UNIVERSITÀ
DEGLI STUDI
FIRENZE

PhD Program in Smart Computing
Dipartimento di Ingegneria dell'Informazione (DINFO)

Applying Attribute Based Encryption in IoT and Automotive Scenarios

Michele La Manna

Dissertation presented in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Smart Computing

PhD Program in Smart Computing
University of Florence, University of Pisa, University of Siena

APPLYING ATTRIBUTE BASED ENCRYPTION IN IOT AND AUTOMOTIVE SCENARIOS

Michele La Manna

Advisor:

Prof. Gianluca Dini

Head of the PhD Program:

Prof. Paolo Frasconi

Evaluation Committee:
Prof. Javier Herranz, *Universitat Politècnica de Catalunya*
Dr. Tooska Dargahi, *University of Salford*

XXXIV ciclo — October 2022

To My Family

Acknowledgments

I would like to thank my advisor, Prof. Gianluca Dini, for giving me the opportunity to pursue my passion for research, as well as my love for teaching. Thanks to his advice, always constructive and never judgmental, I improved the quality of my work. With his kindness and sincerity, he built a competent and enjoyable group as well as a friendly yet productive work environment.

I also wish to thank Dr. Pericle Perazzo, who has assisted me in every step of my Ph.D., basically becoming my second advisor. He taught me how to *properly* write in English and how to perform meaningful experiments. He involved me in his primary research ideas and projects, helping me find the direction of my work. Most importantly, he did not invite me to get lost when, oftentimes, we stubbornly argued over some minor details in our work.

I am grateful to Prof. Enzo Mingozzi and Dr. Stefano Chessa, who have evaluated my work in these three years and deemed it valuable. Thanks to their feedback, I improved my presentation skills and the clarity of my exposition.

I reserve special thanks to Dr. Simona Altamura from the University of Florence, a beacon of light in the night of Italian bureaucracy. Without her, half of my Ph.D. time would have been dedicated to figuring out how to compile modules, requests, and other hideous papers needed at the time.

Abstract

With the advent of the Internet of Things (IoT) and the Industrial IoT (IIoT), the amount of information available has grown so much that the main focus was to collect and store such data efficiently. Unfortunately, as often happens, security was initially overlooked, and this lack of protection led to several cyber-attacks against privates, companies, and even national agencies. Indeed, data generated by IoT devices are usually stored on the Internet (e.g., on a cloud server) in such a way that they are virtually accessible to anybody at any time. Data in such a state is called *data at rest*. Access to those cloud servers is protected with some access control mechanisms, for example, enforced via software so that any user can access only data that he/she possesses. *Data Leakage* is an outpour of sensible information by a hacker that gained control of a cloud server and, therefore, can access every information stored on it. One solution is to encrypt data at rest. On the one hand, encryption addresses the data leakage problem, but, on the other hand, it makes data sharing complex.

Attribute-Based-Encryption (ABE) is an asymmetric encryption scheme that allows one to mathematically enforce an Access Control Mechanism (ACM) during the decryption procedure so that only permitted entities can access the protected data. The main advantage of using ABE in (I)IoT systems is to achieve multiple-receiver encryption with fine-grained access control. ABE provides confidentiality for *data at rest* (e.g., stored on third-party cloud storage) while allowing parties with different access privileges to decrypt it.

In this dissertation, we investigate the problem of engineering the Attribute-Based Encryption schemes within IoT (Internet of Things) and IIoT (Industrial IoT) systems. However, this displays tough challenges, particularly bandwidth consumption, feasibility over constrained devices, access policies management, and key management. Our investigation has been carried out in several directions. First, we compared the ABE approach with another similar technique in the literature, namely the Sticky Policy technique, to point out ABE potentialities. Secondly, we discuss some scenarios in which ABE can be adopted, while improving one of the original schemes. Then, we evaluate ABE performances in terms of bandwidth, energy consumption, computation time, and CPU load on a broad range of devices: from IoT constrained devices like the ESP32, to the RaspberryPi 3, to a more advanced automotive-compliant Xilinx ZCU 102 evaluation board. We also approach the related problems of: (i) reducing the encryption overhead of the ABE ciphertext; (ii) designing a recovery mechanism in case of key compromise; (iii) correctly selecting the most suitable ABE scheme for any IoT applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Security is one of the most critical issues of Internet communications. However, security is always a trade-off between performance and protection: we have to provide the proper protection for the overall lower cost possible. Cyber-threats awareness is widespread since the news is full of new cyber-attacks reports (CSIS, 2021). According to the latest OWASP Top 10 Project classifications (OWASP, 2021), broken authentication and sensitive data exposure are respectively the second and third most diffused cyber-attacks. Broken authentication means, in most cases, that a cryptographic key has been compromised. Sensitive data exposure, instead, is usually possible if *data at rest* is not encrypted. Data at rest is the information stored on the Internet (e.g., on a cloud server), always available for download. Ideally, download is possible only for the owner of such data and to whom it has granted access. To address these two problems, we need a way to encrypt data at rest, possibly accessible for many different entities, and that is resilient to key compromise.

Attribute-Based Encryption (ABE) is an *asymmetric* encryption scheme first proposed in (Sahai and Waters, 2005) as a variant of Identity-Based Encryption (IBE). The basic intuition of the first ABE scheme was to describe with a list of attributes both the data to encrypt and the decrypting user in the system. If the two sets shared a number of attributes greater than a threshold value "x" (chosen at encryption time), then decryption would be possible. This means that ABE provides *fine-grained* access control and multiple-receiver encryption since different entities (described by different attributes) may decrypt the same ciphertext. Researchers have evolved and enriched the original ABE scheme, and now we have many different ABE schemes. In particular, in (Bethencourt et al., 2007) was proposed the first Ciphertext-Policy Attribute-Based Encryption (CP-ABE) scheme. In the CP-ABE paradigm, an *access policy* is embedded in the ciphertext, while an *attribute set* is embedded inside the decryption key. Decryption is possible if and only if the attribute set *satisfies* the access policy. Since the access policy is determined at encryption time, the CP-ABE paradigm guarantees the data producer strong control over the accessibility of the

encrypted data.

Attribute-Based Encryption is a promising technique that can be used to protect data at rest. Through this dissertation, the reader will quickly understand the foundations of the ABE technique and its numerous variants. We will show several scenarios in which the use of ABE increases the overall system's security with a limited impact on the performances. By the end of this thesis, the reader will have a clear picture of the potentiality of ABE, the state-of-the-art, and he/she will have the tools to evaluate the feasibility of any ABE scheme in any IoT application.

1.1 Structure of the Dissertation

The contributions of this thesis are many-folds. Our research efforts were driven by questions that arose each time we found the answer to a previous question. We decided to structure this thesis after the questions that we asked ourselves so that, in each chapter, we can guide the reader to the answers we found.

Chapter 2. Background: What is Attribute-Based Encryption?

In this chapter, we introduce in-depth the main aspects and features of ABE, giving the reader a solid background to understand the contributions of this thesis.

Chapter 3. Attribute-Based Encryption vs. Sticky Policies: What Should We Use?

In this chapter, we consider the specific context of a smart home, which represents one of the main IoT application domains, and we focus on ABE and Sticky Policies, two solutions proposed in the literature to cope with the aforementioned issues. We compare the advantages and the drawbacks in terms of performance and robustness of such two techniques through their integration within the prototype of an IoT middleware named NetwOrked Smart object (NOS). The effectiveness of the presented solutions is validated by means of a real test-bed in the smart home scenario in terms of storage occupancy, CPU load, and data retrieval delay. The final goal is to reveal the best approach to be used depending on the application's requirements.

Chapter 4. Can CP-ABE be Used in a Low-Bitrate WSAN?

In this chapter, we show fABElous, an ABE scheme suitable for Industrial IoT applications, which aims at minimizing the overhead of encryption on communication. Industry can take enormous advantage of IoT, leading to the so-called Industrial IoT (IIoT). In these systems, integrity, confidentiality, and access control over data

are critical requirements. fABElous ensures data integrity, confidentiality, and access control while reducing the communication overhead by 35% compared to using ABE techniques naively.

Chapter 5. How can We Improve the Original CP-ABE?

In this chapter, we propose a novel ABE scheme called SEA-BREW (Scalable and Efficient Abe with Broadcast REvocation for Wireless networks), which is suited for the Internet of Things (IoT) and Industrial IoT (IIoT) applications. In contrast to state-of-the-art ABE schemes, ours can securely perform key revocations with a single short broadcast message instead of a number of unicast messages linear with the number of nodes. This is desirable for low-bitrate Wireless Sensor and Actuator Networks (WSANs), which often are the heart of (I)IoT systems. In SEA-BREW, sensors, actuators, and users can exchange encrypted data via a cloud server or directly via wireless if they belong to the same WSAN. We formally prove that our scheme is secure also in case of an untrusted cloud server that colludes with a set of users under the generic bilinear group model. We show by simulations that our scheme requires a constant computational overhead on the cloud server with respect to the complexity of the access control policies. This is in contrast to state-of-the-art solutions, which require a linear computational overhead instead.

Chapter 6. How Much Classical ABE Schemes Actually Impact Constrained IoT Devices?

In this chapter, we consider IoT devices characterized by strong limitations in terms of computing, storage, and power. Specifically, we assess the performance of ABE in typical IoT constrained devices. We evaluate the performance of two representatives ABE schemes configured considering the worst-case scenario on two popular IoT platforms, namely ESP32 and RE-Mote. Our results show that, if we assume to employ up to 10 attributes in ciphertexts and leverage hardware cryptographic acceleration, ABE can indeed be adopted on devices with very limited memory and computing power while obtaining a satisfactory battery lifetime. In our experiments, as also performed in other works in the literature, we consider only the worst-case configuration, which, however, might not be completely representative of the real working conditions of sensors employing ABE. For this reason, we completed our evaluation by proposing a novel benchmark method that we used to complement the experiments by evaluating the average performance. We show that by always considering the worst case, the current literature significantly overestimates the processing time and the energy consumption.

Chapter 7. What is the Most Suitable ABE Scheme for my System?

In this chapter, we survey the ABE literature proposing schemes and solutions that are best suited for IoT applications. To do so, we first identify six performance indicators in IoT: the data producer CPU efficiency, the data producer bandwidth efficiency, the key authority bandwidth efficiency, the data consumer CPU efficiency, the data consumer bandwidth efficiency, and the data producer storage efficiency. Then, we analyze only those schemes that are promising from the point of view of one or more indicators and, therefore, more applicable in typical IoT applications. The chapter presents a subset of representative schemes and assesses their efficiency by thorough simulations as a further contribution. Such simulations show that no scheme excels in all performance indicators at once, but some simultaneously perform well in two or more indicators.

Chapter 8. Is it Feasible to Leverage CP-ABE in the Automotive Environment?

In this chapter, we show that it is possible to improve security for over-the-air update functionalities in an automotive scenario through the use of ABE, which grants confidentiality to the software/firmware update done Over The Air (OTA). We demonstrate that ABE is seamlessly integrable into the state of the art solutions regarding the OTA update by showing that the overhead of the ABE integration in terms of computation time and storage is negligible w.r.t. the other overheads introduced by the OTA process, also proving that security can be enhanced with a minimum cost. To support our claim, we report the experimental results of an implementation of the proposed ABE OTA technique on a Xilinx ZCU102 evaluation board, which is an automotive-oriented HW/SW platform equipped with a Zynq UltraScale+ MPSoC chip that is representative of the computing capability of real automotive Electronic Control Units (ECUs).

Chapter 2

Background: What is Attribute-Based Encryption?

The basic building block of ABE is the concept of *attribute*, which is a property associated with a piece of data or with a data consumer.

An *access policy* describes the access authorization associated either with a piece of data or with a data consumer. It is typically represented through a Boolean formula that has attributes as arguments. An access policy is typically visualized as a tree (*policy tree*) in which leaf nodes are attributes, and intermediate nodes are Boolean operators. Different schemes allow for different degrees of *expressiveness* in the access policies, meaning that they impose constraints on the shape of the policy tree. For example, some ABE schemes allow only for k-of-n operators (*threshold gates*) or only for AND operators, or they limit the height of the policy tree. Some schemes use an LSSS (Linear Secret Sharing Scheme) representation, which means an access policy is expressed with a matrix-vector pair in place of a Boolean formula. However, there are algorithms in the literature (Lewko and Waters, 2011; Liu et al., 2010) to transform an LSSS representation into a policy tree, hence in the following, we will always represent policies employing policy trees without loss of generality. ABE comes in two paradigms: *Key-Policy Attribute-Based Encryption* (KP-ABE) and *Ciphertext-Policy Attribute-Based Encryption* (CP-ABE). In both paradigms, to encrypt data, it is necessary to own a copy of the *public parameters*, which are public and unique for all encrypting parties. Moreover, to decrypt data, it is necessary to own a *decryption key*, which is private and specific for each decrypting party. In KP-ABE, ciphertexts are associated with a set of attributes that describe them, and decryption keys are associated with an access policy. Access policies describe the "capability to access what", referring to the owner of the decryption key. KP-ABE schemes empower the key authority since it decides the access authorizations when creating decryption keys. Conversely, in CP-ABE, ciphertexts are associated with an access policy, and decryption keys are associated with a set of attributes. Access policies

describe the "capability to be accessed by whom", referring to the encrypted data. CP-ABE schemes empower the data producers since they decide the access authorizations at the moment of encrypting data.

