsf{DesignCrypt}_{\mathsf{out}}$ algorithm. We suppose that $(A_{\mathcal{A},i}, sk_{\mathcal{A},i}, tk_{\mathcal{A},i})$ is included in the table T, otherwise, $\mathcal{C}$ runs the **Transformation Key Query** and generates this tuple.
$\mathcal{A}$ wins the game if $\mathsf{DesignCrypt}(\mathsf{pp}, tpk_{\mathcal{A},i}, (t, S_e^*), (t, S_s^*), \Sigma) \notin \{M^*, \perp\}$ and the verification of the partially designcrypted ciphertext is valid. $\mathcal{A}$'s advantage is noted as:

$$Adv_{\mathcal{A}}[Exp^{verif}(1^{\xi})] = |Pr[Exp^{verif}(1^{\xi})] = 1|$$

Definition 4. *PROUD fulfills the verifiability property if there is no adversary that can succeed the security game Exp^{verif} with non-negligible advantage.*

5. Mathematical Background

5.1. Complexity Assumptions

The proposed PROUD scheme relies on the Computational Diffie Hellman Assumption (CDH) and the augmented multi-sequence of exponents computational Diffie-Hellman ($(\tilde{l}, \tilde{m}, \tilde{t})$-aMSE-CDH), which was introduced by Delerablée et al. [39, 40] in 2007. These assumptions are defined as follows:

Definition 5. ***Computational Diffie Hellman Assumption (CDH)*** *– Let $\mathbb{G}$ be a group of a prime order p, and g is a generator of $\mathbb{G}$. The CDH problem is, given the tuple of elements (g, g^a, g^b), where $\{a,b\} \xleftarrow{R} \mathbb{Z}_p$, there is no efficient probabilistic algorithm $\mathcal{A}_{CDH}$ that computes g^{ab}.*

Definition 6. ***$(\tilde{l},\tilde{m},\tilde{t})$-augmented multi-sequence of exponents computational Diffie-Hellman ($(\tilde{l},\tilde{m},\tilde{t})$-aMSE-CDH)*** *– The $(\tilde{l},\tilde{m},\tilde{t})$-aMSE-CDH problem related to the group pair $(\mathbb{G},\mathbb{G}_\mathbb{T})$ is to compute $T = e(g_0,h_0)^{k\cdot f(\gamma)}$. It takes as **input**: the vector $\vec{x}_{\tilde{l}+\tilde{m}} = (x_1,\cdots,x_{\tilde{l}+\tilde{m}})^\top$ whose components are pairwise distinct elements of $\mathbb{Z}_p$ which define the polynomials f(X) and g(X) as follows:*

$$f(X) = \prod_{i=1}^{\tilde{l}}(X+x_i); \qquad g(X) = \prod_{\tilde{l}+1}^{\tilde{l}+\tilde{m}}(X+x_i) \tag{1}$$

where the values x_i are random and pairwise distinct of $\mathbb{Z}_p^*$, and the values:

$$\begin{cases} g_0, g_0^{\gamma}, \cdots, g_0^{\gamma^{\tilde{l}+\tilde{t}-2}}, g_0^{k\cdot\gamma\cdot f(\gamma)} \\ g_0^{\omega\gamma}, \cdots, g_0^{\omega\gamma^{\tilde{l}+\tilde{t}-2}} \\ g_0^{\alpha}, g_0^{\alpha\gamma}, \cdots, g_0^{\alpha\gamma^{\tilde{l}+\tilde{t}}} \\ h_0, h_0^{\gamma}, \cdots, h_0^{\gamma^{\tilde{m}-2}} \\ h_0^{\omega}, h_0^{\omega\gamma}, \cdots, h_0^{\omega\gamma^{\tilde{m}-1}} \\ h_0^{\alpha}, h_0^{\alpha\gamma}, \cdots, h_0^{\alpha\gamma^{2(\tilde{m}-\tilde{t})+3}} \end{cases}$$

*Where $k, \alpha, \gamma, \omega$ are unknown random elements of $\mathbb{Z}_p$ and g_0 and h_0 are generators of $\mathbb{G}$. We can solve the problem if we get an **output** $b \in \{0,1\}$ where $b = 1$ if $T = e(g_0,h_0)^{k\cdot f(\gamma)}$ or $b = 0$ when T is a random value from $\mathbb{G}_T$.*

5.2. Collision-Resistant Hash Functions

The proposed PROUD scheme relies on the use of collision-resistant hash functions, defined as follows:

Definition 7. ***Collision-Resistant Hash Function*** *– A hash function $\mathcal{H} : \{0,1\}^n \longrightarrow \{0,1\}^m$, where $n,m \in \mathbb{N}$, is said to be collision-resistant if it satisfies the following two properties:*

- length compressing — *$m > n$, typically $m = n/2$;*
- hard to find collisions — *for all non-uniform probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$, there exists a negligible function ε, such that for all $n \in \mathbb{N}$,*

$$Pr[(x_0,x_1) \leftarrow \mathcal{A}(1^n,\mathcal{H}) : x_0 \neq x_1 \wedge \mathcal{H}(x_0) = \mathcal{H}(x_1)] \leq \varepsilon(n)$$

5.3. Aggregate Algorithm

The proposed PROUD scheme is based on the aggregate algorithm `Aggreg` introduced by Delerablee et al. [39, 40]. This algorithm is explained in the following description:

Let us consider a list of values $\{g^{\frac{r}{\gamma+x_i}}, x_i\}_{1\leq i\leq n}$, where $r, \gamma \in \mathbb{Z}_p^*$ and $x_1,\cdots,x_n$ are pairwise different. Then , the algorithm proceeds as follows:

$$\texttt{Aggreg}(\{g^{\frac{r}{\gamma+x_i}}, x_i\}_{1\leq i\leq n}) = g^{\frac{r}{\prod_{i=1}^{n}(\gamma+x_i)}}$$

Concretely, the `Aggreg` algorithm defines $P_{0,m} = g^{\frac{r}{\gamma+x_m}}$ for each $m \in \{1,\cdots,n\}$. Afterwards, the algorithm computes sequentially $P_{i,m}$ for $i = 1\cdots n-1$ and $m = i+1,\cdots,n$ using the induction:

$$P_{i,m} = \left(\frac{P_{i-1,i}}{P_{i-1,m}}\right)^{\frac{1}{x_m - x_i}} \tag{2}$$

Then, we get $P_{i,m} = g^{\frac{r}{(\gamma+x_m)\prod_{k=1}^{i}(\gamma+x_k)}}$ where $1 \leq i \leq m \leq n$.

Therefore, since the elements $x_1,\cdots,x_n$ are pairwise different [41, 42] and using the equation 2, we can compute $P_{i,m}$ for $i = 1\cdots n-1$ and $m = i+1\cdots n$ such as $P_{n,n-1} = g^{\frac{r}{\prod_{i=1}^{n}(\gamma+x_i)}}$

6. PROUD: Verifiable Outsourced Attribute Based SignCryption Supporting Access Policy Update

In this section, we introduce an overview of PROUD framework in subsection 6.1. Then, we detail different PROUD algorithms and functions in subsection 6.2.

6.1. Overview

PROUD presents a new verifiable privacy-preserving outsourced ABSC scheme that ensures flexible access control, data confidentiality and authentication, while supporting policy updates in cloud assisted IoT applications. PROUD scheme relies on the constant size ABSC scheme proposed by Belguith et al. [11], which has been extended to fulfill all security and privacy requirements introduced in section 2.

Figure 2 presents a detailed workflow of PROUD, while enhancing the different interactions between involved actors. Based on four phases, Figure 2 shows the chronological sequence of seven randomized algorithms and a set of functions. Recall that some phases such as the SYS_INIT, STORAGE and RETRIEVAL are compulsory, while the UPDATE phase is considered as optional. Similarly, some functions are considered as supported features, i.e; optional, such as the *Verify* function that enables a honest data user to find out whether a data content that has been designcrypted with the STES assistance is matching the ciphertext downloaded from CSP or not.

Hereafter, we present the details of PROUD algorithms w.r.t. the main four phases. For ease of presentation, the different notations used in this paper are listed in Table 2.

6.2. PROUD Algorithms

The PROUD construction relies on seven randomized algorithms defined as follows:

- SYS_INIT phase:
  - Setup – CTA sets three groups $(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G})$ of prime order p, an asymmetric bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}$ and a hash function $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_p$. Note that $\mathcal{H}$ is collusion resistant as defined in [43]. Additionally, CTA defines a function τ such that $\tau : \mathbb{U} \rightarrow (\mathbb{Z}/p\mathbb{Z})^*$, where $\mathbb{U}$ is the attribute universe supported by CTA and k is the cardinal of $\mathbb{U}$. For

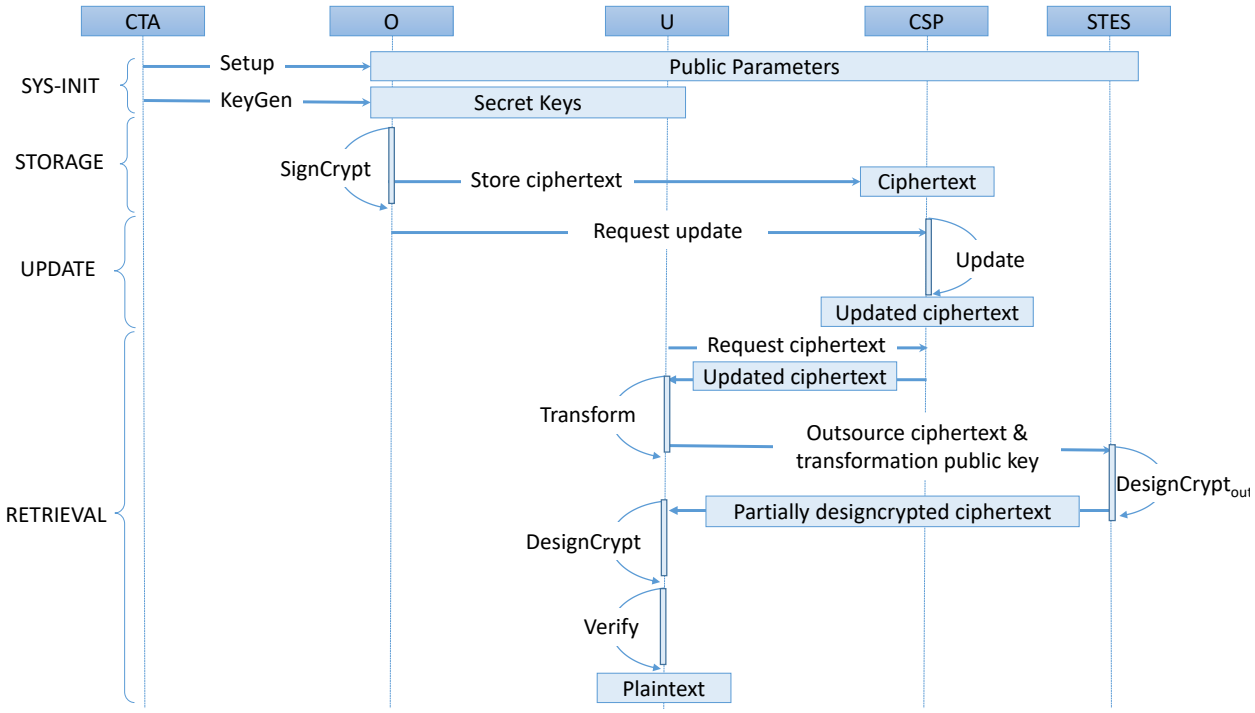

Figure 2: Work Flow of PROUD Scheme.

each attribute $a \in \mathbb{U}$, the encoded attribute values $\tau(a) = x$ are pairwise different. Then, Setup picks two generators g and h of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. It also chooses a set of pairwise different elements of $(\mathbb{Z}/p\mathbb{Z})^*$, $\mathcal{D}_i = \{d_1, \cdots, d_i\}$ where $i \leq k-1$. Finally, it outputs the global public parameters pp defined as follows:

$\mathsf{pp} = \{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}, g, h, \hat{e}, \{h^{\alpha\gamma^j}\}_{\{j=0,\cdots,k\}}, \{u_j = g^{\alpha\gamma^j}\}_{j=1\cdots k}, \hat{e}(g^{\alpha}, h), \tau, \mathcal{H}\}$

The master secret key is defined as $\mathtt{msk} = (\alpha, \gamma)$ where α, γ are two random from values $(\mathbb{Z}/p\mathbb{Z})^*$.

– KeyGen – we denote by A_i the attribute set for an entity where $i = U$ for a data user U and $i = O$ for a data owner O. For a set of attributes $A_i \subset \mathbb{U}$, KeyGen generates the related secret key after selecting a random value $r_i \in (\mathbb{Z}/p\mathbb{Z})^*$ as follows:

$$\begin{aligned} sk_i &= (\{g^{\frac{r_i}{\gamma+\tau(a)}}\}_{a\in A_i}, \{h^{r_i\gamma^j}\}_{j=0,\cdots,k-2}, h^{\frac{r_i-1}{\gamma}}) \\ &= (sk_{i_1}, sk_{i_2}, sk_{i_3}) \end{aligned}$$

- STORAGE phase:

– SignCrypt – let $(t, \mathcal{S}_e)$ and $(t, \mathcal{S}_s)$ be the encryption and signing policies, defined as follows.
$(t, \mathcal{S}_e)$ represents the access policy where $\mathcal{S}_e \subset \mathbb{U}$ and $s = |\mathcal{S}_s| = \mathcal{S}_e|$ such that $1 \leq t \leq |\mathcal{S}_e|$. Note that the access policy is point out by the data owner to define which set of attributes have to be satisfied by requesting data users.
$(t, \mathcal{S}_s)$ represents the signing policy where $\mathcal{S}_s \subset \mathbb{U}$ and $s = |\mathcal{S}_s|$ such that $1 \leq t \leq |\mathcal{S}_s|$. Let A_O be the data owner's sub-set of attribute set where $|A_O \cap \mathcal{S}_s| = t$. Note that the signing policy is already defined by the system.
The SignCrypt algorithm makes use of an aggregate function Aggreg which is introduced and detailed in [39, 40]. Using his secret key sk_O and the Aggreg function, O derives T_1 such as:

$$T_1 = \mathtt{Aggreg}(\{g^{\frac{r_O}{\gamma+\tau(a_i)}}, \tau(a_i)\}_{a_i\in A_O}) = g^{\frac{r_O}{\prod_{a_i\in A_O}(\gamma+\tau(a_i))}}$$

Then, O sets the following polynomial $P_{(A_O,\mathcal{S}_s)}(\gamma)$:

$$P_{(A_O,\mathcal{S}_s)}(\gamma) = \frac{1}{\gamma}\Big(\prod_{a_i\in\mathcal{S}_s\cup D_{k+t-1-s}\setminus A_O}(\gamma+\tau(a_i)) - B_1\Big)$$

Where $B_1 = \prod_{a_i\in\mathcal{S}_s\cup D_{k+t-1-s}\setminus A_O}\tau(a_i)$

Then, using the sk_{O_2}, O computes B_2 such as:

$$B_2 = h^{r_O P_{(A_O,\mathcal{S}_s)}(\gamma)/B_1}$$

In the sequel, O computes $\mathcal{H}(M)$ and generates the signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ defined as:

$$\begin{cases} \sigma_1 = T_1 \cdot g^{\frac{\mathcal{H}(M)}{\prod_{a_i\in A_U} O(\gamma+\tau(a_i))}} \\ \sigma_2 = sk_{O_3} \cdot B_2 \cdot h^{\mathcal{H}(M)P_{(A_O,\mathcal{S}_s)}(\gamma)/B_1} \\ \sigma_3 = h^{\alpha\cdot\mathcal{H}(M)} \end{cases}$$

Table 2: The different notations used in this paper

Notation	Description
CSP	Cloud Service Provider
CTA	Central Trusted Authority
$STES$	Semi-Trusted Edge Server
O	Data Owner
U	User
M	Message
$\mathbb{U}$	The attribute universe
k	The size of the attribute universe $\mathbb{U}$
$\mathbb{M}$	The message universe
pp	Public Parameters
msk	Master Secret Key
a	An attribute
S_s	A signing access policy
S_e	An encrypting access policy
$s = \lvert S_s\rvert = \lvert S_e\rvert$	The size of the signing and encrypting access policies
S_U	A set of attributes belonging to a user U
S_O	A set of attributes belonging to a data owner O
$\mathcal{U}$	A set of attributes to be added/removed to an encrypting access policy
l	The size of $\mathcal{U}$
S'_e	An updated encrypting access policy
t	Threshold
sk_U	Secret key related to a user U
sk_O	Secret key related to a user O
pk_{AA_j}	Public key related to AA_j
tk_U	A transformation key related to user U
tpk_U	A transformation public key
tsk_U	A transformation private key
Σ	The signcrypted message
Σ_{up}	The signcrypted updated message
Σ_{part}	The partially designcrypted message
E_1	An exponentiation overhead in $\mathbb{G}_1$
E_2	An exponentiation overhead in $\mathbb{G}_2$
E	An exponentiation overhead in $\mathbb{G}$
τ_P	The computation overhead of a pairing function $\hat{e}$
$O(M)$	The size of a message M
$\mathcal{H}$	The overhead of a hash function
$\lvert A_U\rvert$	The size of a user's attribute set
$\lvert A_O\rvert$	The size of a user's attribute set
$\lvert s_s\rvert$	The size of a signing access policy
$\lvert s_e\rvert$	The size of encrypting access policy
l	The size of an attribute set to be added or removed from access policy
r	The maximum number of attributes that can be revoked from an access policy
Ω	The size of a ciphertext element in bits
Φ	The size of a user's secret key element in bits

O defines also $r \leq s$ such that r is the maximum number of attributes that can be revoked from the access policy. Note that σ_2 is generated using T_1 that is computed relying on the Aggregate algorithm presented in Section 5.3, and the public elements $\{h^{\alpha\gamma^j}\}_{\{j=0,\cdots,k\}}$ as defined in the $aMSE-CDH$ assumption (*c.f.,* Definition 6). σ_2 is generated using the secret keys sk_{i_2}, sk_{i_3} and the public elements $\{h^{\alpha\gamma^j}\}_{\{j=0,\cdots,k\}}$ while σ_3 is generated using u_0.

Finally, O selects $\kappa \in (\mathbb{Z}/p\mathbb{Z})^*$ and generates E_0, C_1 and C_M. Note that E_0, C_1 are generated using the public parameters $\{u_j\}_{j=1\cdots k}$ and $\hat{e}(g,h)^{\alpha}$. Additionally, $\{h^{\alpha\gamma^j}\}_{j=0,\cdots,2k-1}$ are utilised to generate C_M w.r.t to the equivalence equation defined in the $(\tilde{l},\tilde{m},\tilde{t})-aMSE-CDH$ definition introduced by Delerablee et al. [39, 40]. E_0, C_1 and C_M are detailed as follows:

$$\begin{cases} E_0 = h^{\kappa\alpha\cdot\prod_{a_i\in\mathcal{S}_e}(\gamma+\tau(a_i))}, E_1 = E_0^{\gamma},\cdots,E_{k-s} = E_{k-s-1}^{\gamma} \\ C_1 = u_1^{-\kappa},\cdots,C_{r+1} = u_{r+1}^{-\kappa} \\ C_M = \hat{e}(g,h)^{\alpha\cdot\kappa}\cdot\hat{e}(g,h)^{\alpha\cdot\mathcal{H}(M)}\cdot M = K\cdot M \end{cases}$$

O finally generates the signcrypted message $\Sigma = (C_1,\cdots,C_{r+1},E_0,\cdots,E_{k-s},C_M,\sigma_1,\sigma_2,\sigma_3, P_{(A_O,\mathcal{S}_s)}(\gamma),B_1)$.

- UPDATE phase:
  - Update – the Update algorithm first checks the operation indicator ind. Then, if ind = add, it proceeds as defined in (i), otherwise if ind = revoke it executes (ii):

    (i) – given a signcrypted message Σ encrypted w.r.t. to $\mathcal{S}_e$ and $\mathcal{U} = \{a'_1,\cdots,a'_l\}$ a new set of attributes such that $\mathcal{U}\cap\mathcal{S}_e = \emptyset$, CSP has to add elements of $\mathcal{U}$ to $\mathcal{S}_e$. To this end, CSP performs the following steps:
    Let $F(x)$ be the polynomial in x defined as $F(x) = \prod_{a_i\in\mathcal{U}}(x+\tau(a_i)) = f_l x^l + f_{l-1}x^{l-l} + \cdots + f_0$
    Afterwards, Update generates $E_{0_{up}} = E_0^{F(\gamma)} = \prod_{i=0}^{l} E_i^{f_i}$.
    The updated ciphertext is defined as $\Sigma_{up} = (E_{0_{up}}, C_1, C_M, \sigma_1, \sigma_2, \sigma_3, P_{(A_O,\mathcal{S}_s)}(\gamma), B_1))$ w.r.t. $\mathcal{S}'_e$, the updated set of encrypting attributes $\mathcal{S}'_e = \mathcal{S}_e \cup \mathcal{U}$.

    (ii) – given a ciphertext Σ signcrypted w.r.t. $\mathcal{S}_e$ and a revocation attribute set $\mathcal{U} = \{a'_1,\cdots,a'_l\} \subseteq \mathcal{S}_e$ where $l \leq r$, CSP updates Σ as follows:
    Let $F(x)$ be the polynomial in x as $F(x) = \frac{1}{\prod_{a_i\in U}\tau(a_i)}\prod_{a_i\in\mathcal{U}}(x+\tau(a_i)) = f_l x^l + f_{l-1}x^{l-1} + \cdots + f_0$
    Then, the algorithm computes $\Sigma_{up} = (E_{0_{up}}, C_{1_{up}}, C_{M_{up}}, \sigma_1, \sigma_2, \sigma_3, P_{(A_O,\mathcal{S}_s)}(\gamma), B_1))$ where $E_{0_{up}}, C_{1_{up}}, C_{M_{up}}$ are computed as follows:

$$\begin{cases} E_{0_{up}} = E_0^{\frac{1}{\prod_{a_i\in\mathcal{U}}\tau(a_i)}} = h^{\kappa\cdot\prod_{a_i\in\mathcal{S}_e\setminus\mathcal{U}}(\gamma+\tau(a_i))F(\gamma)} \\ C_{1_{up}} = \prod_{l=1}^{l+1} C_i^{f_{i-1}} = u_1^{-\kappa F(\gamma)} \\ C_{M_{up}} = C_M\cdot\hat{e}(\prod_{i=1}^{l} C_i^{-f_i}, h) = K'\cdot M \end{cases}$$

- RETRIEVAL phase:
  - Transform – the user U executes Transform. U who has a set of attributes A_U and the related secret key sk_U, selects $z \in \mathbb{Z}_N^*$ and derives the transformation keys $tk_U = (tpk_U, tsk_U)$, where tpk_U and tsk_U are computed as follows:

$$\begin{cases} tpk_U = (sk_{O_1}^{\frac{1}{z}}, sk_{O_2}^{\frac{1}{z}}, sk_{O_3}^{\frac{1}{z}}) \\ tsk_U = z \end{cases}$$

    Therefore, tpk_U and tsk_U are defined as:

$$\begin{cases} tpk_U = (\{g^{\frac{r_U}{z(\gamma+\tau(a_i))}}\}_{a_i\in A_U}, \{h^{\frac{r_U\gamma^j}{z}}\}_{j=0,\cdots,k-2}, h^{\frac{r_U-1}{z\gamma}}) \\ tsk_U = z \end{cases}$$

Finally, the user outsources the signcrypted text Σ_{up} as downloaded from CSP while only randomising $(\sigma_3)^{\frac{1}{z}}$ along with the transformation public key tpk_U to the STES.

– $\mathsf{DesignCrypt}_{\mathsf{out}}$ – First, the STES verifies the data owner O signature by checking the correctness of the following equation:

$$\begin{aligned}\text{Ⓢ} &= \hat{e}(u_0^{-1},\sigma_2)\cdot\hat{e}(\sigma_1^{\frac{1}{B_1}},h^{\alpha\cdot\prod_{a_i\in S_s\cup D_{k+t-1-s}}(\gamma+\tau(a_i))})\\ &\quad\cdot\hat{e}(g^{\alpha},h)^{\mathcal{H}(M)\cdot(1-U_1-\frac{1}{U_1})}\\ &= \hat{e}(g^{\alpha},h)\end{aligned}$$

where $U_1=\prod_{a_i\in S_s\cup D_{k+t-1-s}\setminus A_U}\frac{\gamma+\tau(a_i)}{\tau(a_i)}$.

Afterwards, STES aggregates the user's secret key for all $a_i\in A_U$ using the aggregate function $\mathtt{Aggreg}$ [39, 40] such as:

$$A_2=\mathtt{Aggreg}(\{g^{\frac{r_U}{z(\gamma+\tau(a_i))}},\tau(a_i)\}_{a_i\in A_U})=g^{\frac{r_U}{z\prod_{a_i\in A_U}(\gamma+\tau(a_i))}} \quad (3)$$

Afterwards, STES uses the aggregated transformed secret key A_2 and tpk_{U_2} to compute K'' which can be retrieved based on two cases w.r.t. the ind operator value, such that:

* **Case 1:** in the case of adding attributes to the access policy, STES deduces the deciphering key K'' such as:

$$\begin{aligned}K'' &= \hat{e}(C_1,tpk_{U_3})\cdot\hat{e}(g,(\sigma_3)^{\frac{1}{z}})\hat{e}(A_2,E_0)\\ &= \hat{e}(g,h)^{\frac{\alpha\cdot\kappa}{z}}\cdot\hat{e}(g,h)^{\frac{\alpha\cdot\mathcal{H}(M)}{z}}\\ &= K^{\frac{1}{z}}\end{aligned}$$

Finally, the STES returns K'' to the user. Note that if the Update has not been executed, i.e., the access policy has not been updated, the designcryption process follows **Case 1**.

* **Case 2:** in the case of revoking attributes from the access policy, STES deduces the deciphering key K'' such as:

$$\begin{aligned}K'' &= \hat{e}(C_1',tpk_{U_3})\cdot\hat{e}(g,\sigma_3)\cdot\\ &\quad\hat{e}(g,h)^{\kappa\cdot r_{U_2}\cdot\alpha}\\ &= \hat{e}(g,h)^{\frac{\alpha\cdot\kappa\cdot F(\gamma)}{z}}\cdot\hat{e}(g,h)^{\frac{\alpha\cdot\mathcal{H}(M)}{z}}\\ &= K'^{\frac{1}{z}}\end{aligned}$$

Finally, the STES returns k'' to the user.

– $\mathsf{DesignCrypt}$ – This algorithm includes two steps. The first step, denoted by (i), enables U to retrieve the plaintext, while the second step (ii) permits to verify the correctness of the partially designcrypted message received from STES:

(i) First, based on the partially decrypted ciphertext k'', the user executes Equation 4 while performing only one exponentiation without calculating any pairing functions to recover the message. M can be retrieved based on two cases w.r.t. the ind operator value, such that:

* **Case 1:** in the case of adding attributes to the access policy:

$$\begin{aligned}M &= \frac{C_M}{(K'')^{tsk}}\\ &= \frac{K\cdot M}{(K^{\frac{1}{z}})^z}\\ &= \frac{K\cdot M}{K}\end{aligned}$$

* **Case 2:** in the case of revoking attributes from the access policy:

$$\begin{aligned}M &= \frac{C_M'}{(K')^{tsk}}\\ &= \frac{K'\cdot M}{(K'^{\frac{1}{z}})^z}\\ &= \frac{K'\cdot M}{K'}\end{aligned}$$

(ii) Using the retrieved message M, U computes $\mathcal{H}(M)$. Afterwards, U calculates $V=h^{\alpha\cdot\mathcal{H}(M)}$.
To verify the correctness of the retrieved message M, i.e., the partially designcrypted message received from STES K'' is correct, U compares V against σ_3 which was already downloaded from CSP.
If $V=\sigma_3$, then STES has executed $\mathsf{DesignCrypt}_{\mathsf{out}}$ correctly, otherwise, U declines the received partially designcrypted ciphertext K''.