2.1 Pairing-Based Cryptography Basics

Pairing-based cryptography, inaugurated by Boneh and Franklin (Boneh and Franklin, 2001), refers to the usage of bilinear maps (also called *pairings*) to construct cryptographic schemes. Most ABE schemes currently in the literature use pairing-based cryptography. The following definitions are commonly used in pairing-based cryptography. Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be three multiplicative cyclic groups of equal order whose group operations are efficiently computable. Let p be their prime order, g_1 be a generator of $\mathbb{G}_1$, and g_2 be a generator of $\mathbb{G}_2$. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map that has the following properties.

- *Bilinearity:* for all $a, b \in \mathbb{Z}_p$, $u \in \mathbb{G}_1$, and $v \in \mathbb{G}_2$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
- *Non-degeneracy:* $e(g_1, g_2) \neq 1$.
- *Computability:* There exists an efficient algorithm to compute e.

In practical realizations of pairings, $\mathbb{G}_1$ and $\mathbb{G}_2$ are sets of points on an elliptic curve, while $\mathbb{G}_T$ is a finite field. The group operation of $\mathbb{G}_1$ and $\mathbb{G}_2$ is thus the point-scalar multiplication, while that of $\mathbb{G}_T$ is the modular exponentiation.

Literature on pairing-based cryptography traditionally categorizes pairings into three types (Galbraith et al., 2008):

- *Type I*. These pairings have $\mathbb{G}_1 = \mathbb{G}_2$.
- *Type II*. These pairings have $\mathbb{G}_1 \neq \mathbb{G}_2$, but there is an efficient homomorphism to map an element in $\mathbb{G}_2$ to an element in $\mathbb{G}_1$.
- *Type III*. These pairings have $\mathbb{G}_1 \neq \mathbb{G}_2$, and there is no efficient homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

Type I pairings are also called *symmetric pairings*. To describe schemes that use symmetric pairings, we will use a unique symbol $\mathbb{G}$ to represent both $\mathbb{G}_1$ and $\mathbb{G}_2$. Type II and Type III are also called *asymmetric pairings*. However, it is worth noting that Type II pairings are rarely used to construct cryptographic schemes because they are inefficient compared to Type III ones, and they also miss some features, e.g., no efficient method to hash onto $\mathbb{G}_2$ (Menezes et al., 2016).

2.2 Main ABE features

Basic ABE Algorithms

Any KP-ABE scheme implements at least the following four algorithms.

- (MK, EK) = **Setup**(κ). This algorithm initializes the scheme with a strength given by the security parameter κ and randomly creates and returns a *master key MK* and the public parameters *EK*. The master key is kept secret by the key authority, whereas the public parameters are made public and used to encrypt data.

- (C) = **Encrypt**(M, γ, EK). This algorithm encrypts a message M $(plaintext)$ described by the attribute set γ, by means of the public parameters *EK*. It returns the encrypted message C $(ciphertext)$, which embeds the given attribute set.

- (DK) = **KeyGen**$(\mathcal{T}, MK)$. This algorithm creates a new decryption key associated with the access policy $\mathcal{T}$, by means of the master key *MK*. It returns the decryption key *DK*, which embeds the given access policy.

- $(M$ or $\perp)$ = **Decrypt**(C, DK). This algorithm decrypts a ciphertext C with the decryption key *DK*. It returns the original message M if and only if the access policy $\mathcal{T}$ evaluates to true on the attribute set γ embedded in the ciphertext C, otherwise it returns the null value $\perp$.

Any CP-ABE scheme implements at least the following four algorithms.

- (MK, EK) = **Setup**(κ). This algorithm acts similarly to the one in the KP-ABE schemes.

- (C) = **Encrypt**$(M, \mathcal{T}, EK)$. This algorithm encrypts a message M associated with the access policy $\mathcal{T}$, by means of the public parameters *EK*. It returns the encrypted message C, which embeds the given access policy.

- (DK) = **KeyGen**(γ, MK). This algorithm creates a new decryption key associated with the attribute set γ, by means of the master key *MK*. It returns the decryption key *DK*, which embeds the given attribute set.

- $(M$ or $\perp)$ = **Decrypt**(C, DK). This algorithm decrypts a ciphertext C with the decryption key *DK*. It returns the original message M if and only if the access policy $\mathcal{T}$ evaluates to true on the attribute set γ embedded in the decryption key *DK*, otherwise it returns the null value $\perp$.

Security Guarantees

Typically, ABE schemes are proved to be IND-CPA-secure under the standard model or the random oracle model, with the assumption of hardness of the Bilinear Diffie-Hellman (BDH) problem or some other related problem. An exception is the classic CP-ABE scheme by Bethencourt et al. (Bethencourt et al., 2007), which provides a weaker security guarantee under the generic bilinear group model. Note that the IND-CPA security guarantee defends only against passive adversaries. However, cheap transformations (e.g., (Fujisaki and Okamoto, 1999)) are usually applicable to an IND-CPA scheme to obtain an IND-CCA one, which also defends against active adversaries.

Universe Type

The *attribute universe* is the ensemble of all the attributes that can appear in access policies or attribute sets in a particular application. In literature, ABE schemes are traditionally divided into *small-universe schemes* and *large-universe schemes*. A scheme is small-universe if the key authority must fix a finite attribute universe at setup time, i.e., when it executes the **Setup** algorithm.

Small-universe schemes have public parameters that grow linearly with the size of the attribute universe. Typically, the key authority can create new attributes after the setup time, but then it must update the public parameters and deliver them to all the data producers. An example of a small-universe scheme is the work in (Goyal et al., 2006a). On the other hand, a scheme is large-universe if the key authority does not need to fix a finite attribute universe at setup time.

Large-universe schemes have public parameters whose size does not depend on the attribute universe. The attribute universe itself is virtually unlimited, in the sense that data producers can create new attributes at any time without communicating with the key authority. An example of a large-universe scheme is the work in (Bethencourt et al., 2007).

Access Structure Language Expressiveness

Access structure languages define how the attributes in the universe can be combined to express access policies. The expressiveness of an access structure language is a qualitative measure we give to evaluate its capability to express different kinds of access policies. We briefly introduce a few access structure languages sorted by increasing order of expressiveness.

- AND gate on Boolean attributes, which can assume either a positive (e.g., A_+) or a negative (e.g., A_-) value, denoted by the symbol $\text{AND}_\pm$ (Fig. 2.1(a)). All the attributes inside the universe *must* always be included in an access policy.

Therefore, in each policy, the value of any attribute must always be specified (either positive or negative). This language is used and explained, for example, in (Zhang et al., 2014a).

- AND gate with multivalued attributes, denoted by the symbol AND_{m} (Fig. 2.1(b)). It is as expressive as the $\text{AND}_{\pm}$, but it provides a better way to manage attributes. Each attribute in the universe is a group of mutually exclusive values (e.g., A_1, A_2, A_3, ...): one and only one of these values must be in any given policy. This language is used and explained, for example, in (Li et al., 2012).

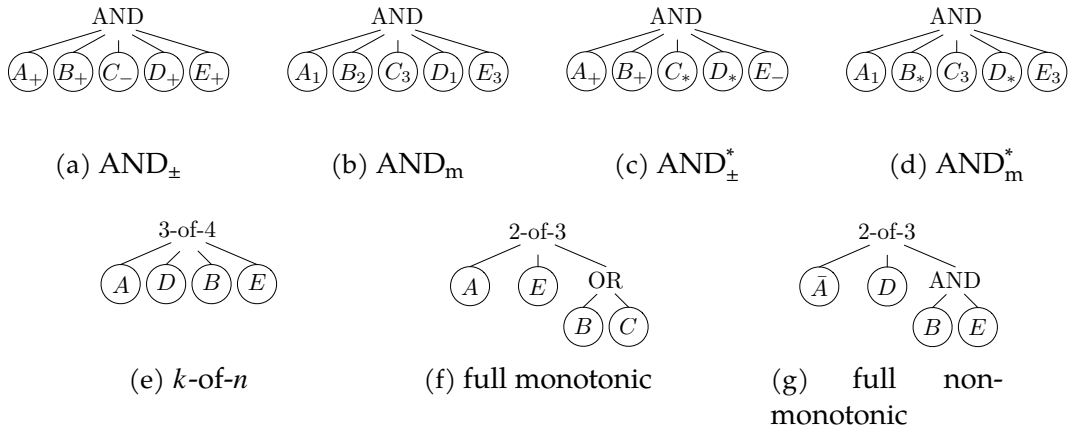
- AND gate on Boolean attributes with *wildcards* (e.g., A_*), and AND gate with multivalued attributes with wildcards, denoted by the symbols $\text{AND}^*_{\pm}$ and AND^*_{m} (Figs. 2.1(c) and 2.1(d)), respectively. The basic idea of a wildcard is a "don't care", meaning that the value of the specific attribute is not relevant to satisfy the access policy. This leads to more effective policies. The $\text{AND}^*_{\pm}$ and the AND^*_{m} languages are used and explained, for example, in (Phuong et al., 2014) and (Zhou et al., 2013), respectively.

- Threshold monotonic language on the presence of attributes in the attribute set, denoted by the symbol k-of-n (Fig. 2.1(e)). A policy is composed of n attributes, where n can be any value from 1 to the total number of attributes inside the universe. For the policy to be satisfied, at least a threshold of k attributes out of n (with $1 \leq k \leq n$) must be present in the attribute set. Note that, from this language on, attributes are not considered Boolean or multivalued variables but rather simple "tags" that can be either present within or absent from the attribute set. This language allows us to implement also AND gates (when $k = n$) and OR gates (when $k = 1$). Intuitively, this language is more expressive than the previous ones since more attribute sets can satisfy a single access policy. This language is used and explained, for example, in (Sahai and Waters, 2005).

- Full monotonic language on the presence of attributes in the attribute set, denoted by the symbol "full monotonic" (Fig. 2.1(f)). This language considers the access structure as a tree, in which the internal nodes are k-of-n gates, and leaf nodes are attributes. Not every attribute inside the universe must appear inside the access policy. This language is used and explained, for example, in (Bethencourt et al., 2007; Goyal et al., 2006b).

- Full non-monotonic language on the presence of attributes in the attribute set, denoted by the symbol "full non-monotonic" (Fig. 2.1(g)). This language includes all the benefits provided by the full monotonic language, plus the ability to express, inside the access structure, the required absence of an attribute

from the attribute set (e.g., $\bar{A}$ is the absence of the attribute A). This language is used and explained, for example, in (Ostrovsky et al., 2007).



(a) $\text{AND}_{\pm}$ (b) AND_{m} (c) $\text{AND}^{*}_{\pm}$ (d) $\text{AND}^{*}_{\text{m}}$

(e) k-of-n (f) full monotonic (g) full non-monotonic

Figure 2.1. Examples of access policies created with different access structure languages. The universe of attributes is $U = \{A, B, C, D, E\}$. In the multivalued access structure languages (Figs. 2.1(b) and 2.1(d)), each attribute can assume one among three distinct values, e.g., the attribute A can assume either the value A_1, A_2, or A_3.

Note that the policies of the AND-based languages ($\text{AND}_{\pm}$, AND_{m}, $\text{AND}^{*}_{\pm}$, and $\text{AND}^{*}_{\text{m}}$) include all the attributes of the universe. With languages with higher expressiveness, one can create access structures that cannot be created using languages with lower expressiveness.

Key Management

To be used in practice, ABE schemes must provide for some additional functionalities of *key management*. In particular, they have to provide mechanisms to distribute decryption keys and to revoke them. While distributing a key to a joining consumer is usually an easy task, key revocation is more challenging, as shown by the rich literature dedicated to the problem (Boldyreva et al., 2008; Attrapadung and Imai, 2009; Yu et al., 2010a; Al-Dahhan et al., 2019; Liu et al., 2016; La Manna et al., 2021; Rasori et al., 2021). An ABE scheme providing for a native key revocation mechanism is called a *revocable scheme*.

It must be noted that any non-revocable scheme can be extended to a revocable one as follows. When a key must be revoked, the key authority runs the **Setup** algorithm, thus creating a new master key and public parameters. Next, for each non-revoked consumer, the key authority generates a new decryption key with the old access privileges. Then, the key authority has two options to confidentially distribute the new decryption keys. It can establish a secure channel with each consumer, e.g., with DTLS. Otherwise, it can encrypt the new decryption keys with the

consumers' public keys, e.g., RSA keys, and store the resulting ciphertexts on the data storage. The latter choice is usually preferable for the key authority since it can rapidly conclude all its revocation operations and go back offline. Therefore, the key distribution task is delegated to the data storage and performed lazily upon data requests by consumers. Note that a consumer will be given more than one decryption key if more than one key revocation happened since its last data request. The key authority also stores the new public parameters on the data storage. Before encrypting a new piece of data, producers must download the latest version of public parameters from the data storage. We call this revocation mechanism, viable for all non-revocable schemes, the *naive revocation mechanism*.

Chapter 3

Attribute-Based Encryption vs. Sticky Policies: What Should We Use?

The spreading and continuous development of Internet of Things (IoT) technologies and services introduces a new way of conceiving and managing the information transmitted over the network (Atzori et al., 2010). The vast amount of data generated and shared every second is in constant increment, thus raising significant scalability issues. One reason for the success of the IoT paradigm is the introduction of miniaturized devices, which can interact and acquire information from the environment they are placed in. Besides such a perk, those devices are often memory- and energy-constrained and, as such, they have a low capability to handle complex data processing and heavy security tasks by themselves.