7. Security Analysis

The security of PROUD relies on the following Theorems.

Theorem 1. Correctness *PROUD is correct if for all* $(\mathtt{pp},\mathtt{msk})\leftarrow\mathsf{Setup}(\xi)$*, all attributes' sets* $A_i\in\mathbb{U}$ *where* $i\in\{U,O\}$*, all private keys* $sk_i\leftarrow\mathsf{KeyGen}(\mathtt{pp},\mathtt{msk},i,A_i)$*, all access policies* $\{\Psi_e=(t,S_e),\Psi_s=(t,S_s)\}$ *such as* t *is the threshold value,* $\Psi_e(A_U)=1$ *and* $\Psi_s(A_O)=1$*, all messages* $M\in\mathbb{M}$*, all signcrypted messages* $\Sigma\leftarrow\mathsf{SignCrypt}(\mathtt{pp},sk_O,A_O,\Psi_e,\Psi_s,M)$*, all updated signcrypted messages* $\Sigma_{up}\leftarrow\mathsf{Update}(\mathtt{pp},\Sigma,\mathsf{ind},\mathcal{U})$*, all transformation keys* $\leftarrow\mathsf{Transform}(\mathtt{pp},sk_U,\Psi_e)$*, all partially designcrypted messages* $\Sigma_{part}\leftarrow\mathsf{DesignCrypt}_{\mathsf{out}}(\mathtt{pp},tpk_U,\Psi_e,\Psi_s,\Sigma)$ *, we have to obtain* $M\leftarrow\mathsf{DesignCrypt}(tsk_U,\Sigma_{part})$.

Theorem 2. Confidentiality *The proposed PROUD scheme is indistinguishable against replayable chosen ciphertext attacks, w.r.t. the a-MSE-CDH assumption.*

Theorem 3. Unforgeability *PROUD is unforgeable against chosen-message attacks, w.r.t. CDH and a-MSE-CDH assumptions.*

Theorem 4. Privacy *The proposed PROUD scheme is computationally private w.r.t. the CDH assumptions.*

Theorem 5. Verifiability *If $\mathcal{H}$ is a collision-resistant hash function, then the proposed PROUD scheme is verifiable against lazy servers.*

PROUD relies on the Computational Diffie Hellman Assumption (CDH) and the augmented multi-sequence of exponents computational Diffie-Hellman ($(\tilde{l},\tilde{m},\tilde{t})$-aMSE-CDH) assumptions (*cf.* Definitions 6 and 5). More details of these assumptions can be found in papers [39], [40], [41] and [11].

Here-after, we start by proving the correctness of the proposed scheme. Afterwards, we introduce the security proofs related to security games presented in Section 4.2.

7.1. Correctness

In this subsection, we show the correctness of PROUD w.r.t. Theorem 1, while detailing the update process in subsection 7.1.1 and the designcryption algorithms in subsection 7.1.2.

7.1.1. Update Correctness

The Update algorithm first checks the operation indicator ind. Then, if ind = add, it proceeds as (i), otherwise if ind = revoke it executes (ii):

(i) – given a signcrypted message Σ encrypted w.r.t. to $\mathcal{S}_e$ and $\mathcal{U}=\{a'_1,\cdots,a'_l\}$.

Let $F(x)$ be the polynomial in x defined as $F(x)=\prod_{a_i\in\mathcal{U}}(x+\tau(a_i))=f_l x^l+f_{l-1}x^{l-1}+\cdots+f_0$

Then, Update calculates $E_{0_{up}}$ such that:

$$\begin{aligned}
E_{0_{up}} &= E_0^{F(\gamma)}\\
&= h^{\kappa\alpha\cdot\prod_{a_i\in\mathcal{S}_e}(\gamma+\tau(a_i))(f_l\gamma^l+f_{l-1}\gamma^{l-1}+\cdots+f_0)}\\
&= h^{\kappa\alpha(f_l\gamma^l)\cdot\prod_{a_i\in\mathcal{S}_e}(\gamma+\tau(a_i))}\cdot h^{\kappa\alpha f_{l-1}\gamma^{l-1}\cdot\prod_{a_i\in\mathcal{S}_e}(\gamma+\tau(a_i))}\\
&\quad\cdots h^{\kappa\alpha f_0)\cdot\prod_{a_i\in\mathcal{S}_e}(\gamma+\tau(a_i))}\\
&= E_l^{\gamma^l}\cdot E_{l-1}^{\gamma^{l-l}}\cdots E_0^{f_0}\\
&= \prod_{i=0}^{l}E_i^{f_i}
\end{aligned}$$

The updated ciphertext Σ_{up} is defined as $\Sigma_{up}=(E_{0_{up}},C_1,C_M)$ w.r.t. $\mathcal{S}'_e$, the new set of encrypting attributes defined as $\mathcal{S}'_e=\mathcal{S}_e\cup\mathcal{U}$.

(ii) – given a ciphertext Σ encrypted w.r.t. a set of attributes $\mathcal{S}_e$ and a revocation attribute set $\mathcal{U}=\{a'_1,\cdots,a'_l\}\subseteq\mathcal{S}_e$ where $l\leq r$, the server updates the signcrypted message Σ as follows:

Let $F(x)$ be the polynomial in x as
$F(x)=\frac{1}{\prod_{a_i\in U}\tau(a_i)}\prod_{a_i\in\mathcal{U}}(x+\tau(a_i))=f_l x^l+f_{l-1}x^{l-1}+\cdots+f_0$

Then, Update calculates Σ_{up} as follows:

$$\begin{aligned}
&\begin{cases}
E_{0_{up}}=E_0^{\frac{1}{\prod_{a_i\in\mathcal{U}}\tau(a_i)}}\\
C_{1_{up}}=\prod_{l=1}^{l+1}C_i^{f_{i-1}}\\
C_{M_{up}}=C_M\cdot\hat{e}(\prod_{i=1}^{l}C_i^{-f_i},h)
\end{cases}\\
=&\begin{cases}
E_{0_{up}}=h^{\frac{\kappa\alpha\cdot\prod_{a_i\in\mathcal{S}_e}(\gamma+\tau(a_i))}{\prod_{a_i\in\mathcal{U}}\tau(a_i)}}\\
C_{1_{up}}=g^{-\alpha\kappa\sum_{i=1}^{l+1}\gamma^{i-1}f_{i-1}}\\
C_{M_{up}}=M\cdot\hat{e}(g,h)^{\alpha\cdot\mathcal{H}(M)}\cdot\hat{e}(g,h)^{\alpha\kappa\sum_{i=0}^{l}f_i\gamma^i}
\end{cases}\\
=&\begin{cases}
E_{0_{up}}=h^{\frac{\kappa\alpha\cdot\prod_{a_i\in\mathcal{U}}(\gamma+\tau(a_i))}{\prod_{a_i\in\mathcal{U}}\tau(a_i)}\prod_{a_i\in\mathcal{S}_e\setminus\mathcal{U}}(\gamma+\tau(a_i))}\\
C_{1_{up}}=u_1^{-\kappa F(\gamma)}\\
C_{M_{up}}=M\cdot\hat{e}(g,h)^{\alpha\cdot\mathcal{H}(M)}\cdot\hat{e}(g,h)^{\alpha\kappa F(\gamma)}
\end{cases}\\
=&\begin{cases}
E_{0_{up}}=h^{\kappa\alpha\cdot\prod_{a_i\in\mathcal{S}_e\setminus\mathcal{U}}(\gamma+\tau(a_i))F(\gamma)}\\
C_{1_{up}}=u_1^{-\kappa F(\gamma)}\\
C_{M_{up}}=M\cdot K'
\end{cases}
\end{aligned}$$

7.1.2. Designcryption Correctness

We assume a data user U having an attribute set A_U which satisfies the encryption and signing policies (t,S_s) and (t,S_e). In the following, we prove the correctness of PROUD, i.e., U can retrieve the plaintext message using his attributes and the related secret keys.

First, U derives the transformation keys $tk_U=(tsk_U,tpk_U)$. Afterwards, she forwards the public transformation key tpk_U to STES. This latter first proceeds by verifying the data owner O signature as follows:

$$\begin{aligned}
\text{Ⓢ} &= \hat{e}(g^{-\alpha\cdot\gamma},h^{\frac{r_{E_S}^{-1}}{\gamma}}\cdot h^{\frac{(r_{E_S}+\mathcal{H}(M))\cdot P_{(A_S,S)}(\gamma)}{\prod_{a_i\in S\cup D_{n+t-1-s}\setminus A_S}(\tau(a_i)+\gamma)}})\\
&\quad\cdot\hat{e}(g^{\alpha},h)^{\mathcal{H}(M)\cdot(1-U_1-\frac{1}{U_1})}\cdot\\
&\quad\hat{e}(g^{\frac{(r_{E_S}+\mathcal{H}(M))}{\prod_{a_i\in A_S}(\gamma+\tau(a_i))\cdot\prod_{a_i\in S\cup D_{n+t-1-s}\setminus A_S}\tau(a_i)}},\\
&\quad h^{\alpha\cdot\prod_{a_i\in S\cup D_{n+t-1-s}}(\tau(a_i)+\gamma)})\\
&= \hat{e}(g^{-\alpha\cdot\gamma},h^{\frac{r_{E_S}^{-1}}{\gamma}}h^{r_{E_S}\cdot\frac{P_{(A_S,S)}(\gamma)}{\prod_{a_i\in S\cup D_{n+t-1-s}\setminus A_S}(\tau(a_i)+\gamma)}})\\
&\quad\cdot\hat{e}(g^{-\alpha\cdot\gamma},h^{\mathcal{H}(M)\cdot\frac{P_{(A_S,S)}(\gamma)}{\prod_{a_i\in S\cup D_{n+t-1-s}\setminus A_S}\tau(a_i)}})\\
&\quad\cdot\hat{e}(g^{\alpha},h)^{\mathcal{H}(M)}\cdot\hat{e}(g^{\alpha},h)^{-\mathcal{H}(M)\cdot U_1}\cdot\hat{e}(g^{\alpha},h)^{\frac{-\mathcal{H}(M)}{U_1}}\\
&\quad\cdot\hat{e}(g^{\frac{r_{E_S}}{\prod_{a_i\in A_S}(\gamma+\tau(a_i))\cdot\prod_{a_i\in S\cup D_{n+t-1-s}\setminus A_S}\tau(a_i)}},\\
&\quad h^{\alpha\cdot\prod_{a_i\in S\cup D_{n+t-1-s}}(\tau(a_i)+\gamma)}).
\end{aligned}$$

$$
\begin{aligned}
& \hat{e}(g^{\frac{\mathcal{H}(M)}{\prod_{a\in A_S}(\gamma+\tau(a_i))\cdot\prod_{a_i\in S\cup D_{n+t-1-s}\setminus A_S}\tau(a_i)}}, \\
& h^{\alpha\cdot\prod_{a_i\in S\cup D_{n+t-1-s}}(\tau(a_i)+\gamma)}) \\
= \;& \hat{e}(g^{\alpha},h)^{\mathcal{H}(M)}\cdot\hat{e}(g^{\alpha},h)^{-\mathcal{H}(M)\cdot\prod_{a_i\in S_s\cup D_{k+t-1-s}\setminus A_U}\frac{\gamma+\tau(a_i)}{\tau(a_i)}} \\
& \cdot\hat{e}(g^{\alpha},h)^{\frac{-\mathcal{H}(M)}{\prod_{a_i\in S_s\cup D_{k+t-1-s}\setminus A_U}\frac{\gamma+\tau(a_i)}{\tau(a_i)}}} \\
& \hat{e}(g^{\alpha},h)\cdot\hat{e}(g^{\alpha},h)^{\mathcal{H}(M)\cdot\prod_{a_i\in S\cup D_{n+t-1-s}\setminus A_S}\frac{\gamma+\tau(a_i)}{\tau(a_i)}} \\
& \cdot\hat{e}(g^{\alpha},h)^{-\mathcal{H}(M)\cdot\prod_{a_i\in S\cup D_{n+t-1-s}\setminus A_S}\frac{\tau(a_i)}{\tau(a_i)}} \\
& \cdot\hat{e}(g^{\alpha},h)^{\frac{\mathcal{H}(M)\cdot\prod_{a_i\in S\cup D_{n+t-1-s}}(\gamma+\tau(a_i))}{\prod_{a_i\in A_S}(\gamma+\tau(a_i))\cdot\prod_{a_i\in S\cup D_{n+t-1-s}\setminus A_S}\tau(a_i)}} \\
= \;& \hat{e}(g^{\alpha},h)
\end{aligned}
$$

Afterwards, STES uses the aggregated transformed secret key A_2 and tpk_{U_3} to derive K' which can be retrieved based on two cases w.r.t. the ind operator value, such that:

- **Case 1:** in the case of adding attributes to the access policy, STES deduces the deciphering key K'' such as:

$$
\begin{aligned}
K'' &= \hat{e}(C_1,tpk_{U_3})\cdot\hat{e}(g,(\sigma_3)^{\frac{1}{z}})\hat{e}(A_2,E_0) \\
&= \hat{e}(g^{-\alpha\gamma\kappa},h^{\frac{r_U-1}{z\gamma}})\cdot\hat{e}(g,h^{\frac{\alpha\mathcal{H}(M)}{z}}) \\
&\quad \cdot\hat{e}(g,h)^{\frac{\kappa\cdot r_U\cdot\alpha}{z}} \\
&= \hat{e}(g,h)^{-\alpha\gamma\kappa\frac{r_U-1}{z\gamma}}\cdot\hat{e}(g,h)^{\frac{\alpha\mathcal{H}(M)}{z}} \\
&\quad \cdot\hat{e}(g,h)^{\frac{\kappa\cdot r_U\cdot\alpha}{z}} \\
&= \hat{e}(g,h)^{\frac{-\alpha\kappa r_U}{z}}\cdot\hat{e}(g,h)^{\frac{r_U\alpha\kappa}{z}}\hat{e}(g,h)^{\frac{\alpha\mathcal{H}(M)}{z}}\cdot \\
&\quad \cdot\hat{e}(g,h)^{\frac{\kappa\cdot r_U\cdot\alpha}{z}} \\
&= \hat{e}(g,h)^{\frac{\alpha\cdot\kappa}{z}}\cdot\hat{e}(g,h)^{\frac{\alpha\cdot\mathcal{H}(M)}{z}} \\
&= K^{\frac{1}{z}}
\end{aligned}
$$

Finally, the STES returns K'' to the user.

- **Case 2:** in the case of revoking attributes from the access policy, STES deduces the deciphering key K'' such as:

$$
\begin{aligned}
K'' &= \hat{e}(C_{1_{up}},tpk_{U_3})\cdot\hat{e}(g,(\sigma_3)^{\frac{1}{z}})\hat{e}(A_2,E_{0_{up}}) \\
&= \hat{e}(g^{-\alpha\gamma\kappa F(\gamma)},h^{\frac{r_U-1}{z\gamma}})\cdot\hat{e}(g,h^{\alpha\cdot\mathcal{H}(M)}) \\
&\quad \cdot\hat{e}(g,h)^{F(\gamma)\kappa\cdot r_U\cdot\alpha} \\
&= \hat{e}(g,h)^{-\alpha\gamma\kappa F(\gamma)\frac{r_U-1}{z\gamma}}\cdot\hat{e}(g,h)^{\alpha\cdot\mathcal{H}(M)} \\
&\quad \cdot\hat{e}(g,h)^{F(\gamma)\kappa\cdot r_U\cdot\alpha} \\
&= \hat{e}(g,h)^{\frac{-F(\gamma)\alpha\kappa r_U}{z}}\cdot\hat{e}(g,h)^{\frac{F(\gamma)\alpha\kappa r_U}{z}} \\
&\quad \hat{e}(g,h)^{\frac{\alpha\mathcal{H}(M)}{z}}\cdot\hat{e}(g,h)^{\frac{F(\gamma)\kappa\cdot r_U\cdot\alpha}{z}} \\
&= \hat{e}(g,h)^{\frac{F(\gamma)\alpha\cdot\kappa}{z}}\cdot\hat{e}(g,h)^{\frac{\alpha\cdot\mathcal{H}(M)}{z}} \\
&= K'^{\frac{1}{z}}
\end{aligned}
$$

Finally, the STES returns k'' to the user.

7.2. Confidentiality

The following proof shows that PROUD is indistinguishable against replayable chosen ciphertext attacks (IND-RCCA2) w.r.t. Theorem 2.

Proof. The Exp^{conf} security game, defined in Section 4.2.1, captures the behavior of a non-authorised entity. That is, the adversary $\mathcal{A}$ attempts to distinguish between two signcrypted messages generated relying on the signcryption algorithm conducted by the challenger $\mathcal{C}$.

As detailed in Section 6.1, the PROUD scheme relies on the [11] construction, was already proved to be IND-CCA2 secure. As such, we prove that if the ABSC scheme proposed in [11] is IND-CCA2 secure, then, our PROUD scheme is IND-RCCA2 secure such that $Adv_{\mathcal{A}}[Exp^{conf}] \leq Adv_{\mathcal{A}}[Exp^{ABSC}]$, w.r.t to Definition 1.

As PROUD introduces two additional features, i.e; update and outsourced-decryption, it is necessary to consider the extra-knowledge of the adversary, inferred while performing these algorithms. To this end, we define an adversary $\mathcal{A}$ running the Exp^{conf} security game with an entity $\mathcal{B}$. This entity $\mathcal{B}$ is also running the Belguith et al's [11] CCA2-security game (ABSC-Game) with a challenger $\mathcal{C}$. The aim of this proof is to demonstrate that the advantage of the adversary $\mathcal{A}$ to succeed in the Exp^{conf} game is smaller than the advantage of the entity $\mathcal{B}$ to win ABSC-Game.

In the following, we specify the interactions carried between $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$. During the intialisation phase, $\mathcal{A}$ first defines a set of signcryption attributes S^*, which is then shared with $\mathcal{C}$ and $\mathcal{B}$. This latter derives and broadcasts the public parameters pp.

In the second phase, the interaction is mainly carried between $\mathcal{C}$ and $\mathcal{B}$, with $\mathcal{A}$ running the following steps and algorithms, as specified in the Exp^{conf} game. That is, $\mathcal{A}$ attempts to infer knowledge about the designcryption process by querying the execution of the KeyGen, Transform and DesignCrypt

algorithms, relying on **Private Key Query**, **Transformation Key Query**, **DesignCryption Query**, respectively.

Phase 1 Queries – $\mathcal{C}$ sets an empty table T. Then, for each session i, $\mathcal{A}$ is allowed to issue **Private Key Query**, **SignCryption Query** and **DesignCryption Query** queries as follows:

- **Private Key Query** – when $\mathcal{A}$ queries a signcryption attribute set $A_{\mathcal{A},i}$ w.r.t. t_i where $|A_{\mathcal{A},i} \cap S_e^*| < t_i$ and $|A_{\mathcal{A},i} \cap S_s^*| < t_i$, $\mathcal{C}$ performs $\mathsf{KeyGen}(\mathtt{PP}, \mathtt{msk}, \mathcal{C}, A_{\mathcal{C},i})$ and returns the generated secret key $sk_{\mathcal{A},i} = (\{g^{\frac{r_{\mathcal{A}}}{\gamma+\tau(a_i)}}\}_{a_i \in A_{\mathcal{A}}}, \{h^{r_{\mathcal{A}}\gamma^j}\}_{j=0,\cdots,k-2}, h^{\frac{r_{\mathcal{A}}-1}{\gamma}})$ to $\mathcal{A}$.
- **Transformation Key Query** – $\mathcal{A}$ requests the transformation key $tk_{\mathcal{A},i}$, associated to a set of attributes $A_{\mathcal{A},i}$. $\mathcal{C}$ seeks the entry $(A_{\mathcal{A},i}, sk_{\mathcal{A},i}, tk_{\mathcal{A},i})$ in table T. It returns the transformation key if it exists in table T, otherwise $\mathcal{C}$ executes the $\mathsf{Transform}$ to derive $tpk_{\mathcal{A},i} = (\{g^{\frac{r_{\mathcal{A}}}{z(\gamma+\tau(a_i))}}\}_{a_i \in A_{\mathcal{A}}}, \{h^{\frac{r_{\mathcal{A}}\gamma^j}{z}}\}_{j=0,\cdots,k-2}, h^{\frac{r_{\mathcal{A}}-1}{z\gamma}})$ and $tsk_{\mathcal{A},i} = z$ and forwards them to $\mathcal{A}$.
- **SignCryption Query** – $\mathcal{A}$ requests the signcryption of a message M_i w.r.t t_i, while considering the access policies Ψ_e, Ψ_s. $\mathcal{C}$ performs KeyGen and derives $sk_{\mathcal{C},i} = \mathsf{KeyGen}(\mathtt{PP}, \mathtt{msk}, \mathcal{C}, A_{\mathcal{C},i})$, such that $|A_{\mathcal{C},i} \cap S_e^*| \geq t_i$ and $|A_{\mathcal{C},i} \cap S_s^*| \geq t_i$. Thus, $\mathcal{C}$ executes $\mathsf{SignCrypt}(\mathtt{pp}, sk_{\mathcal{C}}, A_{\mathcal{C},i}, \Psi_e = (t, S_e), \Psi_s = (t, S_s), M)$ and returns a ciphertext Σ_i such that:

$$\Sigma_i = \begin{cases} B_1 = \prod_{a_i \in S_s \cup D_{k+t-1-s} \setminus A_{\mathcal{C}}} \tau(a_i) \\ P_{(A_{\mathcal{C}}, S_s)}(\gamma) = \frac{1}{\gamma}(\prod_{a_i \in S_s \cup D_{k+t-1-s} \setminus A_{\mathcal{C}}} (\gamma + \tau(a_i)) - B_1) \\ \sigma_1 = g^{\frac{r_O}{\prod_{a_i \in A_{\mathcal{C}}} (\gamma+\tau(a))}} \cdot g^{\frac{\mathcal{H}(M)}{\prod_{a_i \in A_{\mathcal{C}}} O(\gamma+\tau(a_i))}} \\ \sigma_2 = sk_{\mathcal{C}_3} \cdot h^{r_O P_{(A_{\mathcal{C}}, S_s)}(\gamma)/B_1} \cdot h^{\mathcal{H}(M) P_{(A_{\mathcal{C}}, S_s)}(\gamma)/B_1} \\ \sigma_3 = h^{\alpha \cdot \mathcal{H}(M)} \\ E_0 = h^{\kappa\alpha \cdot \prod_{a_i \in S_e} (\gamma+\tau(a_i))}, E_1 = E_0^{\gamma}, \cdots, E_{k-s} = E_{k-s-1}^{\gamma} \\ C_1 = u_1^{-\kappa}, \cdots, C_{r+1} = u_{r+1}^{-\kappa} \\ C_M = \hat{e}(g,h)^{\alpha \cdot \kappa} \cdot \hat{e}(g,h)^{\alpha \cdot \mathcal{H}(M)} \cdot M = K \cdot M \end{cases}$$

- **DesignCryption Query** – $\mathcal{A}$ asks $\mathcal{C}$ to designcryp a ciphertext Σ_i w.r.t t_i and a signcryption attribute set $A_{\mathcal{A},i}$. $\mathcal{C}$ executes the $\mathsf{KeyGen} = (\mathtt{pp}, msk, \mathcal{A}, A_{\mathcal{A},i})$ to derive $sk_{\mathcal{A},i} = \{g^{\frac{r_{\mathcal{A}}}{\gamma+\tau(a)}}\}_{a \in A_{\mathcal{A}}}, \{h^{r_{\mathcal{A}}\gamma^j}\}_{j=0,\cdots,k-2}, h^{\frac{r_{\mathcal{A}}-1}{\gamma}})$, such that $|A_{\mathcal{C},i} \cap S_e^*| \geq t_i$ and $|A_{\mathcal{C},i} \cap S_s^*| \geq t_i$. Finally, $\mathcal{C}$ runs the $\mathsf{DesignCrypt}(\mathtt{pp}, sk_{\mathcal{A},i}, (t, S_e), (t, S_s), \Sigma_i)$ algorithm[4] that outputs a message M_i or a reject symbol $\perp$.

The **Challenge Phase** consists of selecting two messages by $\mathcal{A}$. This latter randomly picks ${M_0}^*$ and ${M_1}^*$ and an attribute set $\mathbb{U}^*$ where $\mathbb{U}^* \cap {\mathcal{S}_e}^* = \emptyset$ if $\mathsf{ind} = \mathsf{add}$ or $\mathbb{U}^* \subset {\mathcal{S}_e}^*$ if $\mathsf{ind} = \mathsf{revoke}$. Afterwards, $\mathcal{B}$ selects $A_{\mathcal{B}}$ such that $A_{\mathcal{B}} \subseteq {\mathcal{S}'_e}^*$, such as ${\mathcal{S}_e}'^* = {\mathcal{S}_e}^* \setminus \mathcal{U}^*$ for $\mathsf{ind} = \mathsf{add}$ or ${\mathcal{S}_e}'^* = {\mathcal{S}_e}^* \cup \mathcal{U}^*$ for $\mathsf{ind} = \mathsf{revoke}$.
Subsequently, $\mathcal{B}$ transmits the access policy $A_{\mathcal{B}}$ and the two messages M_0 and M_1, defined by $\mathcal{A}$ to $\mathcal{C}$. Then, the challenger chooses a random bit b from $\{0,1\}$ with ${\mathcal{S}_e}'^* = {\mathcal{S}_e}^* \setminus \mathcal{U}^*$ for $\mathsf{ind} = \mathsf{add}$ or ${\mathcal{S}'_e}^* = {\mathcal{S}_e}^* \cup \mathcal{U}^*$ for $\mathsf{ind} = \mathsf{revoke}$ and computes ${\Sigma_b}^*$ using $\mathsf{SignCrypt}$.

The challenger $\mathcal{C}$ sends ${\Sigma_b}^*$ to the adversary if $\mathcal{U} = \emptyset$, otherwise $\Sigma^*_{b_{up}} = \mathsf{Update}(\mathtt{pp}, {\Sigma_b}^*, \mathsf{ind}, \mathcal{U}^*)$.

Phase 2 Queries – the adversary is allowed to query a polynomial number of queries as in **Phase 1 Queries**, except that he is unable to request the designcryption of the received challenge signcrypted message ${\Sigma_b}^*$ or $\Sigma^*_{b_{up}}$.