A critical issue is how the information acquired by such devices, which act as producers, could be shared with the interested consumers. Multiple parties may be involved in the IoT context, thus requiring strict rules regulating access to the IoT resources. In particular, sensitive data must be disclosed only to authorized parties. Infrastructures, both public and private, that use IoT technologies could grow faster by ensuring their customers the reliability and trustworthiness of their data management practices.

In such a direction, two different approaches seem promising in providing an effective solution to the issues above. The first approach is based on ABE since, although more energy consumptive than traditional symmetric or asymmetric cryptography, it allows to make data safely rest or travel over untrusted channels and platforms and, at the same time, enforce fine-grained access control.

The second approach involves the use of sticky policies (Pearson and Mont, 2011), which can be defined by the producer and can travel along with the associated information through the whole data life cycle. Recipients can only retrieve the desired information according to the associated sticky policy, which is evaluated by a trusted authority. Though sticky policies require the trusted authority to be always online to

provide the required decryption keys, it can be very lightweight as it can exclusively leverage symmetric cryptography.

This chapter compares ABE and sticky policies techniques to reveal their advantages or drawbacks in a smart home scenario concerning robustness in terms of reliability and performance (e.g., storage occupancy, CPU load, data retrieval delay). The main goal behind this chapter is to establish the differences in choosing one of the two approaches considering specific application domain's requirements. To this end, both the approaches have been integrated within the same existing flexible and cross-domain middleware, named *NetwOrked Smart object* (*NOS*). NOS is an IoT platform, originally conceived to manage data generated by heterogeneous sources and share them with interested parties, adopting specific algorithms and protocols (Sicari et al., 2016a). Note that an enforcement framework based on sticky policies is already available for the NOS architecture, as presented in (Sicari et al., 2017b). Instead, in this chapter, a specific type of ABE, named Ciphertext-Policy Attribute-Based Encryption (CP-ABE) (Bethencourt et al., 2007), has been integrated within the NOS system for comparison purposes. CP-ABE is also considered due to its similarities with the approach based on sticky policies. To give some preliminary details, note that, in the CP-ABE paradigm, as for sticky policies, the access rule resides within the encrypted data itself, while the attributes used for evaluating the policy are directly associated with decryptors.

To summarize, the main contributions proposed in this chapter are the following ones:

- CP-ABE scheme has been integrated within NOS architecture. Note that the NOS platform has been chosen due to its modular architecture, enabling it to adapt its behavior dynamically; thus, it is particularly suitable for new functionalities. Moreover, an implementation of sticky policies in the NOS platform already exists (Sicari et al., 2017b).

- the behavior of CP-ABE and sticky policies approaches has been compared to reveal their potentialities and weaknesses from a functional point of view, following the data flow management in a not-fully-trusted environment, as it happens in the typical IoT contexts. The comparison is carried out in a smart home scenario, where different IoT devices produce heterogeneous data, from video streams to electrical data sets. Such a kind of scenario also enables the presence of different kinds of users. In this way, a simple yet accurate case study is provided, which could be further expanded for future analysis. The aforementioned smart home data-set is considered, and the following metrics are analyzed and measured: storage occupancy, CPU load, and data retrieval delay.

- the performance of the employed CP-ABE scheme with respect to the sticky policy method has been evaluated using a real test-bed. The main outcomes reveal that the sticky policy approach is more efficient in terms of CPU load, but its storage occupancy and data retrieval delay are higher than those of the CP-ABE approach.

The remainder of the chapter is structured as follows. Section 3.1 presents the preliminaries of the proposed work, which include the sticky policy and the basic NOS architecture. In Section 3.2 the related work is presented. Then, Section 3.3 presents the integration of CP-ABE functionalities into the NOS middleware, along with the comparison between the CP-ABE approach and the one based on sticky policies. Section 3.4 presents the threat model, the smart home application scenario, and the performed experiments. Section 3.5 ends the chapter, answering the question inquired.

3.1 Preliminaries

In this section, the necessary preliminaries for clearly understanding the mechanisms related to adopting sticky policies for securing access to the information transmitted within an IoT system are detailed. Moreover, a sketch of NOS architecture is presented.

NetwOrked Smart objects architecture

Two main entities compose a typical IoT system: (i) the data producers, conceived as heterogeneous data sources (e.g., WSN, RFID, NFC, actuators, etc.) which generate data to be sent to the IoT platform; (ii) the data consumers, who interact with the IoT platform through services making use of such IoT-generated data, typically accessing them by means of a mobile device (e.g., smartphone, tablet) connected to the Internet, through WiFi, 3G, or Bluetooth technologies.

In such a scenario, NetwOrked Smart objects' (NOS) middleware (Sicari et al., 2016a) has been conceived as a layered architecture, providing lightweight and flexible functionalities; it represents a comprehensive approach for managing data gathered from heterogeneous sources in a distributed way and for providing customized services to users, assessing security as well as data quality requirements.

Proper interfaces for the communications of NOSs with the data producers and consumers have been defined. The HTTP protocol is usually adopted for collecting data from IoT devices. For each incoming data, we gather the following pieces of information:

- the kind of data producer, which describes the type of IoT source;

- the communication mode, that is, how the data is collected (e.g., discrete or streaming communication);
- the data schema, which represents the type (e.g., number, text) and the format of the received data;
- the data content;
- the reception timestamp.

Instead, Message Queue Telemetry Transport (MQTT) protocol (Hunkeler et al., 2008) is used for disseminating the information to the interested data consumers. To this end, a topic is assigned by NOSs to each processed data. NOSs also provide a lightweight and secure information exchange process, based on an authenticated publish and subscribe mechanism (Rizzardi et al., 2016), integrated with the MQTT protocol.

A scheme of NOS architecture is sketched in Figure 3.1, along with its current integration with sticky policy enforcement framework (Sicari et al., 2017b), which is described in the following section.

Sticky Policies

The sticky policy paradigm was first proposed by Karjoth, Schunter, and Waidner (Karjoth et al., 2002). Sticky policies are transmitted along with the data they refer to throughout the entire data life cycle. Specifically, Sticky Policies allow us to define the following aspects:

- the owner of the data;
- the data content, possibly encrypted;
- the scope of the data;
- where and when data will be available;
- specific obligations and restrictions.

In detail, the concept of sticky policy is to attach security and privacy policies to owners' data and drive access control decisions and policy enforcement. Sticky policies allow specifying access rules in a fine-grained manner: in principle, every data unit could have its unique policy. Furthermore, as policies 'travel' with the data across the entire system, they could protect the entire data life cycle. Such an approach has been mainly introduced for security and privacy enforcement: when submitting data to a consumer, a user consents to the applicable policies selecting the proper preferences.

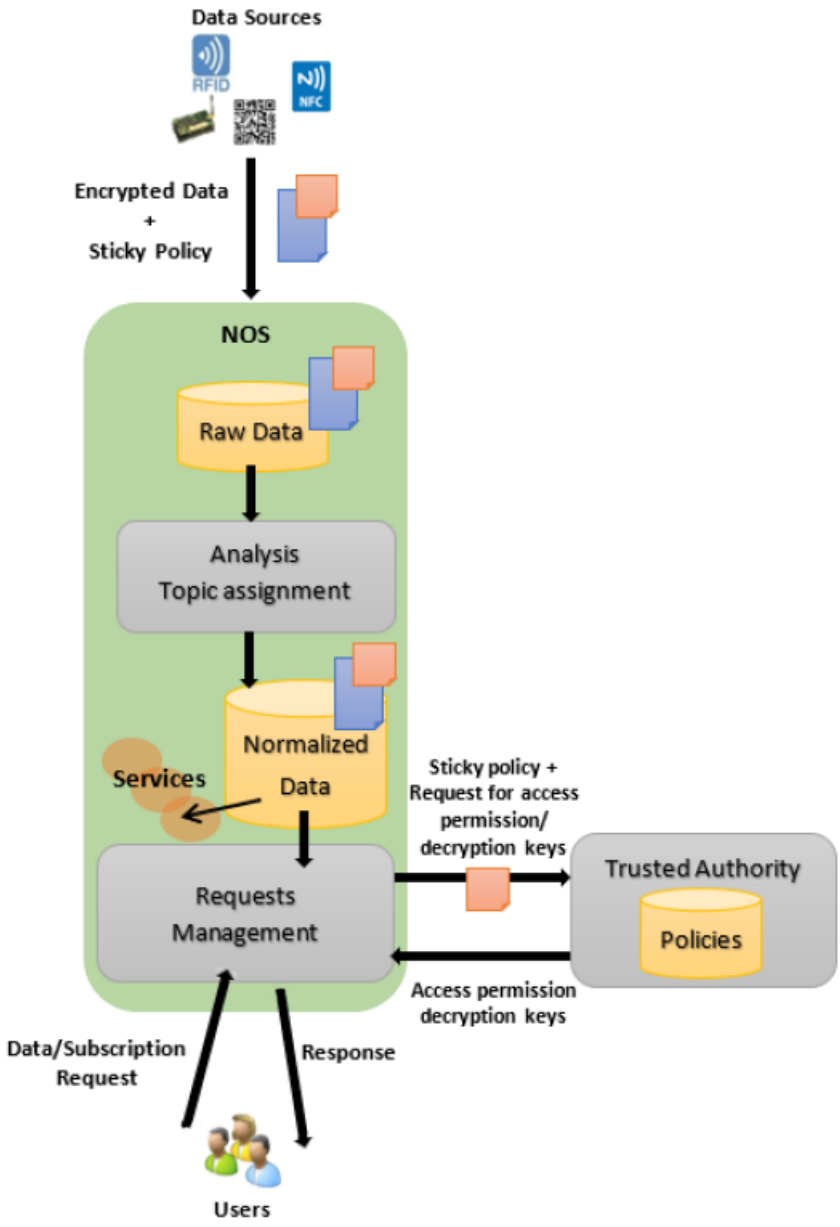


Figure 3.1. NOS data flow with sticky policies.

Such features are particularly interesting in some scenarios, like that of IoT, where users' or businesses' confidential information may flow across organizational boundaries (Pearson and Mont, 2011). For example, social networks may share some information with marketing companies; similarly, cloud applications may transfer data among different realms depending on a need. Such situations represent well-known open issues in the field of security and privacy enforcement.

The sticky policy concept has already been integrated into the NOS platform, as presented in (Sicari et al., 2017b). Note that NOSs own no policies/credentials because an external *Trusted Authority* (TA) is responsible for their management. The data producer sends them in an encrypted way along with the associated sticky policy to NOS; clearly, data producers have in-depth control over their information flow since they are responsible for the access rules. Then, each NOS can contact the TA to obtain the access permissions on the received data when there is the need to disclose them to interested consumers (i.e., the users who interact with the IoT platform). In this way, no synchronization or policy sharing is required among multiple NOSs, since the TA manages the access permissions. Figure 3.1 summarizes the just described behavior, as anticipated in Section 3.1.

3.2 Related Work

Typically, current proposals addressing security and privacy issues in the IoT focus on data communications by enforcing data exchanges according to strict protection constraints, considering, at the same time, the heterogeneity of devices and communication technologies. Devices can be characterized by different protocols. For example, many smart devices can natively support IPv6 communications (Palattella et al., 2013) (Bagci et al., 2013), while other existing deployments might not support the IP protocol within the local area scope, and this requires the design of ad-hoc gateways and middleware (Boswarthick et al., 2012). This is the reason for introducing NOS middleware in the envisioned solution.

Relevant contributions on security-oriented IoT middleware include: VIRTUS (Conzon et al., 2012), which relies on the open eXtensible Messaging and Presence Protocol (XMPP) to provide secure event-driven communications; Otsopack (Gòmez-Goiri et al., 2014) and Naming, Addressing and Profile Server (NAPS) (Liu et al., 2013a), which are data-centric frameworks based on the usage of HTA and REpresentational State Transfer (REST) interfaces. With respect to such frameworks, NOS is more recent and adopts a lightweight technology based on Node.js in an event-driven fashion, which perfectly fits the requirements of IoT applications. Also, various projects have the final purpose of delivering a framework able to dynamically integrate user data (e.g., location, behavior) in privacy and security protocols, as reported in (Sicari et al., 2015).

The IoT middleware named NOS, firstly implemented and presented in (Sicari et al., 2016a), tried to fill the gap by providing efficient processing and assessment of the IoT data. Such functionalities have been further coupled with a policy enforcement framework based on sticky policies (Sicari et al., 2017b), and relevant security requirements have been addressed, as detailed in Section 3.1. The novelty introduced in this chapter is the ABE paradigm's integration into NOS and its comparison with the sticky policy based approach. It is worth remarking that the presented solutions based on sticky policies and CP-ABE mechanisms are conceived to include multiple NOSs. They can easily and securely share data, acting as intermediaries with each other. A specific application domain could require a mechanism for policies' synchronization. With this regard, a solution able to synchronize the policies among different NOSs has already been provided in (Sicari et al., 2017a).