Hereafter, two cases are defined w.r.t. the ind operator value, chosen by $\mathcal{C}$ to signcrypt the challenging message such that:

- **Case A** – it corresponds to attributes' addition, such that $\mathcal{C}$ sets ${\mathcal{S}_e}'^* = {\mathcal{S}_e}^* \cup \mathcal{U}^*$ and generates a signcrypted message $\Sigma_{up,b}$. In this case, we first show that how a challenge signcrypted message should be produced in the following:.

$$\Sigma_{up,b} = \begin{cases} B_1 = \prod_{a_i \in S_s \cup D_{k+t-1-s} \setminus A_{\mathcal{C}}} \tau(a_i) \\ P_{(A_{\mathcal{C}}, S_s)}(\gamma) = \frac{1}{\gamma}(\prod_{a_i \in S_s \cup D_{k+t-1-s} \setminus A_{\mathcal{C}}} (\gamma + \tau(a_i)) - B_1) \\ \sigma_1 = g^{\frac{r_{\mathcal{C}}}{\prod_{a_i \in A_{\mathcal{C}}} (\gamma+\tau(a_i))}} \cdot g^{\frac{\mathcal{H}(M_b)}{\prod_{a_i \in A_{\mathcal{C}}} (\gamma+\tau(a_i))}} \\ \sigma_2 = sk_{\mathcal{C}_3} \cdot h^{r_{\mathcal{C}} P_{(A_{\mathcal{C}}, S_s)}(\gamma)/B_1} \cdot h^{\mathcal{H}(M_b) P_{(A_{\mathcal{C}}, S_s)}(\gamma)/B_1} \\ \sigma_3 = h^{\alpha \cdot \mathcal{H}(M_b)} \\ E_{0_{up}} = \prod_{i=0}^{l} E_i^{f_i} \\ C_1 = u_1^{-\kappa} \\ C_M = \hat{e}(g,h)^{\alpha \cdot \kappa} \cdot \hat{e}(g,h)^{\alpha \cdot \mathcal{H}(M_b)} \cdot M_b \end{cases}$$

- **Case B** – this case refers to attributes' revocation, i.e., $\mathcal{C}$ defines ${\mathcal{S}_e}'^* = {\mathcal{S}_e}^* \setminus \mathcal{U}^*$ and generates a signcrypted message $\Sigma_{up,b}$.

$$\Sigma_{up,b} = \begin{cases} B_1 = \prod_{a_i \in S_s \cup D_{k+t-1-s} \setminus A_{\mathcal{C}}} \tau(a_i) \\ P_{(A_{\mathcal{C}}, S_s)}(\gamma) = \frac{1}{\gamma}(\prod_{a_i \in S_s \cup D_{k+t-1-s} \setminus A_{\mathcal{C}}} (\gamma + \tau(a_i)) - B_1) \\ \sigma_1 = g^{\frac{r_{\mathcal{C}}}{\prod_{a_i \in A_{\mathcal{C}}} (\gamma+\tau(a_i))}} \cdot g^{\frac{\mathcal{H}(M_b)}{\prod_{a_i \in A_{\mathcal{C}}} (\gamma+\tau(a_i))}} \\ \sigma_2 = sk_{\mathcal{C}_3} \cdot h^{r_{\mathcal{C}} P_{(A_{\mathcal{C}}, S_s)}(\gamma)/B_1} \cdot h^{\mathcal{H}(M_b) P_{(A_{\mathcal{C}}, S_s)}(\gamma)/B_1} \\ \sigma_3 = h^{\alpha \cdot \mathcal{H}(M_b)} \\ E_{0_{up}} = h^{\kappa \cdot \prod_{a_i \in S_e \setminus \mathcal{U}} (\gamma+\tau(a_i)) F(\gamma)} \\ C_{1_{up}} = u_1^{-\kappa F(\gamma)} \\ C_{M_{up}} = M_b \cdot \hat{e}(g,h)^{\alpha \cdot \mathcal{H}(M_b)} \cdot \hat{e}(g,h)^{\alpha\kappa F(\gamma)} \end{cases}$$

Guess – the adversary $\mathcal{A}$ selects a bit b'. Then, $\mathcal{B}$ sends b' to $\mathcal{C}$ as its guess about b. If $b' = b$, $\mathcal{C}$ answers 1 as the solution to the given instance of the $aMSE-CDH$ problem with respect to Definition 2 as introduced in [11].

The adversary $\mathcal{A}$ outputs a bit b'. The probability to break the instance of Exp^{conf} game is smaller than the ABSC-Game, as it is necessary for $\mathcal{B}$ to win the game for $\mathcal{A}$ to be able to get the correct Σ^*_b, and try to guess the value of b.

[4] As $\mathcal{C}$ executes both *DesignCrypt*$_{out}$ and *DesignCrypt* algorithms, we use *DesignCrypt* to refer to both algorithms

Referring to **Case A** and **Case B**, it is noticeable that the distribution of the received signcrypted challenge message does not depend on the attributes' addition and revocation. In other words, the distribution of the challenge signcrypted message is quite similar in both cases. In addition, thanks to the hardness of the $aMSE-CDH$ problem, $\mathcal{A}$ cannot guess the secret values such as r_i, γ used to generate the secret keys associated to the set of signcryption attributes S^*. Thus, the adversary is not able to guess the message even if he might after seeing the challenge signcrypted message. Thus, $Pr[Exp^{conf}(1^{\xi})] \leq Pr[Exp^{ABSC}(1^{\xi})]$, and the advantage of $\mathcal{A}$ is negligible.

Consequently, we show that our PROUD is secure against replayable chosen ciphertexts attacks in the standard model, under the $aMSE-CDH$ assumption, w.r.t Exp^{conf} security experiment.

□

7.3. Unforgeability

The following proof shows that PROUD is unforgeable against chosen message attacks w.r.t. Theorem 3.

Proof. The Exp^{unf} security game, introduced in Section 4.2.2, captures the behavior of a non-authorised signing entity. In this security game, the adversary $\mathcal{A}$ attempts to compute a signcrypted message, that can be correctly verified by the challenger $\mathcal{C}$, based on the DesignCrypt algorithm.

Similar to the confidentiality security game, discussed in subsection 7.2, we show that the proposed PROUD scheme inherits the unforgeability property from the ABSC scheme presented in [11], w.r.t Definition 2.

In fact, as stated above, PROUD introduces two additional properties, namely the update and outsourced-decryption features. However, while three main algorithms are added to support those features, they do not induce any changes on the signature process and signcrypted messages' distribution, contrary to the encryption process. More precisely, the Update and Transform algorithms do not involve any signature's component (i.e., $(\sigma_1, \sigma_2, \sigma_3)$), in both inputs and outputs, compared to the ABSC scheme, presented in [11]. Note that, the $\mathsf{DesignCrypt}_{out}$ algorithm also keeps unchanged the signature's components, except σ_3 which will be randomized using the transformation factor z.

To this end, we define an adversary $\mathcal{A}$ performing the Exp^{unf} security game with an entity $\mathcal{B}$. This entity $\mathcal{B}$ is also running the Belguith et al's EUF-CMA-security game (ABSC-Game) with a challenger $\mathcal{C}$ [11]. The aim of the proof is to demonstrate that the advantage of the adversary $\mathcal{A}$ to succeed in the Exp^{unf} game is smaller than the advantage of the entity $\mathcal{B}$ to win ABSC-Game.

During the initialisation phase, the adversary $\mathcal{A}$ chooses a set of signcryption attributes S^* and a threshold t. Afterwards, he asks for the execution of the **Private Key Query** while changing the threshold t_i. In addition, $\mathcal{A}$ asks for the signcryption of a message M under different signcryption attribute sets and threshold values. By executing this **SignCryption Query**, the adversary attempts to get information about the secret values included in the KeyGen and SignCrypt algorithms.

- **Private Key Query** – when $\mathcal{A}$ queries a signcryption attribute set $A_{\mathcal{A},i}$ w.r.t. a threshold t_i where $|A_{\mathcal{A},i} \cap S_e^*| < t_i$ and $|A_{\mathcal{A},i} \cap S_s^*| < t_i$, $\mathcal{C}$ answers by running the $\mathsf{KeyGen}(\mathtt{PP}, \mathtt{msk}, \mathcal{C}, A_{\mathcal{C},i})$ algorithm and returns the generated secret key $sk_{\mathcal{A},i} = (\{g^{\frac{r_{\mathcal{A}}}{\gamma+\tau(a)}}\}_{a\in A_{\mathcal{A}}}, \{h^{r_{\mathcal{A}}\gamma^j}\}_{j=0,\cdots,k-2}, h^{\frac{r_{\mathcal{A}}-1}{\gamma}})$ to $\mathcal{A}$.

- **SignCryption Query** – $\mathcal{A}$ requests the signcryption of a message M_i w.r.t a threshold t_i, while considering the access policies Ψ_e, Ψ_s. $\mathcal{C}$ executes the KeyGen algorithm to generate the secret key $sk_{\mathcal{C},i} = \mathsf{KeyGen}(\mathtt{PP}, \mathtt{msk}, \mathcal{C}, A_{\mathcal{C},i})$, such that $|A_{\mathcal{C},i} \cap S_e^*| \geq t_i$ and $|A_{\mathcal{C},i} \cap S_s^*| \geq t_i$. Thus, $\mathcal{C}$ executes $\mathsf{SignCrypt}(\mathsf{pp}, sk_{\mathcal{C}}, A_{\mathcal{C},i}, \Psi_e = (t, S_e), \Psi_s = (t, S_s), M)$ and returns a ciphertext Σ_i as following:

$$\Sigma_i = \begin{cases} B_1 = \prod_{a_i \in S_s \cup D_{k+t-1-s} \setminus A_{\mathcal{C}}} \tau(a_i) \\ P_{(A_{\mathcal{C}}, S_s)}(\gamma) = \frac{1}{\gamma}(\prod_{a_i \in S_s \cup D_{k+t-1-s} \setminus A_{\mathcal{C}}} (\gamma + \tau(a_i)) - B_1) \\ \sigma_1 = g^{\frac{r_O}{\prod_{a_i \in A_{\mathcal{C}}} (\gamma+\tau(a))}} \cdot g^{\frac{\mathcal{H}(M)}{\prod_{a_i \in A_{\mathcal{C}}} O(\gamma+\tau(a_i))}} \\ \sigma_2 = sk_{\mathcal{C}_3} \cdot h^{r_O P_{(A_{\mathcal{C}}, S_s)}(\gamma)/B_1} \cdot h^{\mathcal{H}(M) P_{(A_{\mathcal{C}}, S_s)}(\gamma)/B_1} \\ \sigma_3 = h^{\alpha \cdot \mathcal{H}(M)} \\ E_0 = h^{\kappa\alpha \cdot \prod_{a_i \in S_e} (\gamma+\tau(a_i))}, E_1 = E_0^{\gamma}, \cdots, E_{k-s} = E_{k-s-1}^{\gamma} \\ C_1 = u_1^{-\kappa}, \cdots, C_{r+1} = u_{r+1}^{-\kappa} \\ C_M = \hat{e}(g,h)^{\alpha \cdot \kappa} \cdot \hat{e}(g,h)^{\alpha \cdot \mathcal{H}(M)} \cdot M = K \cdot M \end{cases}$$

Subsequently, the adversary $\mathcal{A}$ attempts to generate a valid signcryption w.r.t to the challenge policy $(t^*, S*)$. Thus, the adversary $\mathcal{A}$ has to solve the $aMSE-CDH$ problem for proving that he has the required attributes to satisfy the (t^*, S^*) access policy. Additionally, thanks to the random values added to the generated signature, $\mathcal{A}$ has to solve the CDH problem.

However, thanks to the hardness of to the $aMSE-CDH$ problem and the CDH problem, the adversary cannot learn information about the secret values used in the key generation and the signcryption process.

Consequently, we show that our PROUD scheme is unforgeable against chosen message attack (EUF-CMA) in the standard model, under the $aMSE-CDH$ and CDH assumptions, w.r.t Exp^{unf} security experiment.

□

7.4. Privacy

In the following proof, we prove that PROUD is computationally private w.r.t. Theorem 4.

Proof. The Exp^{priv} security game, introduced in Section 4.2.3, captures the behavior of a curious entity. Indeed, the adversary $\mathcal{A}$ attempts to distinguish between two correctly-generated signcrypted messages that have been derived w.r.t. the same set

of attributes. More precisely, an adversary cannot link a signcrypted message to a specific signing entity. PROUD is said to be computationally private if any adversary $\mathcal{A}$ cannot win the Exp^{Priv} game with non-negligible advantage.

Upon receiving the set of public parameters generated by the challenger during the initialisation phase, $\mathcal{A}$ selects two attribute sets A_{S_1} and A_{S_2} which satisfy the access policy (t^*, S^*) and sends them to $\mathcal{C}$. The challenger generates the private keys related to the sets of attribute A_{S_1} and A_{S_2} as follows:

$$sk_{C_1} = (\{g^{\frac{r_C}{\gamma+\tau(a_i)}}\}_{a_i \in A_{S_1}}, \{h^{r_C\gamma^j}\}_{j=0,\cdots,m-2}, h^{\frac{r_C-1}{\gamma}})$$

$$sk_{C_2} = (\{g^{\frac{r_C}{\gamma+\tau(a_i)}}\}_{a_i \in A_{S_2}}, \{h^{r_C\gamma^j}\}_{j=0,\cdots,m-2}, h^{\frac{r_{E_C}-1}{\gamma}})$$

During the **Challenge Phase**, the adversary $\mathcal{A}$ generates a challenge message M and asks $\mathcal{C}$ to output the signcryption of M using one of the private keys sk_{C_1} or sk_{C_2}, associated to either A_{S_1} or A_{S_2}, respectively. For this purpose, the challenger chooses a random bit $b \in \{0,1\}$ and generates Σ_b by executing the algorithm $\mathsf{SignCrypt}(\mathsf{pp}, sk_{C_b}, A_{S_b}, (t^*, S^*), M_b)$. Recall that $|A_{S_b} \cap S^*| = t^*$, thus $\mathcal{C}$ can derive a valid signcrytpion on the randomly selected message M_b. Thus, to demonstrate that PROUD is privacy preserving, we only have to demonstrate that the signcrypted messages created using sk_{C_1} or sk_{C_2} are identically-distributed.

Relying on sk_{C_b}, the generated signcrypted message Σ_b is detailed as follows:

$$\Sigma_b = \begin{cases} B_1 = \prod_{a_i \in S^*{}_s \cup D_{k+t-1-s} \setminus A_{C_b}} \tau(a_i) \\ P_{(A_{C_b}, S^*{}_s)}(\gamma) = \frac{1}{\gamma}(\prod_{a_i \in S^*{}_s \cup D_{k+t-1-s} \setminus A_{C_b}} (\gamma + \tau(a_i)) - B_1) \\ \sigma_1 = g^{\frac{r_O}{\prod_{a_i \in A_{C_b}} (\gamma+\tau(a_i))}} \cdot g^{\frac{\mathcal{H}(M)}{\prod_{a_i \in A_{C_b}} O(\gamma+\tau(a_i))}} \\ \sigma_2 = sk_{C_{b,3}} \cdot h^{r_O P_{(A_C, S*_s)}(\gamma)/B_1} \cdot h^{\mathcal{H}(M_b) P_{(A_{C_b}, S^*{}_s)}(\gamma)/B_1} \\ \sigma_3 = h^{\alpha \cdot \mathcal{H}(M_b)} \\ E_0 = h^{\kappa\alpha \cdot \prod_{a_i \in S^*{}_e}(\gamma+\tau(a_i))}, E_1 = E_0^{\gamma}, \cdots, E_{k-s} = E_{k-s-1}^{\gamma} \\ C_1 = u_1^{-\kappa}, \cdots, C_{r+1} = u_{r+1}^{-\kappa} \\ C_{M,b} = \hat{e}(g,h)^{\alpha \cdot \kappa} \cdot \hat{e}(g,h)^{\alpha \cdot \mathcal{H}(M_b)} \cdot M_b = K \cdot M_b \end{cases}$$

Upon receiving Σ_b, the adversary checks the validity the signature, by calculating:

$$\begin{aligned} \text{Ⓢ} &= \hat{e}(u_0^{-1}, \sigma_{2_b}) \cdot \hat{e}(\sigma_{1_b}^{\frac{1}{B_1}}, h^{\alpha \cdot \prod_{a_i \in S \cup D_{n+t-1-s}} (\gamma+\tau(a_i))}) \cdot \\ &\quad \hat{e}(g^{\alpha}, h)^{\mathcal{H}(M_b) \cdot (1 - U_1 - \frac{1}{U_1})} = \hat{e}(g^{\alpha}, h) \end{aligned}$$

Since $|A_{S_b} \cap S^*| = |A_{S_1} \cap S| = |A_{S_2} \cap S^*| = t^*$, we prove that the signature generated using the set of attributes A_{S_1} (i.e., using the private key sk_{C_1}) is similar to the signature derived relying on the set of attributes A_{S_2} (i.e., using the secret key sk_{C_2}). Consequently, it is impossible for the adversary $\mathcal{A}$ to deduce the set of attributes used to derive the signature w.r.t. the hardness of the *CDH* problem.

Thus, our proposed PROUD scheme is computationally private, under the *CDH* assumption, w.r.t Exp^{priv} security experiment □

7.5. Verifiability

In the following proof, we prove that PROUD is verifiable against lazy servers w.r.t. Theorem 5.

Proof. The Exp^{verif} security games, presented in 4.2.4 captures the behaviour of lazy server STES. That is, the goal of an adversary $\mathcal{A}$ is to forge a compromised partially designcrypted ciphertext, that can be correctly verified by the challenger $\mathcal{C}$, by running Designcrypt algorithm.

To this end, we define an adversary $\mathcal{A}$ running the Exp^{verif} security game with an entity $\mathcal{B}$. This entity $\mathcal{B}$ is also running a collusion attack against hash function $\mathcal{H}$ with a challenger $\mathcal{C}$. The aim of this proof is to demonstrate that the advantage of the adversary $\mathcal{A}$ to succeed in the Exp^{verif} game is smaller than the advantage of the entity $\mathcal{B}$ to win the collusion game.

In the following, we detail the interactions carried between $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$. During the intialisation phase, $\mathcal{A}$ first defines a set of signcryption attributes S^*, which is then shared with $\mathcal{C}$ and $\mathcal{B}$. This latter derives and publishes the public parameters.

Afterwards, the interaction is mainly carried between $\mathcal{C}$ and $\mathcal{B}$, with $\mathcal{A}$ running the following steps and algorithms, as detailed in the Exp^{verif} game. That is, $\mathcal{A}$ tries to gain knowledge about the designcryption process by requesting the execution of the KeyGen, Transform and DesignCrypt algorithms, relying on **Private Key Query**, **Transformation Key Query**, **DesignCryption Query**, respectively.

Phase 1 Queries – $\mathcal{C}$ sets an empty table T. Then, for each session i, $\mathcal{A}$ is allowed to issue **Private Key Query**, **SignCryption Query** and **DesignCryption Query** queries as follows:

- **Private Key Query** – when $\mathcal{A}$ queries a signcryption attribute set $A_{\mathcal{A},i}$ w.r.t. a threshold t_i where $|A_{\mathcal{A},i} \cap S_e^*| < t_i$ and $|A_{\mathcal{A},i} \cap S_s^*| < t_i$, $\mathcal{C}$ answers by running the KeyGen $(\mathtt{PP}, \mathtt{msk}, \mathcal{C}, A_{\mathcal{C},i})$ algorithm and returns the generated secret key $sk_{\mathcal{A},i} = (\{g^{\frac{r_{\mathcal{A}}}{\gamma+\tau(a_i)}}\}_{a_i \in A_{\mathcal{A}}}, \{h^{r_{\mathcal{A}}\gamma^j}\}_{j=0,\cdots,k-2}, h^{\frac{r_{\mathcal{A}}-1}{\gamma}})$ to $\mathcal{A}$.
- **Transformation Key Query** – $\mathcal{A}$ requests the transformation key $tk_{\mathcal{A},i}$, associated to a set of attributes $A_{\mathcal{A},i}$. $\mathcal{C}$ searches the entry $(A_{\mathcal{A},i}, sk_{\mathcal{A},i}, tk_{\mathcal{A},i})$ in table T. It returns the transformation key if it exists in table T, otherwise $\mathcal{C}$ executes the Transform algorithm to generate $tpk_{\mathcal{A},i} = (\{g^{\frac{r_{\mathcal{A}}}{z(\gamma+\tau(a_i))}}\}_{a_i \in A_{\mathcal{A}}}, \{h^{\frac{r_{\mathcal{A}}\gamma^j}{z}}\}_{j=0,\cdots,k-2}, h^{\frac{r_{\mathcal{A}}-1}{z\gamma}})$ and $tsk_{\mathcal{A},i} = z$ and forwards them to the adversary.
- **SignCryption Query** – $\mathcal{A}$ requests the signcryption of a message M_i with respect to a threshold t_i, while considering the access policies Ψ_e, Ψ_s. $\mathcal{C}$ executes the KeyGen algorithm to generate the secret key $sk_{\mathcal{C},i} = \mathsf{KeyGen}(\mathtt{PP}, \mathtt{msk}, \mathcal{C}, A_{\mathcal{C},i})$, such that $|A_{\mathcal{C},i} \cap S_e^*| \geq t_i$ and $|A_{\mathcal{C},i} \cap S_s^*| \geq t_i$. Thus, $\mathcal{C}$ executes $\mathsf{SignCrypt}(\mathsf{pp}, sk_{\mathcal{C}}, A_{\mathcal{C},i}, \Psi_e = (t, S_e), \Psi_s = (t, S_s), M)$ and returns a ciphertext

$$\Sigma_i = \begin{cases} B_1 = \prod_{a_i \in \mathcal{S}_s \cup D_{k+t-1-s} \setminus A_C} \tau(a_i) \\ P_{(A_C,\mathcal{S}_s)}(\gamma) = \frac{1}{\gamma}(\prod_{a_i \in \mathcal{S}_s \cup D_{k+t-1-s} \setminus A_C} (\gamma + \tau(a_i)) - B_1) \\ \sigma_1 = g^{\frac{r_O}{\prod_{a_i \in A_C} (\gamma+\tau(a_i))}} \cdot g^{\frac{\mathcal{H}(M)}{\prod_{a_i \in A_C} O(\gamma+\tau(a_i))}} \\ \sigma_2 = sk_{C_3} \cdot h^{r_O P_{(A_C,\mathcal{S}_s)}(\gamma)/B_1} \cdot h^{\mathcal{H}(M) P_{(A_C,\mathcal{S}_s)}(\gamma)/B_1} \\ \sigma_3 = h^{\alpha \cdot \mathcal{H}(M)} \\ E_0 = h^{\kappa\alpha \cdot \prod_{a_i \in \mathcal{S}_e} (\gamma+\tau(a_i))}, E_1 = E_0^{\gamma}, \cdots, E_{k-s} = E_{k-s-1}^{\gamma} \\ C_1 = u_1^{-\kappa}, \cdots, C_{r+1} = u_{r+1}^{-\kappa} \\ C_M = \hat{e}(g,h)^{\alpha \cdot \kappa} \cdot \hat{e}(g,h)^{\alpha \cdot \mathcal{H}(M)} \cdot M = K \cdot M \end{cases}$$

- **DesignCryption Query** – $\mathcal{A}$ asks $\mathcal{C}$ to designcryp a ciphertext Σ_i with respect to a threshold t_i and a signcryption attribute set $A_{\mathcal{A},i}$. $\mathcal{C}$ executes the $\mathsf{KeyGen} = (\mathsf{pp}, msk, \mathcal{A}, A_{\mathcal{A},i})$ algorithm to output the related private key as $sk_{\mathcal{A},i} = \{g^{\frac{r_{\mathcal{A}}}{\gamma+\tau(a_i)}}\}_{a_i \in A_{\mathcal{A}}}, \{h^{r_{\mathcal{A}}\gamma^j}\}_{j=0,\cdots,k-2}, h^{\frac{r_{\mathcal{A}}-1}{\gamma}})$, such that $|A_{C,i} \cap S_e^*| \geq t_i$ and $|A_{C,i} \cap S_s^*| \geq t_i$. Finally, $\mathcal{C}$ runs the $\mathsf{DesignCrypt}(\mathsf{pp}, sk_{\mathcal{A},i}, (t,S_e), (t,S_s), \Sigma_i)$ algorithm[5] that outputs a message M_i or a reject symbol $\perp$.

Phase 2 Queries – in this phase, $\mathcal{A}$ can query a polynomially bounded number of queries as in **Phase 1 Queries**, except that $\mathcal{A}$ cannot query the designcryption of the challenge messages M_0 and M_1.

Hereafter, two cases are defined w.r.t. the ind operator value, chosen by $\mathcal{C}$ to encrypt the challenging message such that:

- **Case A** – it corresponds to attributes' addition, such that $\mathcal{C}$ sets $\mathcal{S}_e^{'*} = \mathcal{S}_e^* \cup \mathcal{U}^*$ and generates a signcrypted message $\Sigma_{up,b}$. In this case, we first show that how a challenge signcrypted message should be produced in the following:.

$$\Sigma_{up,b} = \begin{cases} B_1 = \prod_{a_i \in \mathcal{S}_s \cup D_{k+t-1-s} \setminus A_C} \tau(a_i) \\ P_{(A_C,\mathcal{S}_s)}(\gamma) = \frac{1}{\gamma}(\prod_{a_i \in \mathcal{S}_s \cup D_{k+t-1-s} \setminus A_C} (\gamma + \tau(a_i)) - B_1) \\ \sigma_1 = g^{\frac{r_C}{\prod_{a_i \in A_C} (\gamma+\tau(a_i))}} \cdot g^{\frac{\mathcal{H}(M_b)}{\prod_{a_i \in A_C} (\gamma+\tau(a_i))}} \\ \sigma_2 = sk_{C_3} \cdot h^{r_C P_{(A_C,\mathcal{S}_s)}(\gamma)/B_1} \cdot h^{\mathcal{H}(M_b) P_{(A_C,\mathcal{S}_s)}(\gamma)/B_1} \\ \sigma_3 = h^{\alpha \cdot \mathcal{H}(M_b)} \\ E_{0_{up}} = \prod_{i=0}^{l} E_i^{f_i} \\ C_1 = u_1^{-\kappa} \\ C_M = \hat{e}(g,h)^{\alpha \cdot \kappa} \cdot \hat{e}(g,h)^{\alpha \cdot \mathcal{H}(M_b)} \cdot M_b \end{cases}$$

- **Case B** – this case refers to attributes' revocation, i.e., $\mathcal{C}$ defines $\mathcal{S}_e^{'*} = \mathcal{S}_e^* \setminus \mathcal{U}^*$ and generates a signcrypted message $\Sigma_{up,b}$.

[5] As $\mathcal{C}$ executes both *DesignCrypt*$_{out}$ and *DesignCrypt* algorithms, we use *DesignCrypt* to refer to both algorithms

$$\Sigma_{up,b} = \begin{cases} B_1 = \prod_{a_i \in \mathcal{S}_s \cup D_{k+t-1-s} \setminus A_C} \tau(a_i) \\ P_{(A_C,\mathcal{S}_s)}(\gamma) = \frac{1}{\gamma}(\prod_{a_i \in \mathcal{S}_s \cup D_{k+t-1-s} \setminus A_C} (\gamma + \tau(a_i)) - B_1) \\ \sigma_1 = g^{\frac{r_C}{\prod_{a_i \in A_C} (\gamma+\tau(a_i))}} \cdot g^{\frac{\mathcal{H}(M_b)}{\prod_{a_i \in A_C} (\gamma+\tau(a_i))}} \\ \sigma_2 = sk_{C_3} \cdot h^{r_C P_{(A_C,\mathcal{S}_s)}(\gamma)/B_1} \cdot h^{\mathcal{H}(M_b) P_{(A_C,\mathcal{S}_s)}(\gamma)/B_1} \\ \sigma_3 = h^{\alpha \cdot \mathcal{H}(M_b)} \\ E_{0_{up}} = h^{\kappa \cdot \prod_{a_i \in \mathcal{S}_e \setminus \mathcal{U}} (\gamma+\tau(a_i)) F(\gamma)} \\ C_{1_{up}} = u_1^{-\kappa F(\gamma)} \\ C_{M_{up}} = M_b \cdot \hat{e}(g,h)^{\alpha \cdot \mathcal{H}(M_b)} \cdot \hat{e}(g,h)^{\alpha\kappa F(\gamma)} \end{cases}$$

Forge – $\mathcal{A}$ generates an attribute set $\{A_{\mathcal{A},i}^*\}$ and a valid partially designcrypted ciphertext Σ_{part}^* without executing the $\mathsf{DesignCrypt}_{\mathsf{out}}$ algorithm. The adversary $\mathcal{A}$ has to break the collusion resistance property of a $\mathcal{H}$. Thanks to the collusion resistance feature of hash functions [43], $\mathcal{A}$ cannot generates $\mathcal{H}(M_{part})$ that satisfies the equality $V = \sigma_3$ without knowing M. Therefore, if $M_{part} \neq M'$, then $V \neq \sigma_3$.

Consequently, we demonstrate that PROUD scheme is verifiable against lazy server attacks in the standard model, under to the collusion resistance of hash functions w.r.t to Exp^{verif} security experiment.

□

8. Performances Analysis

This section introduces the computation, storage and communication overheads of PROUD scheme. For our performance analysis, we present a detailed comparison of processing costs of main algorithms, i.e, SignCrypt, Update, DesignCrypt and $\mathsf{DesignCrypt}_{\mathsf{out}}$ while considering the size of users' secret keys, transformation keys and ciphertext against closely related ABSC schemes offering similar features, such as policy update and outsourced designcryption.