Regarding ABE, some IoT-focused cryptographic schemes (Yu et al., 2011; Yao et al., 2015; Odelu et al., 2017) and architectures (Picazo-Sanchez et al., 2014; Singh et al., 2015; Rasori et al., 2018; Hernández-Ramos et al., 2018; Rasori et al., 2020) can be found in the literature. In (Yu et al., 2011) the first KP-ABE scheme for Wireless Sensor Networks (WSNs) is presented. The proposed scheme is composed of one trusted network controller, several users, and several sensor nodes. Each user owns a secret key generated by the network controller, according to a policy that describes the type of data he/she can access. Each sensor node is pre-loaded with a set of attributes and their relative public quantities generated by the network controller. In (Yao et al., 2015) a lightweight KP-ABE scheme for the IoT is presented. The math behind the proposed scheme is based on elliptic-curve cryptography rather than pairing-based cryptography as the majority of the other ABE schemes. This makes the scheme more efficient from the point of view of encryption and decryption times. In (Odelu et al., 2017) a CP-ABE scheme allowing for constant-size keys and cipher-texts is presented. This makes the scheme more scalable, especially in battery-limited devices and bit-rate-limited channels, as in the typical IoT application. The scheme allows only AND operators to be used in the Boolean formulas of the policies, so it provides for limited expressiveness. The above cryptographic schemes are unsuitable to be used in the present chapter, which compares CP-ABE and sticky policy approaches. This is either because they follow a KP-ABE approach instead of a CP-ABE one ((Yu et al., 2011; Yao et al., 2015)), or because they provide no security at all (the work in (Odelu et al., 2017) has been shown to be insecure (Herranz, 2020)).

In (Picazo-Sanchez et al., 2014) a secure publish-subscribe protocol for medical Wireless Body Area Network (WBANs) using ABE is proposed. The conceived architecture follows a star-topology network, where a smartphone (or a similar device) manages the communication among various nodes placed over/inside the user's body, monitoring his/her health conditions. Each node can publish its data

and subscribe to data generated from other nodes. In (Singh et al., 2015) a secure MQTT for IoT is introduced, along with the possibility of using ABE. The proposed architecture comprises one Public Key Generator (PKG), one broker, and several devices, which can act both as subscribers and publishers. Each device owns the public key and a secret key associated with some attributes that describe its features. Then, each device subscribes to specific topics to receive the data of interest. In (Rasori et al., 2018) a system for smart cities using ABE is presented. The application offers a service of real-time road monitoring in a smart city scenario. Smart objects (e.g., cameras) are placed along the roads and store their sensed data on a cloud storage service. Users can pay a subscription and obtain an ABE decryption key to retrieve and decrypt the video streams of the city traffic in real-time. In (Hernández-Ramos et al., 2018) a system for protecting location data in smart buildings using CP-ABE is presented. Their approach is based on the concept of "bubbles", which are coalitions of smart objects defined according to relationships between their owners.

As emerged, ABE schemes are not widely adopted in IoT scenarios yet. For such a reason, the analysis conducted in such a chapter contributes to assessing ABE capabilities, feasibility, and potentialities within an IoT middleware in an IoT typical scenario.

3.3 Integration of CP-ABE into NOS architecture and comparison with sticky policies

Generally, an enforcement framework is composed of the following main standard elements: (i) a *Policy Enforcement Point* (*PEP*), which intercepts the access requests and queries the PDP about its acceptance; (ii) a *Policy Decision Point* (*PDP*), which evaluates the access requests against the authorization policies and takes the authorization decisions; (iii) a *Policy Administration Point* (*PAP*), which contains the complete set of authorization policies established by the system's administrators. In a previous NOS version, such components are all located into NOS (Sicari et al., 2016b).

By introducing the sticky policies (Section 3.1), only the PEP is located into NOSs, while the PDP is located within the TA, as the PAP. Consequently, the role of NOSs in the enforcement process is softened, and NOSs can no longer be considered a single point of failure in the security of the information transmitted within the whole IoT system. By delegating some operations and controls to the TA, the overall efficiency of the NOSs middleware has been improved, as demonstrated in (Sicari et al., 2017b).

However, the main drawback that emerged from the sticky policies approach is the need for an always-online TA, responsible for trustworthy evaluating the poli-

cies in relation to the subscribing data consumer. To overcome such an issue and, at the same time, to provide a flexible and efficient data access control framework, the CP-ABE scheme is introduced. It includes a mechanism for access control able to embed PEP and PDP just inside the cipher-text (hence, the policy is not separated from the encrypted data itself, as it is for sticky policies). The existing NOS architecture, described in Section 3.1, must be revised in order to include the new components/functionalities required by the CP-ABE scheme.

In Figure 3.2, the modified NOS architecture is shown, including the CP-ABE primitives, which are integrated into the data flow. More in detail, concerning Figure 3.1, it is worth noting that the policy associated with the data now depends on the CP-ABE encryption (the Encrypt primitive is executed by the new introduced *CP-ABE Data Encryption* module). The *CP-ABE Data Encryption* module has three main goals: (i) it performs the encryption in place of data producers, thus lightening the memory- and energy-constrained IoT devices from such a complex and expensive task; (ii) it defines and correctly associates the access policies to the processed data; (iii) it stores normalized data in the *Normalized Data* storage unit in an encrypted form, so such data at rest is also protected in the case of NOS compromised by unauthorized parties. The *CP-ABE Data Encryption* module combines the policies defined by NOS and the encryption keys defined by the TA, which executes the Setup and KeyGen primitives. In this sense, data producers have less control over the disclosure of their information, but the IoT platform has more control over them. Another remark is that the TA now needs to communicate with both NOS, in order to provide the required encryption key for performing the encryption task, and users, to disclose the decryption key to authorize them to access the NOS's resources; hence, the Decrypt primitive is executed by the data consumer. After that, the TA can go offline.

To show the differences between the NOS' data flow with the sticky policies approach and with the CP-ABE scheme, a further overview of the two systems is provided in Figures 3.3 and 3.4, respectively.

Figure 3.3 highlights that in the sticky policies approach, for each subscription to a specific topic by an interested consumer (step 10), the TA must be contacted by NOS to make the access decision (steps 11-13); then, if the TA agrees to the subscription, the data consumer will be notified of the data belonging to the requested topic (step 14). The credentials used for exchanging such data between NOS and the consumer are established *a priori* by an agreement between them (steps 1-2). Hence, a sort of double agreement should be done: one for the encryption key and another one for the topic's subscription. The regular NOS's processing activity is independent of such tasks (steps 3-9).

Instead, in a scenario that adopts CP-ABE (Figure 3.4), the TA is no longer required to be online during subscription and data transmission (steps 8-9). NOS is

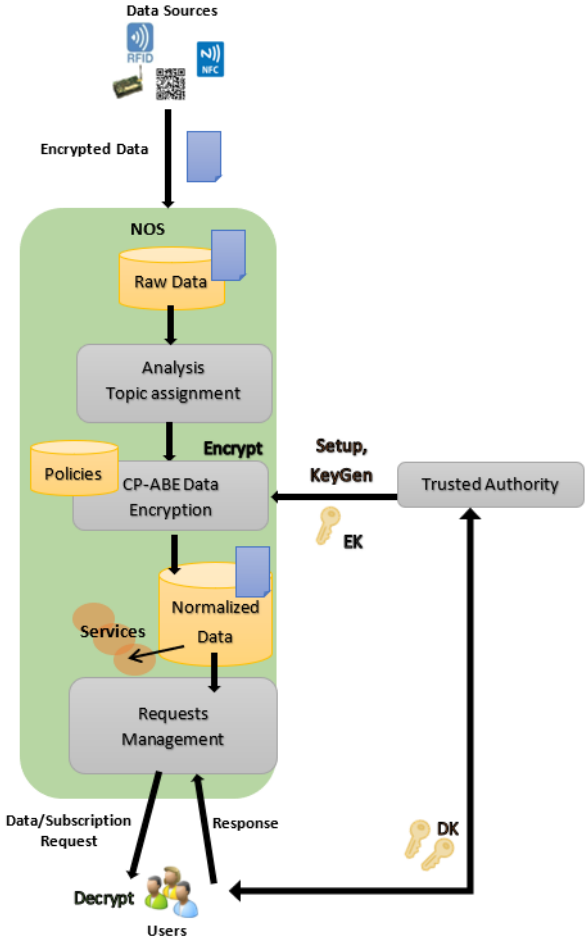


Figure 3.2. New proposed NOS architecture.

not required to know the decryption keys needed by the consumers to decrypt the information (step 10). This lightens the keys' management from the NOS's viewpoint. In fact, with CP-ABE, the access policy is embedded in the encrypted data based on the assigned attributes (steps 1-7). Such an aspect represents the crucial difference between the two approaches, which reveals the effectiveness of the CP-ABE approach in facilitating, from a performance perspective, the whole management of data encryption/decryption in relation to the established policies. In that sense, sticky policies seem more challenging to manage because encryption/decryption are not so related to access policies with respect to the CP-ABE paradigm.

In case of policies' update, addition, or revocation, the sticky policy approach requires an update of the scopes and constraints of the TA; hence, the related decryption keys must be revoked and re-assigned to the consumers involved in that policies. On the other hand, the CP-ABE mechanism simply requires that NOSs change the policy used to encrypt data, which can be done without involving the TA.

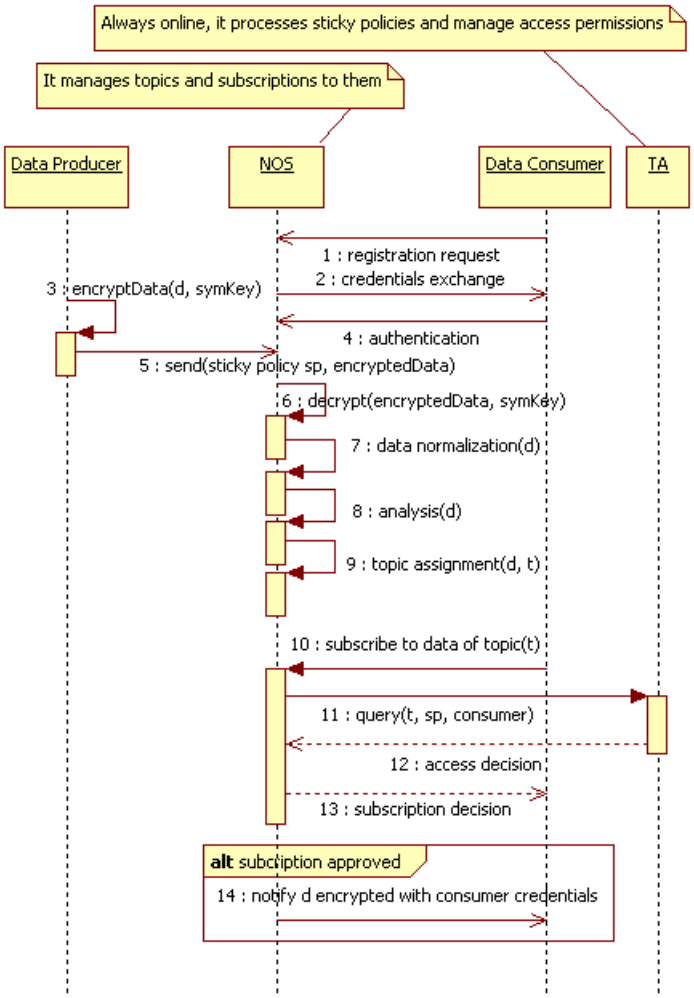


Figure 3.3. Scheme of sticky policies based data flow within the NOS system.

To summarize, the CP-ABE Encrypt, Setup and KeyGen primitives are not so complex to be integrated into the NOS platform due to its modular design. In fact, the introduction of the new module *CP-ABE Data Encryption* does not affect the behavior of the existing ones, as it emerges in Figure 3.4; moreover, communications with the TA were already available in the previous version with sticky policies (Sicari et al., 2017b). The main difficulty is represented by the management of the decryption keys concerning the consumers subscribed to NOS. The following section focuses on the key management mechanism to clarify how decryption keys are distributed and assigned and how decryption is enabled.

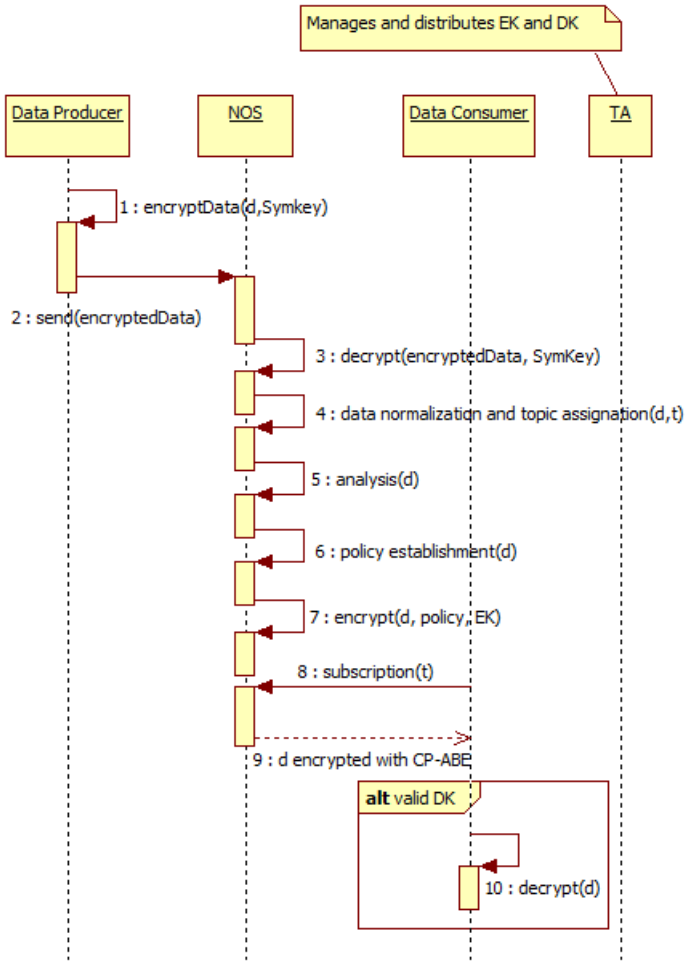


Figure 3.4. Scheme of CP-ABE based data flow within the NOS system.