Table 3 details the computation costs of PROUD compared to the state-of-the-art schemes. While, Table 4 presents the storage and communication overheads comparison. Note that notations used in these comparisons are defined in Table 2.

In the following, we first discuss the computation costs in Section 8.2, and then, we detail the storage and communication overheads in Section 8.1.

8.1. Computation Complexities

As the computation of exponentiations operations and pairing functions are considered as the most costly mathematical operations in attribute based techniques, we compare ABSC schemes based on the execution of these operations in Table 3.

Liu et al. [26] have proposed an ABSC scheme supporting outsourced designcryption. In this scheme, the execution of the SignCrypt algorithm requires $(1+4|A_U|)$ exponentiations in $\mathbb{G}_1$. The outsourced designcryption procedure necessitates the performance of $2|A_U|$ pairing functions. The user is required to compute one exponentiation in $\mathbb{G}_1$, one exponentiation in $\mathbb{G}_1$ and $|A_U|$ execution of pairing functions-where $|A_U|$ is the size of the user's attribute set are computed, during the running of the SignCrypt algorithm.

Table 3: Computation Costs of ABSC Schemes

Scheme	*Update*	*Outsourcing*	*Update Cost*	*SignCryption Cost*	*User DesignCrytion Cost*	*STES DesignCrytion Cost*
[26]	✗	✓	$--$	$E_1(1+4\|A_U\|)$	$\|A_U\|\tau_p+E_1+E_T$	$2\tau_p\|A_U\|$
[25]	✗	✓	$--$	$(\|A_U\|+1)E+10E_1+E_1(6s_s+2s_e)+O(H)$	$3O(H)+\tau_p(5+2s_s)+E_1(5+7s_s+2s_e)$	$E_1(\|A_U\|+4+6s_s+2s_e)+E(2+s_s)+5\tau_p$
[29]	✗	✓	$--$	$E_1(7s_s+9+\|A_U\|+4s_e)+E(s_e+1)+3O(H)$	E_1	$E_1(9s_e+2+3s_s)+\tau_p(3+s_e)+2s_eE$
[6]	✓	✗	$(k-s_e+2)E_1+E$ $(r+2)E_1+E$	lE_1 $(2l+2)E_1+\tau_p$	$\|A_U\|E_1+2\tau_p$ $\|A_U\|E_1+2\tau_p$	$--$
[5]	✓	✗	$(k-s_e+2+r)E_1+E$	$(3l+2)E_1+\tau_p$	$2\tau_p+E_1+(\|A_U\|+1)E$	$--$
[30]	✓	✗	lE_3	$(1+k+\|A_E\|)E_2+E_1+E+\tau_p$	$2\tau_p+E_2+(\|A_U\|+1)E$	$--$
PROUD	✓	✓	$(1+l)E_2+2lE_1+\tau_p$	$(r+1)E_1+(3+k-s_e)E_2+2E$	$E+E_2+\mathcal{H}(M)$	$6\tau_p+3E_1+E_2+2E$

Table 4: Comparison of Storage Costs and Communication Overheads between PROUD and Closely Related ABSC Schemes

Schemes	*Key Size*	*Transformation Key size*	*Ciphertext Size*	SysInit *Communication Cost* (bits)	Storage *Communication Cost* (bits)	Retrieval From CSP *Communication Cost* (bits)	Retrieval From STES *Communication Cost* (bits)
[26]	$3s_s$	$2+3\|A_U\|+\|A_U\|\|A_O\|$	$\|A_U\|$	$\|pp\|+3s_s\Phi$	$\|A_U\|\Omega$	$\|A_U\|\Omega$	$(2+3\|A_U\|+\|A_U\|\|A_O\|)\Phi+\|A_U\|\Omega$
[29]	$2+2\|A_U\|$	$2+2\|A_U\|+2k$	$4+s_s+4s_e$	$\|pp\|+(2+2\|A_U\|)\Phi$	$(4+s_s+4s_e)\Omega$	$(4+s_s+4s_e)\Omega$	$(2+2\|A_U\|+2k)\Phi+(4+s_s+4s_e)\Omega$
[25]	$5+2\|A_U\|$	$4+\|A_U\|$	11Ω	$\|pp\|+(5+2\|A_U\|)\Phi$	11Ω	11Ω	$11\Omega+(4+\|A_U\|)\Phi$
[6]	$\|A_U\|+1$ $\|A_U\|+1$	$--$ $--$	$3+k-s_e$ / 3 $r+3$ / 3	$\|pp\|+(\|A_U\|+1)\Phi$ $\|pp\|+(\|A_U\|+1)\Phi$	$(3+k-s_e)\Omega$ $(r+3)\Omega$	3Ω 3Ω	$--$ $--$
[5]	$(\|A_U\|+1)k$	$--$	$3+k-s_e+r/3$	$\|pp\|+((\|A_U\|+1)k)\Phi$	$(3+k-s_e+r)\Omega$	3Ω	$--$
[30]	$(\|A_U\|+1)k$	$--$	$1+k-\|A_U\|/3$	$\|pp\|+((\|A_U\|+1)k)\Phi$	$(1+k-\|A_U\|)\Omega$	3Ω	$--$
PROUD	$\|A_U\|+k$	$\|A_U\|+k+1$	$6+k-s+r/8$	$\|pp\|+(\|A_U\|+k)\Phi$	$(6+k-s+r)\Omega$	8Ω	$(\|A_U\|+k+1)\Phi+8\Omega$

In [25], the signcryption of a message needs $(10+6s_s+2s_e)$ exponentiations in $\mathbb{G}_1$, and $(|A_U|+1)$ exponentiations in $\mathbb{G}$. To recover a plaintext, the edge server executes $(5+7s_s+2s_e)$ exponentiations in $\mathbb{G}_1$ and $(5+2s_s)$ pairing functions, while the end-users is required to compute $(|A_U|+4+6s_s+2s_e)$ exponentiations in $\mathbb{G}_1$ and 5 pairing functions to fully retrieve data.

The authors in [29] propose an outsourced ABSC scheme where the signcryption algorithm requires an overhead equal to $E_1(7s_s+9+|A_U|+4s_e)+E(s_e+1)+3O(H)$. The outsourced designcryption algorithm $\mathsf{DesignCrypt}_{\mathsf{out}}$ executes $(9s_e+2+3s_s)$ exponentiations in $\mathbb{G}_1$, $(3+s_e)$ pairing functions and $2s_e$ exponentiations in $\mathbb{G}$. Unlike the above-mentioned schemes, this scheme fully releases the user from most of the designcryption computation, therefore, DesignCrypt algorithm necessitates only one exponentiation in $\mathbb{G}$.

Despite of the application of the outsourcing feature have reduced the computation costs at the end-user side, the schemes presented in [26], [25] and [29] do not support access policy update.

Jiang et al. [6] introduces a CP-ABE scheme which supports access policy update. This scheme considers two independent constructions, the first for adding attributes while the second allows attributes revocation. In the attribute addition algorithm, the encryption of a message requires $k-s_e+2$ exponentiations in $\mathbb{G}_1$ and only one exponentiation in $\mathbb{G}$ while in the second algorithm $r+2$ exponentiations in $\mathbb{G}_1$ and only one exponentiation in $\mathbb{G}$. The update algorithm execution requires l exponentiations in $\mathbb{G}_1$ to add attributes and $2l+2$ exponentiations in $\mathbb{G}_1$ and one pairing operation to revoke attributes. The decryption algorithm overhead is equal to $2\tau_p+nE_1$.

The scheme proposed in [5] requires the execution of $(k-s_e+2+r)$ exponentiations in $\mathbb{G}_1$ and only one exponentiation in $\mathbb{G}$ during the encryption phase. In this scheme, the update procedure is executed using one algorithm that requires $(3l+2)$ exponentiations in $\mathbb{G}_1$ and one pairing function, to add and remove attributes.

PROUD is the first ABSC scheme that ensures updating access policies after generating ciphertext while reducing users overhead by applying outsourcing technique. During signcryption, the overhead is equal to $(r+1)E_1+(3+k-s_e)E_2+2E$. To designcrypt data, users leverages most of computation overhead to STES which executes 3 exponentiations in $\mathbb{G}_1$, one exponentiation in $\mathbb{G}_2$, 2 exponentiations in $\mathbb{G}$ and 6 pairing operations. In the last phase, the user is only required to execute one exponentiation in $\mathbb{G}_1$ to designcrypt data and $E_2+\mathcal{H}(M)$ to verify the accuracy of the plaintext.

PROUD scheme presents quite similar overhead compared to the costs of related ABSC schemes while providing more practical features mainly related to desingncryption outsourcing and policy update.

8.2. Storage and Communication Overheads

In [26], the authors proposed an outsourced ABSC scheme. Nevertheless, the storage overhead incurred by this scheme depends on the size of encryption and signing. For instance, the size of the generated ciphertext is proportional to the number of attributes composing the user's access policy. Deng et al. [25] have introduced a constant size ABSC scheme ensuring outsourced designcryption. This scheme generates a fixed number of ciphertext elements equal to 11. Although the scheme proposed in [29] supports outsourcing designcryption overheads to an edge server, it outputs a ciphertext whose size is equal to $4+s_s+4s_e$. Indeed, this ciphertext size grows with the number of attributes composing both signing and encrypting access policies. Except for ABSC scheme proposed in [25], state of the art outsourced ABSC schemes such as [25, 26] generate important storage and communication costs due to the size of the ciphertext. In ABE schemes [6, 5, 30], policy update feature affects the size of the ciphertext. Therefore, the size of the ciphertext produced by the data owner is not constant and is equal to $3+k-s_e+r$ that depends on the size of attributes universes, encrypting access policy and revocation set. This important size

is mainly due to addition of ciphertext elements required by the update algorithm. However, after performing the access policy update, the end-user only downloads a fixed number of ciphertext elements which saves storage costs at his side.

Similar to ABE schemes supporting access policy update, the data owner in PROUD generates a ciphertext whose size is equal to $6+k-s_e+r$. This ciphertext depends on r and k due to the ciphertext elements required to fulfill the update feature. The end-user downloads a fixed ciphertext size equal to 8 which reduces the storage and communication overhead incurred by the use of PROUD scheme.

As depicted by Table 4, the communications overhead of PROUD is acceptable in a cloud assisted IoT environment. Compared to other state of the art ABSC schemes, PROUD supports access policies update which requires storing extra ciphertext components at the CSP side. However, communication between users and CSP are consuming low costs thanks to the constant size of the ciphertext. Although the scheme presented in [25] offers a limited bandwidth consumption as well, PROUD outperforms it by ensuring access policy update.

PROUD introduces two interesting features, i.e., outsourced signcryption and access policy update while incurring reasonable storage and communication overheads.

8.3. Resource-Constrained Performance Analysis

As the presented use case scenario is a vehicular network, we aim at first testing the computation overheads on several generic IoT devices as well as on the servers sides.

To simulate the performances of PROUD on the server sides *i.e.,* CSP and STES, we have tested the performances of the elementary cryptographic operations (pairing and exponentiation operations) on a laptop. Table 5 presents the specification of the used computer.

As PROUD relies on the use of bilinear maps as well as mathematical operations in a multiplicative group, we investigate the impact of these operations on the performances of our proposal while considering three security levels (*cf.* Figure 3 and Figure 4).

In cryptography, the brute-force attack consists in checking all possible keys until the correct one is found (i.e; with a key of length S bits, there are 2^S possible keys). S is defined as the security level in symmetric cryptography. In asymmetric cryptography, the security level of an algorithm is defined with respect to the hardness of solving a mathematical problem such as the Discrete Logarithm Problem (*DLP*). The time required to resolve the *DLP* problem is much less important than trying the 2^S keys by a brute-force attack. That is why, public key cryptosystems must be longer than symmetric algorithm keys. For example, a 1024 RSA key-length bits provides a 80 key-length equivalent key of a symmetric algorithm.

On one hand, we choose to implement two symmetric pairing functions mainly type A and type E as well as two asymmetric pairing functions, type D and type G [44, 45**?**]. As shown in Figure 3, we notice that type A pairing function is slower than type D pairing function respectively type E and type G. In addition, the processing overheads of the different pairing functions

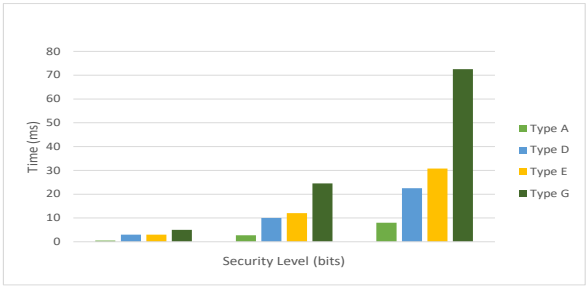

Figure 3: Pairing Function Computation Costs at the Server Side

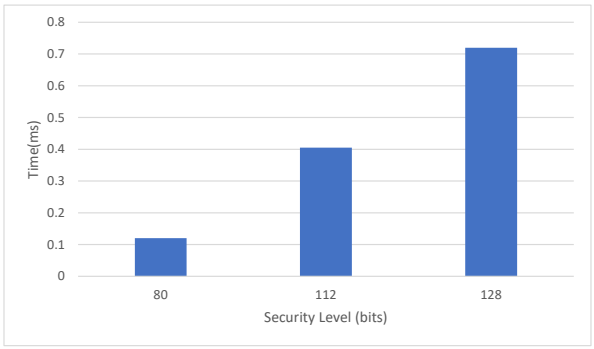

Figure 4: Exponentiation Computation Costs at the Server Side

increase along with the security level. For instance, type A pairing function requires 0.76 ms considering a security level equal to 80, however, type D pairing function takes 03.23 ms under the same security level. Besides, while considering a security level equal to 128, type A, type D and type G pairing functions' time durations are equal to 07.79ms, 22.6 ms and 72.46 ms, respectively. As such, the type of the pairing function should be taken into account, while implementing a cryptographic mechanism.