Key Management

In a system adopting CP-ABE, the decryption keys must be distributed to the data consumers and revoked if they get compromised somehow. The mechanisms of distribution and revocation of decryption keys are often critical and complex. To implement such mechanisms, some additions to the basic Bethencourt's scheme have been introduced in NOS.

A version number is associated with each attribute, and inside the TA an *Attribute Version List* (AVL) is implemented, which is a list of all the attributes used in the system together with their latest version number. The AVL is created by the TA just after the execution of the Setup procedure, and all the version numbers are initialized to 1. From now on, an attribute is intended as including its version number. For example, the attribute "*attr*" will become "*attr_vn*" where *n* is the version number, and "*attr_v*1" and "*attr_v*2" are considered two distinct attributes. Further-

more, the TA also detains a table of the unique identifiers of all the data consumers ($ConsID$) and their attribute sets, named *Consumers Attribute List* (CAL). This table is updated every time a new data consumer joins the system through a subscription to a certain topic since the IoT platform runs on a publish&subscribed sharing, as described in Section 3.1.

It is supposed that each data consumer and the TA have their own pair of asymmetric keys (e.g., RSA or ECC) used for digital signature and encryption. Furthermore, it is assumed that the TA's public key is well known to all NOSs and data consumers, possibly obtained offline. Hence, the following procedures for key management are defined:

- *System initialization*. In the *system initialization* procedure, the TA runs the CP-ABE primitive Setup, thus generating the couple (MK, EK). The TA has the responsibility to keep MK secret. Then, the TA creates the AVL by inserting all the attributes used in the system along with their version. Finally, the TA also creates the CAL, which is empty at the system initialization.

- *NOS join*. The *NOS join* procedure is executed whenever a new NOS joins the system. The TA signs and communicates EK and the AVL to the NOS. The NOS can encrypt data using only attributes contained in the AVL.

- *Consumer join*. The *consumer join* procedure is executed whenever a new data consumer joins the system. The data consumer requests a decryption key to the TA by declaring an attribute set γ that describes him/her. The TA has the responsibility to verify that the declared γ describes the consumer. Every attribute inside γ must belong to the AVL maintained by the TA. Then, the TA executes the CP-ABE primitive KeyGen, generating a decryption key based on the previously mentioned γ. The TA updates the CAL by adding a tuple including the $ConsID$ of the consumer and its associated attribute set. Then, the TA signs the decryption key with its private key and encrypts the signed decryption key with the consumer's public key, which may have been acquired offline. The TA sends such signed and encrypted decryption key to the new consumer, which decrypts and authenticates it. If both operations are successful, the consumer accepts the decryption key and starts using it to decrypt data.

- *Data producer deployment*. The *data producer deployment* procedure is executed whenever a new data producer is installed in the system. The data producer agrees with the associated NOS on a symmetric key, with which all the subsequent messages will be encrypted. Such a key agreement could be done in several ways, depending on the capabilities of the specific data producer. For example, NOS can transmit the symmetric key in clear to the data producer with

a low-power wireless signal. This is a lightweight technique recommended by some of IETF's RFCs (Baccelli et al., 2016) for smart home applications. It assumes that no eavesdropper is present at deployment time. If this assumption does not hold, more advanced key agreement protocols can be used, for example, anonymous or authenticated Diffie-Hellman.

- *Key revocation*. The *key revocation* procedure is executed whenever a decryption key is compromised. This procedure drastically reduces the risk of data leakage by invalidating and making the compromised decryption key useless. In order to ease the reading, the key revocation procedure is explained through an example in the following.

Suppose that the decryption key DK_2 of a consumer identified by *Cons2* has been compromised and must be revoked. The attribute set γ_2 associated with the decryption key includes the attributes A_v1, D_v1. To revoke DK_2, the TA updates the AVL by an increment in the version number of all these attributes, thus updating A_v1 to A_v2, and D_v1 to D_v2 (Figure 3.5).

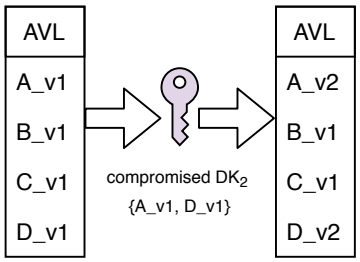


Figure 3.5. Example of AVL update during a key revocation procedure. On the left, the AVL before the procedure of key revocation. In the middle, the key that has been compromised. On the right, the updated AVL.

Then, the TA proceeds with re-generating the decryption keys of the *affected consumers* by executing the CP-ABE primitive KeyGen. The affected consumers are those consumers that have at least one attribute in common with the revoked decryption key. Let us suppose that the decryption key of the consumer identified by *Cons1* has one attribute (A_v1) in common with DK_2. Such a consumer is thus an affected one, and the TA re-generates his/her decryption key. The TA also updates the CAL table (see Figure. 3.6) by removing *Cons2*, whose decryption key has been revoked, and by upgrading A_v1 to A_v2 in the attribute set of the affected consumer *Cons1*.

Then, the TA proceeds to sign, encrypt (with the consumer's public keys), and send the re-generated decryption key to each affected data consumer. Such an operation guarantees future decryption for affected data consumers; otherwise, their old decryption keys will not decrypt new cipher-texts. Since the decryption key is

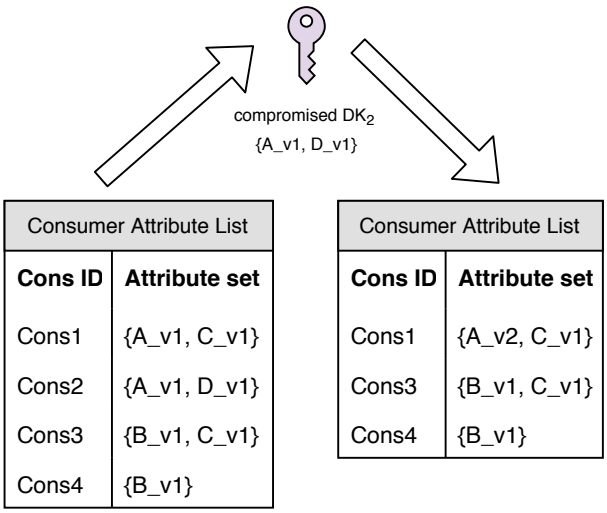


Figure 3.6. Example of CAL update during a key revocation procedure. On the left, the table before the key revocation. On the right, the table after the key revocation.

encrypted, the TA can send it through an insecure channel (e.g., a simple email). Finally, the TA signs the updated AVL and sends it to the joined NOSs. From this moment, NOSs will encrypt with the new versions of the attributes. Such an operation makes the compromised decryption key useless because the old version A_v1 and D_v1 are no longer used to encrypt data. To send the updated AVL, NOSs can do an MQTT subscription to the broker on a special-purpose topic dedicated to AVL updates from the TA, as just done in (Sicari et al., 2017a). In this way, the TA can send a single AVL update to the broker, and the broker will eventually distribute it to all the NOSs.

For each key revocation, supposing n consumers and m NOSs in the system, the TA must send a single AVL update and $a \cdot n$ emails, where $a \in [0, 1]$ is the ratio of affected consumers. Such a ratio highly depends on the policy complexity. The authors in (Rasori et al., 2018) computed that, in a large IoT system, the affected consumers of the average key revocation can be about $a = 12\%$ of the total consumers. Sending such a quantity of emails should not be a problem with state-of-the-art bulk email software, given that key revocations should be rare events.

Note that, while data encrypted *after* the key revocation procedure will not be decryptable by the revoked key, data encrypted *before* may still be accessible.

3.4 Validation and Experiments

For evaluating the approaches, just compared in Section 3.3, a threat model, an application scenario related to a smart home, and a test-bed for simulations are firstly presented. Then, numerical results concerning the following metrics are provided:

storage occupancy, CPU load, and data retrieval delay.

Threat Model and Security Analysis

We assume that each NOS has a copy of a trusted certification authority's public key. The TA owns a certificate released by such a CA. We then assume that each NOS knows the public key of the TA, which is used for digital signature, through the use of certificates. If the TA is able to issue certificates, it can also act as a CA.

The first threat considered is related to the violation attempts performed by malicious external parties. An external party acts from the outside of the IoT system, and he does not own any decryption key. He intends to access encrypted information. In order to do so, he can try to eavesdrop on a decryption key during a consumer join/subscription procedure or a key revocation procedure. Such an attack is avoided because, in both procedures, the decryption keys are encrypted with the consumer's public key. Alternatively, he can try to carry out an active *Man In the Middle* (MITM) attack. For CP-ABE, when the TA communicates the encryption key to the new joined NOS, during the NOS join procedure, the attacker can try to impersonate the TA and communicate to the NOS a malicious encryption key so that he can decrypt all the ciphertexts produced by that NOS. Such an attack is avoided because the TA digitally signs the encryption key. Similar is the sticky policy paradigm case, where, instead, the TA only communicates with NOSs, thus reducing the vulnerabilities. Hence, both approaches (i.e., CP-ABE and sticky policies) are robust for such a kind of attack.

The second threat considers an external party that compromises a NOS. The effects of such an attack are many. First and foremost, the attacker has access to all the data that the smart objects will produce (and consequently send to the NOS) from that moment on. The attacker cannot access past data encrypted with CP-ABE and stored in the NOS. However, the same cannot be said for past data encrypted with sticky policies and stored in the NOS since it is symmetrically encrypted, and thus the decryption key is known to the NOS. Secondly, the compromised NOS can manipulate the data it receives from the smart objects. Now we analyze the system's response once the compromise has been solved and the security hole that allowed it had been patched. In both approaches, the symmetric key used by each sensor managed by the NOS must be renewed since they are also stored inside the NOS and considered compromised. If the sticky policy approach is used, all the subscribers associated with the attacked NOS have to renew their credentials. Since the subscriber credentials are stored in the NOS, they must be considered compromised by the attacker. Instead, if the CP-ABE approach is used, no additional cryptographic value has to be considered compromised. As a matter of fact, the NOS possesses only the encryption key, which is public, and therefore it is of no use for the attacker.

The third threat concerns possible colluding consumers wanting to acquire data that they cannot obtain singularly. Concerning this attack, the original Bethencourt's CP-ABE scheme (Bethencourt et al., 2007) is natively collusion-resistant. This means that two or more consumers cannot combine their decryption keys in such a way to decrypt data that they cannot access singularly. Please refer to (Bethencourt et al., 2007) for mathematical proof of this.

The CP-ABE scheme must be indistinguishable under the adaptive chosen ciphertext attack (IND-CCA) to cope with the threats described above and, therefore, to resist active adversaries. Moreover, the signature scheme must be unforgeable under the chosen message attack (EUF-CMA). As the signature scheme, we chose the ECDSA algorithm, which offers the needed security requirement. The original CP-ABE scheme that we employed (taken from the work of Bethencourt et al., (Bethencourt et al., 2007)) is only proved to be indistinguishable under the chosen-plaintext attack (IND-CPA). The proof of that is given by Bethencourt et al., and it is supported by the complexity of the Bilinear Diffie-Hellman (BDH) problem. For being suitable against active adversaries, we converted the IND-CPA in an IND-CCA scheme by applying the efficient and straightforward Fujisaki-Okamoto transformation (Fujisaki and Okamoto, 1999), which only requires the random oracle model assumption.

The fourth threat, instead, regards only the sticky policies, and it concerns an active adversary that performs a MITM attack between the NOS and the TA during a decryption request. In this case, the adversary may change the sticky policy that travels through the internet. To avoid this attack, the NOS shall communicate with the TA (and vice-versa) only through a secure channel (e.g., TLS).

Smart Home Scenario

An application scenario related to a typical smart home is used for conducting the performance evaluation, presented in Section 3.4. Data from real-world smart home test-bed has been gathered[1]; such data regards some smart meters installed in two houses, named *A* and *B*, which include, among the others, the electricity consumption related to: kitchen lights, bedroom lights, duct heater HRV, and HRV furnace. Note that the house has a total of eight rooms and includes three full-time occupants. Measures are acquired through installed smart objects that collect electricity data every minute. Detailed information about such a smart home data-set and on how information is thereby collected are available in (Barker et al., 2012).

Each person, which interacts with the houses, can be described by one or more of the following attributes:

[1] `http://traces.cs.umass.edu/index.php/Smart/Smart`

- Landlord of the house X $(LanX)$, who is the house's landlord, but it does not imply that he/she lives there. The landlord might rent out the house. For example, the landlord can be a young man that has rented out an inherited apartment.
- Tenant of house X $(TenX)$, who manages and lives in the house. The tenant has access to all the data generated in the house. The tenant and the landlord role may coincide. For example, a young woman who has recently bought a house and moved in is both tenant and landlord. Such an attribute is intended as a numerical, representing the date when the person was nominated tenant.
- Guest of the house X $(GueX)$, who has access to the house, and he/she may not live there. For example, an old couple's daughter can live elsewhere, but she has Guest rights to check on her parents. He/she has access to a limited number of data. Such an attribute is intended as $TenX$, and it represents the date when the person was nominated guest.
- Expiring date for the tenant role of house X $(ExTenX)$; it is also intended as a numerical attribute, as for $TenX$ and $GueX$, representing the date when the role of tenant will expire.
- Expiring date for the guest role of house X $(ExGueX)$; it is also intended as a numerical attribute, as for $ExTenX$, and it represents the date when the role of guest will expire.

An example of an attribute set γ for a person named Robert (R) is the following:

$$\begin{aligned}\gamma(R) = \{&LanB,\\ &TenA = 2/2/2000,\\ &ExTenA = 2/2/2020,\\ &TenB = 2/2/2015,\\ &ExTenB = 2/2/2020\}\end{aligned} \tag{3.1}$$

The above statement must be intended as follows: (i) Robert is the landlord of the house B; (ii) he is the tenant of house A since February 2^{nd} 2000 and of the house B since February 2^{nd} 2015; (iii) both his tenant roles will expire on February 2^{nd} 2020. In such a scenario, the versioning of attributes is not considered for readability.

Three possible data requests for each house are made available, even obtained from the data-set mentioned above:

- Access to the electrical data-set: this is a data-set related to the energy consumption of all the electronic and electric devices inside the house. Only the

landlord and the tenants can access this data. To access them, a viable policy could be:

$$\begin{aligned}\mathcal{T}(ElectricalDataset) = \{&LanX \vee \\ &(\text{today} \geq TenX \wedge \text{today} \leq ExTenX)\},\end{aligned} \tag{3.2}$$

where today is the date when data has been produced. Note that, due to the way in which numerical attributes are implemented in (Bethencourt et al., 2007), the $\geq$ and $\leq$ operators return "false" in the case the numerical attribute does not exist in the decryption key.

- Video streaming: it provides live images from the inside of the house. Only who has actual access to the house can see video streaming from it. To request such a kind of data, the consumer must be an authorized as an tenant or a guest. Therefore, a viable policy could be:

$$\begin{aligned}\mathcal{T}(VideoStream) = \{&(\text{today} \geq TenX \wedge \\ &\text{today} \leq ExTenX) \vee \\ &(\text{today} \geq GueX \wedge \\ &\text{today} \leq ExGueX)\}.\end{aligned} \tag{3.3}$$

- Remote monitoring of house's current state: this implies the monitoring of relevant parameters such as temperature, humidity, lights switched on/off. Only the tenant can remotely monitor the status of the smart home. A viable policy could be:

$$\begin{aligned}\mathcal{T}(Monitoring) = \{&(\text{today} \geq TenX \wedge \\ &\text{today} \leq ExTenX)\}.\end{aligned} \tag{3.4}$$

The examples of policies just presented are derived from the attributes defined above. They will be used for the performance evaluation in Section 3.4.

Performance Evaluation

In the experimental setup, the NOS platform is deployed on a Raspberry Pi, a device widely used in IoT applications. The behavior of a set of consumers subscribing to obtain information about the smart homes (see Section 3.4) is emulated using a laptop, with the following features: (i) Core i7-4710HQ 2,5 GHz; (ii) 16 gigabytes of RAM; (iii) OS Ubuntu 16.04. The laptop uses WiFi IEEE 802.11 network to communicate with the Raspberry Pi. The same WiFi connection is also used for the communications with the MQTT broker and with the TA module, implemented as

separate components, which interact with NOS on-demand and run on separate laptops. A toolkit available online[2] has been used to implement the required CP-ABE primitives into the IoT system, as presented in Section 3.3.

Sticky policy and CP-ABE approaches are compared w.r.t. the following metrics: storage and CPU load overhead and data retrieval delay. The obtained results are compared based on the application scenario, defined in Section 3.4. More in detail, one packet per minute is fetched from the simulated data sources, and one data request per minute is simulated from the consumers. The number of data producers and consumers is set to six and three, respectively; such values are derived from the simulation setups of two previous works on policy enforcement within the NOS architecture (Sicari et al., 2016b) (Sicari et al., 2017b), in order to ease the results' comparison and evaluation. Table 3.1 summarizes the setup parameters, while Figure 3.7 sketches the interactions among the participants to the smart home scenario and the NOS platform. Note that bold text and arrows denote the interactions valid for CP-ABE, while the dashed arrow denotes the interactions valid for sticky policies; finally, thin arrows are common to both approaches.

Table 3.1
Experimental Configuration

Parameter	**Value**
Number of data producers	6
Number of data consumers	3
Number of attributes per policy	5
Data-rate provision from producers	1 pck/minute
Requests' data-rate from consumers	1 request/minute
Duration of the experiments	1 hour
Time window of the data gathered from the data-set	1 week

Storage, Network and CPU Load

NOS components have the following storage requirements, which are different in the two approaches:

- In the sticky policy approach, the data sources and the consumers must store the credentials for ciphering the data to be transmitted to NOS. When producers transmit data to NOS, they may also send the related sticky policy. Such an aspect unavoidably causes an increase in traffic into the network since it is

[2] `http://acsc.cs.utexas.edu/cpabe/`

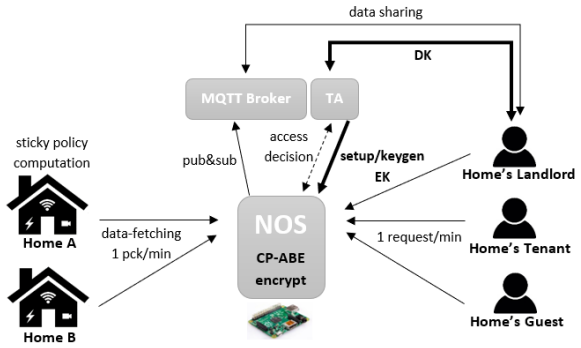


Figure 3.7. Scheme of the performance evaluation setup.

transmitted not only the data but also the associated policy. An average increase of 0.5 kilobytes is measured for each transmitted data unit, considering the sticky policy format specified in (Sicari et al., 2017b), which approximately consists of 500 bytes. Whereas adopting an approach based on CP-ABE, data sources have not to send a sticky policy along with the data to NOS; therefore, such an increment is negligible. Indeed, such an aspect represents a relevant advantage of adopting CP-ABE because IoT networks usually transmit a huge amount of data. Note that, in CP-ABE, the packets' dimension increases once NOS has performed the encryption task.

- Starting for such premises, it is worth noting that the described behavior also influences the network load. In fact, for both the approaches, the information which is transmitted over the network are: (i) the data from producers; (ii) the consumers' requests; (iii) the consumers' responses (i.e., the data release). Figure 3.8 shows a reduced network load when adopting CP-ABE, and it is mainly because data sources do not transmit to NOS the data along with the policy (as happens for the sticky policy approach). The network load remains lower for the CP-ABE approach with respect to the sticky policy one, even if there is an increment in the packet dimension when NOS performs the CP-ABE encryption.

- NOSs have to store different kinds of information. Nevertheless, it is worth remarking that NOSs do not support persistent storage of IoT data for *Raw Data* and *Normalized Data* collections. In fact, incoming data is only temporarily cached on the NOSs' memory while being processed before being submitted to requesting consumers. Once data is further pushed to or pulled from the MQTT client (which handles the topics notification to subscribers), the data can be safely removed from NOSs. In both sticky policy and CP-ABE approaches, no further storage is required because the policies themselves are directly associated or embedded into the data. Hence, NOSs do not have to

store all the policies managed by the IoT system, as it happens in traditional approaches, as the one presented in (Sicari et al., 2016b). However, it is fundamental to evaluate if it takes up more memory a sticky policy attached to the data or encrypted with CP-ABE. The average memory occupancy on NOS at runtime is 10.2 megabytes with sticky policies, whereas, with CP-ABE, it slightly decreases to 8.4 megabytes. Note that such results have been obtained by equally setting the following factors for the two approaches: (i) the frequency of data fetching from sources (i.e., 1 packet/minute); (ii) the frequency of execution of the routines for removing data from non-persistent collections (in the actual environment, such a task is executed every 5 minutes); (iii) the number of sources (in the actual setup, 6 data producers are introduced). Obviously, for CP-ABE, the attributes' number highly influences the dimension of the encrypted data; however, it is true also for sticky policies.

- Concerning the sticky policy-based approach, the TA must store the whole set of the valid scopes and constraints used for sticky policies' composition (Sicari et al., 2017b). The dimension of this storage depends on the specific application domain. In the sample implementation, this was negligible. On the other hand, in the CP-ABE based approach, the TA has to maintain the AVL and CAL tables, whose sizes are also negligible in our sample implementation.

The just presented analysis about memory occupancy reveals that adopting an approach based on CP-ABE would reduce the memory occupancy and the network load, thus improving the system's scalability in the presence of a higher amount of data. However, the CP-ABE approach affects the CPU load on NOS more than the sticky policy approach, which, on the other side, increases the computational load on data sources. In fact, following the sticky policy approach, the data sources are in charge of computing the sticky policies and transmitting them along with the information to NOS; whereas, following the CP-ABE approach, the computational load is moved to NOS, which has to perform the encryption task on each incoming data. Hence, NOS shows a mean CPU load of 15.4% by adopting sticky policies, while, when running CP-ABE, the mean CPU load on NOS is 26.7%. Figure 3.8 sketches the comparison about storage occupancy, network load, and CPU load.

Considering such two perspectives, the most viable solution is based on CP-ABE because it is more efficient for end-devices since more powerful and secure devices, as NOSs, perform the heavier processing tasks. Such a point of view perfectly fits the principles of the emerging fog computing paradigm (Yi et al., 2015), which aims to: (i) reduce network's latency; (ii) prevent unnecessary network resources' consumption; (iii) enhance service availability; (iv) increase the robustness of the whole IoT system thanks to the removal of always-online points of failures into the security network infrastructure. Note that the TA is a single point of failure in the CP-ABE

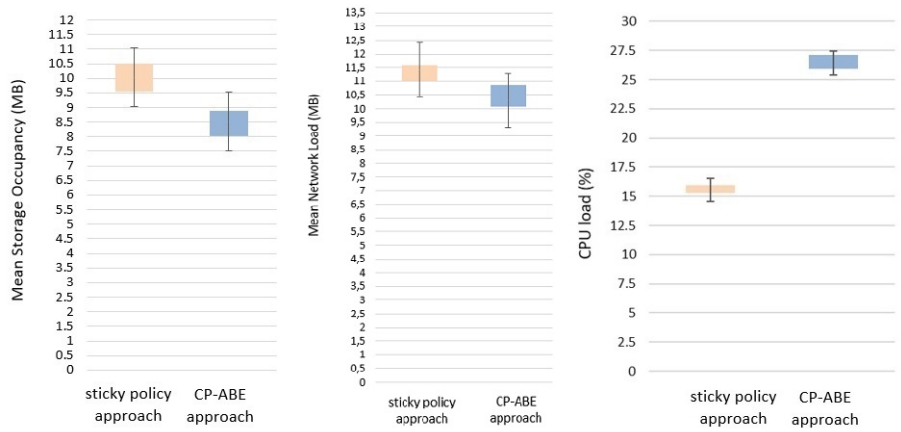


Figure 3.8. Whiskers-box diagram of mean storage occupancy and CPU load comparison: sticky policies vs CP-ABE approach.

approach as well, but it has to be online only when revoking a decryption key, so it is hardly exposed to attacks.

Data Retrieval Delay

The main difference between the two mechanisms resides in how data are disclosed and, therefore, how policies are evaluated. An important metric to be considered is the delay introduced by the enforcement framework using sticky policies with respect to CP-ABE. Note that "data retrieval delay" means the time elapsed since a consumer requests a topic subscription up to when the same consumer receives and decrypts the requested data. Moreover, as emerged in Section 3.4, the packets transmitted by the data sources to NOS in the case of the sticky policy approach are approximately 0.5 kilobytes larger than the same packets sent with CP-ABE.

In the sticky policy approach, to obtain access permission, the recipients can subscribe to specific topics, and the subscription is only accepted if the request satisfies the requirements established by the sticky policies associated with the data. NOSs do not locally evaluate access permissions, but they are delegated to the TA; a query to the TA is sent for each occurring change and, in general, for each incoming request, thus clearly spending time for transmissions and processing. Different is the approach based on CP-ABE: once the subscribers obtained the decryption keys needed for disclosing the authorized information, they no longer have to make requests to the TA, which, as just said, can be offline most of the time.

For such a reason, the data retrieval delays are different, as shown in Figure 3.9. Hence, CP-ABE allows to spend less time from two perspectives: (i) the data transmission from the source to NOS; (ii) the data disclosure. Figure 3.9 shows a comparison of the mean distribution of the delays generated by the two approaches,

measured with the considered prototypical implementation over one hour. Data-rate strictly depends on the fetching of data acquisition of the used data-set, which is every minute. The considered time window concerns a week of measurements.

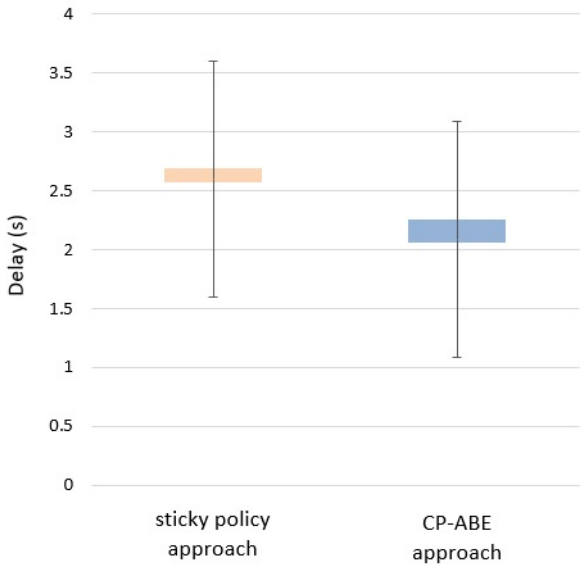


Figure 3.9. Whiskers-box diagram of mean data retrieval delay comparison: sticky policies vs CP-ABE approach.