On the other hand, we evaluate the computation cost of the multiplication operation as this elementary operation is an important criterion to evaluate the system performances. Figure 5 illustrates that the average time of exponentiation operations increases along with the security level. We must note that these results have been obtained while choosing type A pairing function and fixing three security levels. The exponentiation oper-

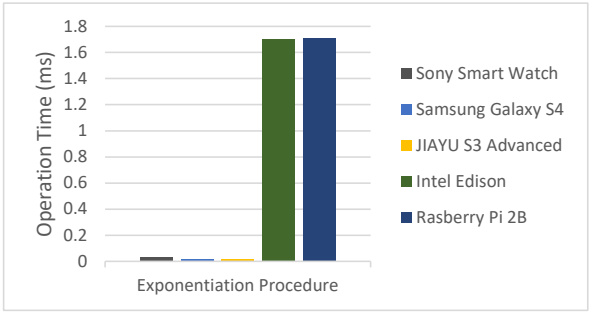

Figure 5: Exponentiation Computation Costs

Table 5: Performances of the Used Machines

Machine	OS	Processor	Memory	CPU
ASUS	Ubuntu 12.04	Intel Core i7-5500U	8 Go	3 GHZ

Table 6: Selected Devices

Device	Type	Processor
Sony SmartWatch 3 SWR50	Smart Watch	520 MHz Single-core Cortex-A7
Samsung I9500 Galaxy S4	Smartphone	1.6 GHz Dual-Core Cortex-A15
Jiayu S3 Advanced	Smartphone	1.7 GHz Octa-Core 64bit Cortex A53
Intel Edison	IoT Development Board	500 MHz Dual-Core Intel AtomTM CPU, 100 Mhz MCU
Raspberry Pi 2 model B	IoT Development Board	900 MHz Quad-Core ARM Cortex-A7

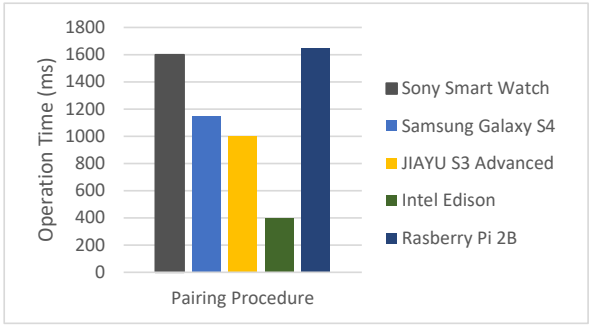

Figure 6: Pairing Computation Costs

ation requires 0.112 ms and 0.727 considering security levels equal to 80 and 128, respectively. In addition, the multiplication overhead starts from 0.001 ms while fixing a security level equal to 80 and goes up to 0.003 ms considering a security level equal to 128.

Figure 6 shows the computation cost of one pairing operation in different IoT devices (cf., Table 6). The most efficient device is Intel Edison that computes a single pairing operation in around 500 ms. The computation cost of an exponentiation operation in different IoT devices, as presented in Table 6, is shown in Figure 5.

9. CONCLUSIONS

IoT devices are generating an extremely growing amount of data that cannot be handled by IoT devices themselves. Cloud and edge computing permit to assist and provide storage and computation capabilities to IoT devices in order to help them with processing and storing generated data. Thus, Cloud assisted IoT has emerged as an extension of IoT networks to ensure the optimal computation and storage for data collected by IoT devices. However, these paradigms raise several security and privacy concerns.

A novel privacy-preserving attribute based signcryption scheme, PROUD, is introduced. It presents several security features that makes it suitable for several Cloud-assisted IoT environments such as vehicular networks, e-health systems and smart homes. PROUD ensures privacy preserving and flexible access control to shared data among dynamic groups of users. It also reduces computation overheads at the end-users' side by leveraging edge servers support to partially designcrypt received data. PROUD is the first ABSC scheme that support updating access policies by adding and/or removing attributes after generating and storing the ciphertext in the cloud without requiring any proxy-servers neither re-encrypting data or re-issuing users' keys.

The security of PROUD has been proved through four security games, w.r.t. the confidentiality, unforgeability, privacy and verifiability properties. That is, additionally to correctness, PROUD has been proven to be resistant to several data leakage and forgery attacks performed either by a curious entity, a malicious user or a lazy edge server. Finally, a theoretical performances' evaluation pointed out that PROUD provides acceptable overheads in terms of communication, storage and processing, compared to related work.

As future work, we aim to test the performances of PROUD in a real-world environment by implementing the different algorithms in an Autonomous Vehicle Platooning environment as explained in the proposed use-case scenario. In addition, we aim to review the designed attribute based signcryption scheme in order to propose a post-quantum construction [46].

References

[1] Regulation(EU), 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation), ojeu l 119/1 of 4.05.2016., 2016.

[2] M. Gagne, S. Narayan, R. Safavi-Naini, Threshold attribute-based signcryption, in: Security and Cryptography for Networks, Springer, 2010.

[3] S. Belguith, N. Kaaniche, M. Mohamed, G. Russello, Coop-daab: Cooperative attribute based data aggregation for internet of things applications, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, 2018, pp. 498–515.

[4] S. Debnath, M. V. Nunsanga, B. Bhuyan, Study and scope of signcryption

for cloud data access control, in: Advances in Computer, Communication and Control, Springer, 2019, pp. 113–126.

[5] S. Belguith, N. Kaaniche, G. Russello, Lightweight attribute-based encryption supporting access policy update for cloud assisted iot, in: 13th IEEE International Conference on Security and Cryptography(Secrypt), 2018, pp. 135–146. doi:10.5220/0006854601350146.

[6] Y. Jiang, W. Susilo, Y. Mu, F. Guo, Ciphertext-policy attribute-based encryption supporting access policy update and its extension with preserved attributes, International Journal of Information Security (2017) 1–16.

[7] L. Nkenyereye, Y. Park, K. H. Rhee, A secure billing protocol over attribute-based encryption in vehicular cloud computing, EURASIP Journal on Wireless Communications and Networking 2016 (1) (2016) 196.

[8] A. Sajid, H. Abbas, Data privacy in cloud-assisted healthcare systems: state of the art and future challenges, Journal of medical systems 40 (6) (2016) 155.

[9] S. Cui, S. Belguith, P. De Alwis, M. R. Asghar, G. Russello, Malicious entities are in vain: Preserving privacy in publish and subscribe systems, in: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), IEEE, 2018, pp. 1624–1627.

[10] C. Esposito, M. Ciampi, On security in publish/subscribe services: a survey, IEEE Communications Surveys & Tutorials 17 (2) (2015) 966–997.

[11] S. Belguith, N. Kaaniche, M. Laurent, A. Jemai, R. Attia, Constant-size threshold attribute based signcryption for cloud applications, in: SECRYPT 2017: 14th International Conference on Security and Cryptography, Vol. 6, Scitepress, 2017, pp. 212–225.

[12] N. Chen, M. Gerla, D. Huang, X. Hong, Secure, selective group broadcast in vehicular networks using dynamic attribute based encryption, in: Ad Hoc Networking Workshop (Med-Hoc-Net), 2010 The 9th IFIP Annual Mediterranean, IEEE, 2010, pp. 1–8.

[13] J. Zhou, X. Dong, Z. Cao, A. V. Vasilakos, Secure and privacy preserving protocol for cloud-based vehicular dtns, IEEE Transactions on Information Forensics and Security 10 (6) (2015) 1299–1314.

[14] F. Gonçalves, B. Ribeiro, V. Hapanchak, S. Barros, O. Gama, P. Araújo, M. J. Nicolau, B. Dias, J. Macedo, A. Costa, et al., Secure management of autonomous vehicle platooning, in: Proceedings of the 14th ACM International Symposium on QoS and Security for Wireless and Mobile Networks, ACM, 2018, pp. 15–22.

[15] F. Chen, Y. Han, D. Jiang, X. Li, X. Yang, Outsourcing the unsigncryption of compact attribute-based signcryption for general circuits, in: International Conference of Young Computer Scientists, Engineers and Educators, Springer, 2016, pp. 533–545.

[16] S. Belguith, N. Kaaniche, M. Hammoudeh, Analysis of attribute-based cryptographic techniques and their application to protect cloud services, Transactions on Emerging Telecommunications Technologies e3667.

[17] Y. S. Rao, R. Dutta, Efficient attribute-based signature and signcryption realizing expressive access structures, International Journal of Information Security 15 (1) (2016) 81–109.

[18] Y. S. Rao, Attribute-based online/offline signcryption scheme, International Journal of Communication Systems 30 (16) (2017) e3322.

[19] S. Belguith, N. Kaaniche, M. Laurent, A. Jemai, R. Attia, Accountable privacy preserving attribute based framework for authenticated encrypted access in clouds, Journal of Parallel and Distributed Computing 135 (2020) 1–20.

[20] M. Green, S. Hohenberger, B. Waters, et al., Outsourcing the decryption of abe ciphertexts, in: USENIX Security Symposium, no. 3, 2011.

[21] W. H. Negalign, H. Xiong, A. A. Addis, Y. G. Ashenafi, D. M. Geresu, Outsourced attribute-based signcryption in the cloud computing, in: 2018 15th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), IEEE, 2018, pp. 40–44.

[22] J. Lai, R. H. Deng, C. Guan, J. Weng, Attribute-based encryption with verifiable outsourced decryption, IEEE Transactions on Information Forensics and Security 8 (8) (2013) 1343–1354.

[23] J. Li, F. Sha, Y. Zhang, X. Huang, J. Shen, Verifiable outsourced decryption of attribute-based encryption with constant ciphertext length, Security and Communication Networks 2017.

[24] S. Belguith, N. Kaaniche, M. Laurent, A. Jemai, R. Attia, Phoabe: Securely outsourcing multi-authority attribute based encryption with policy hidden for cloud assisted iot, Computer Networks 133 (2018) 141–156.

[25] F. Deng, Y. Wang, L. Peng, H. Xiong, J. Geng, Z. Qin, Ciphertext-policy attribute-based signcryption with verifiable outsourced designcryption for sharing personal health records, IEEE Access.

[26] X. LIU, Y. Xia, Z. Sun, Provably secure attribute based signcryption with delegated computation and efficient key updating, KSII Transactions on Internet and Information Systems 11 (5) (2017) 2646.

[27] K. Liang, M. H. Au, J. K. Liu, W. Susilo, D. S. Wong, G. Yang, Y. Yu, A. Yang, A secure and efficient ciphertext-policy attribute-based proxy re-encryption for cloud data sharing, Future Generation Computer Systems 52 (2015) 95–108.

[28] C. Ge, W. Susilo, L. Fang, J. Wang, Y. Shi, A cca-secure key-policy attribute-based proxy re-encryption in the adaptive corruption model for dropbox data sharing system, Designs, Codes and Cryptography (2018) 1–17.

[29] Q. Xu, C. Tan, Z. Fan, W. Zhu, Y. Xiao, F. Cheng, Secure data access control for fog computing based on multi-authority attribute-based signcryption with computation outsourcing and attribute revocation, Sensors 18 (5) (2018) 1609.

[30] S. Belguith, N. Kaaniche, G. Russello, Pu-abe: Lightweight attribute-based encryption supporting access policy update for cloud assisted iot, in: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), IEEE, 2018, pp. 924–927.

[31] S. Belguith, N. Kaaniche, G. Russello, Cups: Secure opportunistic cloud of things framework based on attribute-based encryption scheme supporting access policy update, Security and Privacy e85.

[32] Y. S. Rao, A secure and efficient ciphertext-policy attribute-based signcryption for personal health records sharing in cloud computing, Future Generation Computer Systems 67 (2017) 133–151.

[33] K. Emura, A. Miyaji, M. S. Rahman, Dynamic attribute-based signcryption without random oracles, International Journal of Applied Cryptography 2 (3).

[34] J. Liu, X. Huang, J. K. Liu, Secure sharing of personal health records in cloud computing: ciphertext-policy attribute-based signcryption, Future Generation Computer Systems 52.

[35] S. Belguith, N. Kaaniche, A. Jemai, M. Laurent, R. Attia, Pabac: a privacy preserving attribute based framework for fine grained access control in clouds, in: 13th IEEE International Conference on Security and Cryptography(Secrypt), 2016.

[36] N. Kaaniche, M. Laurent, Attribute-based signatures for supporting anonymous certification, in: European Symposium on Research in Computer Security, Springer, 2016, pp. 279–300.

[37] D. H. Phan, D. Pointcheval, On the security notions for public-key encryption schemes, in: International Conference on Security in Communication Networks, Springer, 2004, pp. 33–46.

[38] R. Canetti, H. Krawczyk, J. B. Nielsen, Relaxing chosen-ciphertext security, in: Annual International Cryptology Conference, Springer, 2003, pp. 565–582.

[39] C. Delerablée, P. Paillier, D. Pointcheval, Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys, in: International Conference on Pairing-Based Cryptography, Springer, 2007, pp. 39–59.

[40] C. Delerablée, D. Pointcheval, Dynamic threshold public-key encryption, in: Annual International Cryptology Conference, Springer, 2008.

[41] J. Herranz, F. Laguillaumie, C. Ràfols, Constant size ciphertexts in threshold attribute-based encryption, in: International Workshop on Public Key Cryptography, Springer, 2010, pp. 19–34.

[42] N. Attrapadung, J. Herranz, F. Laguillaumie, B. Libert, E. De Panafieu, C. Ràfols, Attribute-based encryption schemes with constant-size ciphertexts, Theoretical Computer Science 422 (2012) 15–38.

[43] Y. Lindell, J. Katz, Introduction to modern cryptography, Chapman and Hall/CRC, 2014.

[44] B. Lynn, On the implementation of pairing-based cryptosystems, Ph.D. thesis, Stanford University (2007).

[45] S. Belguith, N. Kaaniche, M. Mohamed, G. Russello, C-absc: cooperative attribute based signcryption scheme for internet of things applications, in: 2018 IEEE International Conference on Services Computing (SCC), IEEE, 2018, pp. 245–248.

[46] M. S. Rahman, A. Basu, S. Kiyomoto, Decentralized ciphertext-policy attribute-based encryption: A post-quantum construction., J. Internet Serv. Inf. Secur. 7 (3) (2017) 1–16.