Going in-depth into the analysis of delays, Figure 3.10 presents the encryption time required by CP-ABE for the three different kinds of data managed within the smart home, which are: the electrical data-set, the streaming video, and the remote monitoring, as explained in Section 3.4.

Finally, Figure 3.11 shows the time required for decryption in CP-ABE by varying the kind of data requested.

3.5 Answer

Attribute-Based Encryption vs. Sticky Policies: What Should We Use?

As it turns out, Attribute-Based Encryption is the answer to that question since it demonstrates several advantages compared to sticky policies in terms of memory occupancy on the IoT platform, delay, and usability of the system.

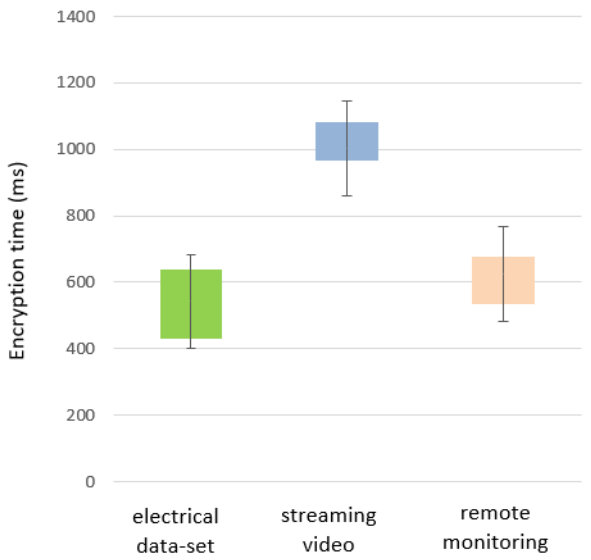


Figure 3.10. Whiskers-box diagram of mean encryption time required by CP-ABE.

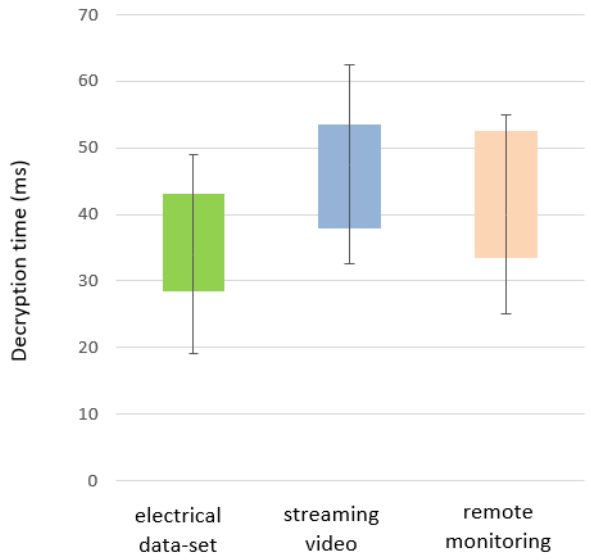


Figure 3.11. Whiskers-box diagram of mean decryption time required by CP-ABE.

Chapter 4

Can CP-ABE be Used in a Low-Bitrate WSAN?

Internet of Things (IoT) technologies (Mainetti et al., 2011; Ashton et al., 2009; Atzori et al., 2010) allow us to connect constrained or embedded devices through the Internet. This strongly impacts our everyday lives, as common objects can be empowered with communication and cooperation capabilities. In particular, the industry can take enormous advantage of IoT. For example, a smart factory can be monitored and controlled through the Internet, thus optimizing the industrial processes (Gilchrist, 2016). Another example is a smart warehouse, in which sensors can tell automated guided vehicles where to find particular goods in order to load them on a given truck. Security is a crucial requirement in all these systems, especially for integrity, confidentiality, and access control over data. Though ABE techniques offer a high level of security and intrinsic fine-grained access control, they do not fit easily in the IoT world. One of the most challenging aspects is the communication overhead generated by the ABE encryption (over 1 KB overhead per message), which may be quite burdensome for wireless networks with limited bitrate like those employed in IoT (Farrell, 2018; Montenegro et al., 2007). Indeed, modern IoT networks use low-power communication protocols like Bluetooth LE, IEEE 802.15.4, and LoRa, which provide for low bitrates (230 Kbps for BLE (Tosi et al., 2017), 163 Kbps for 802.15.4 (Latré et al., 2005), 50 Kbps for LoRa (Georgiou and Raza, 2017)).

This chapter presents fABElous, an ABE solution suitable for Industrial IoT applications that minimize the communication overhead introduced by ABE encryption. The fABElous solution ensures data integrity, confidentiality, and access control while reducing the communication overhead of 35% with respect to using ABE techniques naively.

The rest of the chapter is structured as follows. Section 4.1 introduces the related work. Section 4.2 describes fABElous in detail: its architecture, its reference use case, its system procedures, its reference threat model. Section 4.3 analyzes the

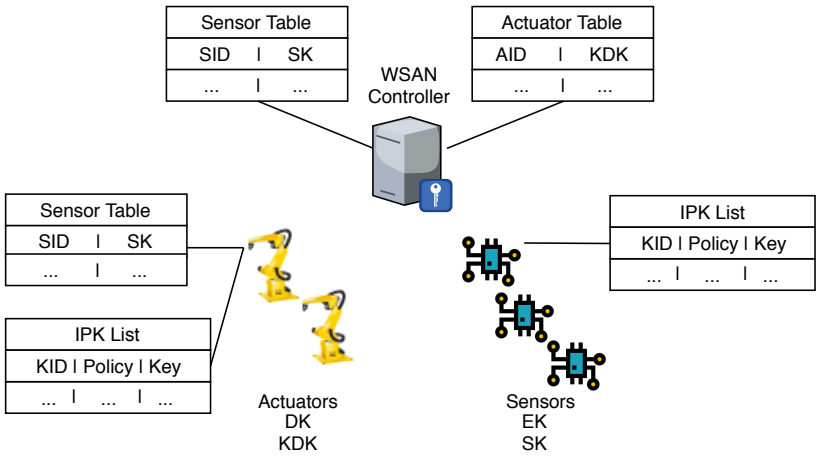


Figure 4.1. An overview of fABElous architecture.

performances of fABElous in terms of communication overhead. Section 4.4 ends the chapter, answering the question inquired.

4.1 Related Work

Attribute-Based Encryption has been applied to protect confidentiality and ensure fine-grained access control in many different application scenarios like cloud computing (Ming et al., 2011; Yu et al., 2010a; Xu and Martin, 2012; Hur, 2013), e-health (Picazo-Sanchez et al., 2014), wireless sensor networks (Yu et al., 2011), Internet of Things (Touati and Challal, 2015; Singh et al., 2015), smart cities (Rasori et al., 2018), online social networks (Jahid et al., 2011). In this chapter, we will discuss the usage of ABE in an Industrial IoT Scenario.

4.2 Architecture

We assume a low-bitrate *Wireless Sensor and Actuator Network* (WSAN), composed by a set of *sensors* and *actuators*, which exchange encrypted data with each other (Fig. 4.1). As a use-case example, consider a smart factory with many sensors and actuators which must communicate in a delay-bounded way to implement a real-time application (Chen et al., 2009). Given the strict requirements, sensors and actuators must communicate directly through the WSAN. The WSAN inside the smart factory uses IEEE 802.15.4 as a link-layer protocol, which is low-energy and low-bitrate. As a consequence, communications and encrypt/decrypt operations must be as lightweight as possible.

A sensor is a data producer that measures and encrypts some quantity, and then it sends the encrypted data to a set of actuators over the WSAN. An actuator is a

data consumer that receives encrypted data from a set of sensors over the WSAN and then uses it to control some mechanism. The encrypted data received by an actuator could be a command that the actuator executes, or it can be measured data from a sensor that the actuator uses to make a decision. Sensors and actuators are regulated by a *WSAN controller* node belonging to the WSAN. For the sake of simplicity, we keep the "sensor" role and the "actuator" role separated; however, a single device may act as both. We assume that the WSAN controller has its own pair of asymmetric keys (e.g., RSA, ECC, etc.) used for digital signature and encryption. In addition, each sensor and each actuator has a unique identifier called, respectively, *Sensor ID* (*SID*) and *Actuator ID* (*AID*), which are assigned by the controller.

Key Distribution Mechanism

In order to satisfy the strict requirements of our model regarding security and messages size, we diminish the use of CP-ABE heavier ciphertext and primitives. Indeed, in fABElous, each sensor executes the Encrypt primitive only once for securing multiple data described by the same policy. Similarly, each actuator executes the Decrypt primitive only once for extracting data generated by the same sensor and described by the same policy. The basic idea is to distribute symmetric keys using the CP-ABE scheme as a reliable tool to achieve fine-grained multicast. Each sensor encrypts a symmetric key with the CP-ABE Encrypt primitive under a specific policy and broadcasts it to all the actuators. All the actuators will receive the ciphertext, but only a few will be able to successfully execute the Decrypt primitive and retrieve the symmetric key. In this way, when the sensor wants to transmit data and apply the aforementioned policy to it, the sensor encrypts such data with the symmetric key instead. In the following, we explain the procedures that the WSAN controller, the sensors, and actuators may execute inside our system.

System Procedures

System Initialization

The system initialization procedure is executed only once to start the system. The controller runs the Setup primitive, thus obtaining the master key and the encryption key.

Sensor Join

The sensor join procedure (Fig. 4.2) is executed whenever a new sensor joins the WSAN.
First, the human operator who is physically deploying the sensor generates a pair of asymmetric keys, which the sensor will use for digital signatures. The operator

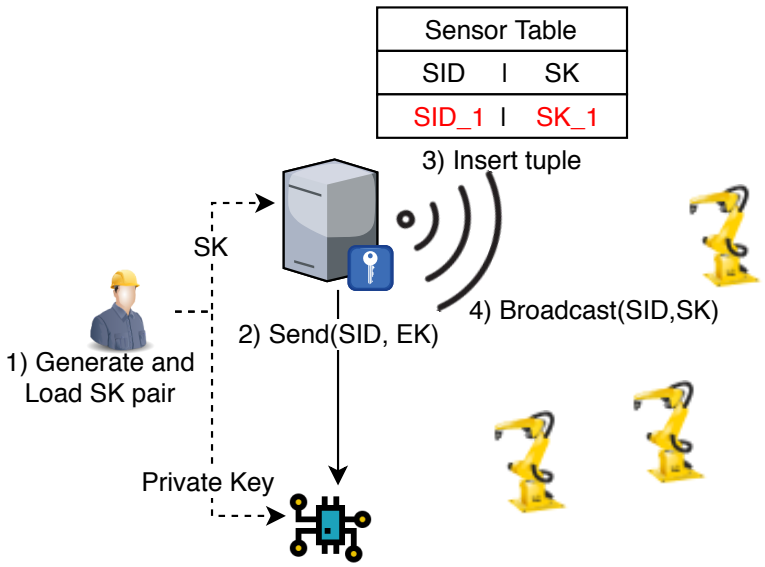


Figure 4.2. Sensor join procedure. Dashed lines represent human-device communication.

loads the private key on the sensor and the public key on the controller (step 1). We call *signature key* (*SK*) such a public key. After that, the controller assigns an identifier *SID* to the sensor, and it sends the identifier and the encryption key to the sensor with a signed message (step 2). The controller adds a tuple $\langle SID, SK \rangle$ to a locally maintained *Sensor Table* (step 3). Each tuple in the Sensor Table represents a sensor in the system. Finally, the controller signs and broadcasts the tuple to all the WSAN actuators (step 4). The WSAN actuators add such a tuple to their locally maintained copy of the WSAN Sensor Table.

Actuator Join

The actuator join procedure is executed whenever a new actuator joins the WSAN.

First, the human operator who is physically deploying the actuator generates a pair of asymmetric keys, which the actuator will use for receiving encrypted keys. The operator loads the private key on the actuator and the public key on the controller (step 1). We call *key-distribution key* (*KDK*) such a public key. After that, the controller assigns an identifier *AID* to the actuator, and it generates a decryption key with the KeyGen primitive, according to the actuator's attribute set. The controller signs the identifier and the decryption key, it encrypts such signed message with the actuator's key-distribution key, then sends the obtained ciphertext to the actuator. The controller adds a tuple $\langle AID, KDK \rangle$ to a locally maintained *Actuator Table* (step 3). Each tuple in the Actuator Table represents an actuator in the system. Finally, the WSAN controller sends the WSAN Sensor Table to the actuator with a signed message (step 4).

New Policy installation

The new policy installation procedure (Fig. 4.3) is executed by a sensor to share a symmetric key with some actuators belonging to the WSAN. When a sensor per-

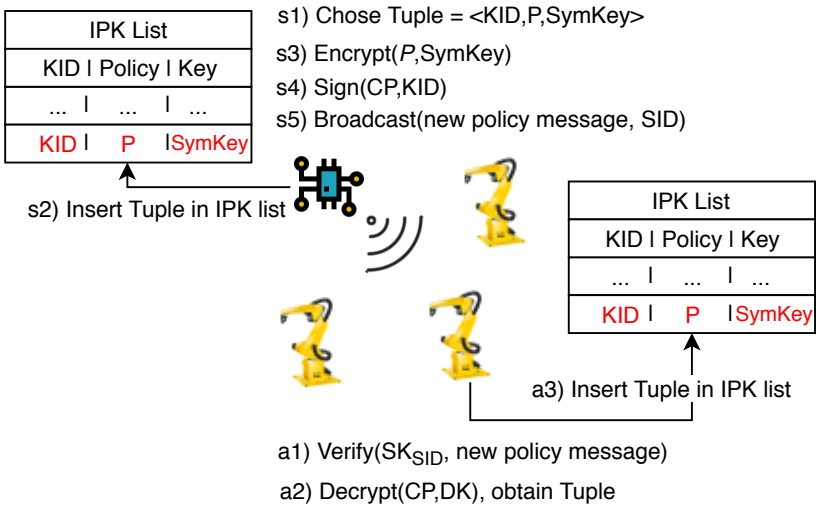


Figure 4.3. New policy installation procedure.

forms this procedure for the first time, it creates an *Identifier Policy Key* (IPK) list. This list links a policy $\mathcal{P}$ to a symmetric key SymKey through a *symmetric key identifier* (*KID*), and this allows a sensor to encrypt data with the advantages of symmetric encryption (faster and with smaller ciphertexts than asymmetric encryption), plus the capability of enforcing an access policy on said data. In other words, the IPK list is composed of one or more tuples in the form $\langle KID, \mathcal{P}, \text{SymKey} \rangle$. The IPK list is a structure owned by each sensor that belongs to the WSAN, and thus, two different sensors will have two different IPK lists. Actuators store a similar IPK list containing the tuples that the sensors shared with them through this procedure.

The following steps allow a sensor to create a tuple for its IPK list and share said tuple with some actuators over the WSAN. The sensor generates an IPK tuple by choosing a policy $\mathcal{P}$, a random symmetric key SymKey and a random *KID* (step s1). The sensor adds to its IPK list the tuple generated in step s1 (step s2). The sensor encrypts the symmetric key under $\mathcal{P}$ using the Encrypt primitive (step s3). Then the sensor signs the concatenation of its *SID*, the ciphertext, and the *KID* (step s4). Throughout the chapter, we will refer to a signed concatenation of a *SID*, an ABE ciphertext, and a *KID* as a *new policy message*. The sensor transmits the new policy message over the WSAN (step s5). Each actuator inside the WSAN verifies the sensor signature on the new policy message, using the signature key associated with the received *SID* inside the sensor table (step a1). If the signature is not correct, the message is discarded. Otherwise, the actuator checks if its attribute set γ satisfies $\mathcal{P}$. If the attribute set γ does not satisfy the policy $\mathcal{P}$, the message is discarded. Other-

wise, if the attribute set γ satisfies the policy $\mathcal{P}$, the actuator decrypts the ciphertext executing the primitive Decrypt, obtaining the symmetric key (step a2). Finally, the actuator inserts the tuple in its IPK list with the quantities retrieved in step a2 (step a3).

Data Exchange

The data exchange procedure (Fig. 4.4) is executed by a sensor to transmit data to one or more actuators in a low-latency fashion inside the WSAN.

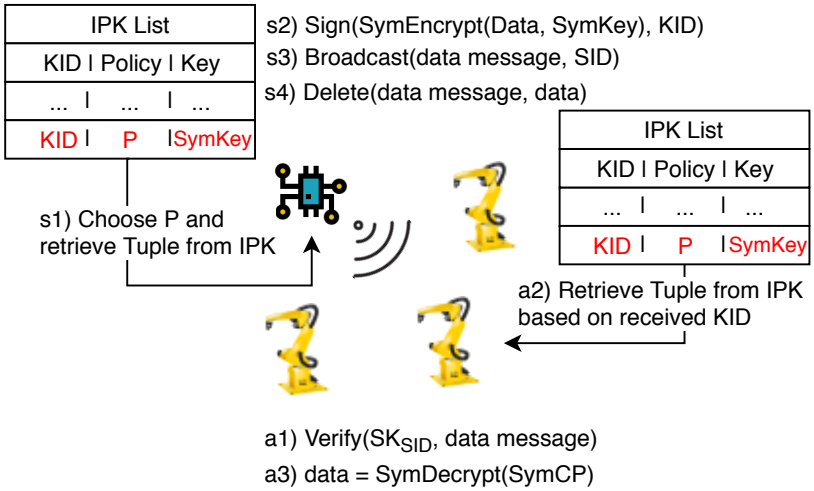


Figure 4.4. Data exchange procedure.

The sensor chooses a policy $\mathcal{P}$ and retrieves the associated symmetric key and *KID* from its IPK list (step s1). If there is no matching tuple in its IPK list, the sensor performs a new policy installation procedure. The sensor encrypts the data using the symmetric key (obtaining $SymCP$), and it signs the concatenation of its *SID*, the ciphertext, and the *KID* (step s2). Throughout the chapter, we will refer to a signed concatenation of a *SID*, asymmetric ciphertext, and a *KID* as a *data message*. Then, the sensor broadcasts the data message over the WSAN (step s3). Each actuator inside the WSAN, which is interested in the transmitted data, verifies the sensor signature on the data message by retrieving from the sensor table the signature key associated with the received *SID* (step a1). If the signature is not correct, the data message is discarded. Otherwise, the actuator retrieves from its IPK list the tuple associated with the received *KID* (step a2). The actuator uses the latest symmetric key retrieved to decrypt the data message and consumes its content (step a3). Finally, the sensor securely deletes the sensed data (step s4).

Threat Model

The fABElous scheme provides data integrity, confidentiality, and access control. In the following, we analyze possible threats and explain how fABElous addresses them.

Eavesdropper

Eavesdroppers are undoubtedly a threat to confidentiality. An eavesdropper can gain information by examining the traffic between sensors, actuators, and the controller. However, every exchange of information is protected. If an eavesdropper can obtain a data message, he cannot access the data since he does not have the symmetric key. Even if he has access to the ABE ciphertext containing that symmetric key, he cannot decrypt it either because he lacks an ABE decryption key. Even more so, if the eavesdropper intercepts the message exchange between an actuator and the WSAN controller during the actuator join procedure, he cannot retrieve the decryption key since it is safely encrypted with the key-distribution key of said actuator.

Compromised Sensor

Suppose that an attacker gains complete access over a sensor. Said attacker obtains: (i) the data generated by the sensor from the moment of the compromise on; (ii) the private signature key of the said sensor; (iii) the IPK list used by the sensor, including the symmetric keys used for data encryption. Note that this attacker cannot, in any way, obtain data generated by other sensors. Each sensor deletes past data securely, so the attacker cannot retrieve it from the sensor. However, if an attacker intercepted and stored past transmissions, he would retrieve past data by using the stolen symmetric keys. To mitigate this, sensors could periodically refresh the symmetric keys by deleting some tuples from their IPK list and executing the new policy installation procedure on the same policies again. In this way, the attacker cannot retrieve data produced before the last refresh of the symmetric key.

Note that the attacker could also disseminate malicious data authenticated with the signature key of the compromised sensor. In this way, malicious data is accepted by the actuators that receive it. This attack can be thwarted by revoking the signature key of the compromised sensor. We plan to add this functionality to a future version of fABElous.

Compromised Actuator

Suppose that an attacker gains complete access over an actuator. The said attacker obtains: (i) the private ABE decryption key of the said actuator; (ii) the private key-distribution key of the said actuator; (iii) the IPK list used by the actuator, in-

cluding the symmetric keys used for data decryption. Each actuator may delete past data securely after consumption, so the attacker cannot retrieve it from the actuator. However, if an attacker intercepted and stored past transmissions, he would retrieve past data by using the stolen symmetric keys. With this set of information, the attacker can decrypt every past and future data message that the compromised actuator has access to. Note that this does not imply that the attacker has access to all the data generated by the sensors. Indeed, his decryption capabilities are limited by the access privileges of the compromised actuator. If the compromised actuator cannot decrypt some data because its attribute set does not satisfy the policy of such data, then the attacker will not be able to decrypt it as well. This is achieved thanks to ABE technology, which enforces a fine-grained access control even in case of device compromise. The actuator compromise can be addressed by revoking its decryption key. We plan to add this functionality to a future version of fABElous.

4.3 Performance Evaluation

In this section, we show more in detail the parameters we considered for evaluating fABElous. Data is composed by the combination of 120 bytes of raw data, plus 4 byte of KID, plus 4 bytes of timestamp (to avoid replay attacks). The used digital signature algorithm is ECDSA, which has the benefit of adding a constant size signature of 40 byte (considering 80-bit security). For the symmetric key encryption, we used AES with 128-bit keys in CBC mode. The policy used to evaluate CP-ABE communication overhead is a simple, yet effective $\mathcal{P}_1 = (A \text{ AND } B \text{ AND } C)$. We used only AND operators without losing in generality since the specific Boolean operator does not influence the communication overhead. To measure the communication overhead, we used the CP-ABE toolkit of Bethencourt et al. (Bethencourt et al., 2007). Table 4.1 shows the communication overhead of fABElous compared to other schemes.

Table 4.1
Transmission Size

Scheme	Size (bytes)	Overhead (%)
No security	120	0%
Authentication only	160	25%
"Naive" CP-ABE	1.250	90%
fABElous	192(+1.122*)	100%−37.5%

*Once per policy installation

No security scheme refers to sensors transmitting raw data without any protection. *Authentication only* scheme refers to sensors transmitting signed data using

ECDSA. *"Naive" CP-ABE* scheme refers to sensors constantly transmitting data encrypted with CP-ABE. The reduction of the fABElous communication overhead over number of transmissions is calculated as:

$$\text{Overhead}(\%) = \frac{1122 + N \cdot 72}{1122 + N \cdot 192} \cdot 100,$$

where N is the number of data messages sent, 72 is the amount of overhead bytes in each data message, 1.122 byte is the size of the CP-ABE ciphertext containing an AES key, and 192 byte is the total size of each data message. As seen from the table, the overhead of fABElous encryption is as big as 100% in the worst case (no data exchange procedure executed after a new policy installation procedure).

Compared to no security, fABElous has an incredible amount of communication overhead, even in its best-case scenario. However, fABElous grants data integrity, data confidentiality, and fine-grained access control, which are three features required by our use case. Compared to authentication only, fABElous has more than twice the communication overhead, even in its best-case scenario. However, in addition to data integrity, fABElous also grants data confidentiality and fine-grained access control, which are two features required by our use case. Compared to "Naive" CP-ABE, fABElous has less communication overhead since the second data exchange execution ($N \geq 2$). Indeed, in its best-case scenario, fABElous communication overhead is 49% less than "Naive" CP-ABE communication overhead ($N \geq 99$). Furthermore, in addition to data confidentiality and fine-grained access control, fABElous also grants data integrity, which is a feature required by our use case.

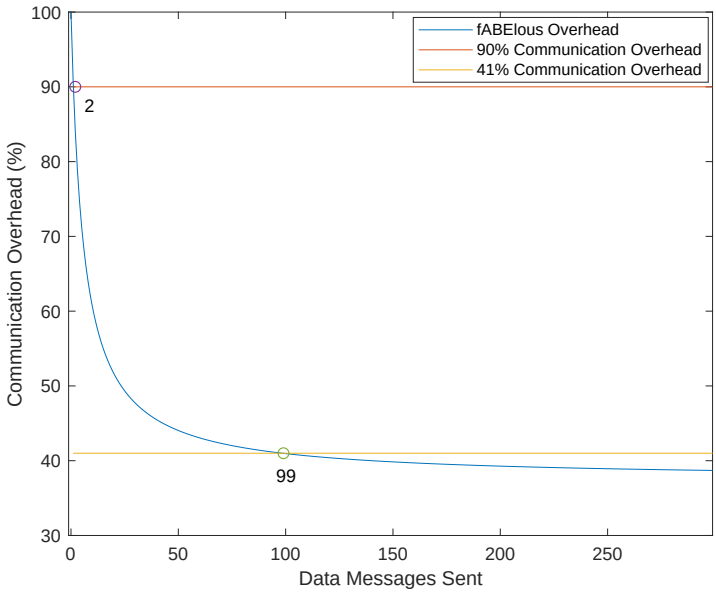


Figure 4.5. fABElous communication overhead.

Fig. 4.5 shows how fABElous communication overhead drops with each execution of the data exchange procedure. The communication overhead is lower than 90% after two executions of the data exchange procedure. Furthermore, the communication overhead drops below 41% after 99 executions of the data exchange procedure.

4.4 Answer

Can CP-ABE be Used in a Low-Bitrate WSAN?

Yes, it can! Despite the high overhead that CP-ABE introduces on the ciphertexts, by using the proposed technique, fABElous reduces such an overhead below 41%. This makes CP-ABE a viable solution to provide confidentiality in an IIoT application.

Chapter 5

How can We Improve the Original CP-ABE?

One of the most problematic aspects of cryptosystems is the recovery procedure in case of key compromise, which usually requires sending an update message to all the devices (Yu et al., 2010a). Sending many update messages could be quite burdensome for wireless networks with a limited bitrate, like those employed in IoT and IIoT as in (Farrell, 2018; Montenegro et al., 2007) or Chapter 4.

In this chapter, we propose *SEA-BREW* (Scalable and Efficient ABE with Broadcast REvocation for Wireless networks), an ABE revocable scheme suitable for low-bitrate Wireless Sensor and Actuator Networks (WSANs) in IoT applications. SEA-BREW is highly scalable in the number and size of messages necessary to manage decryption keys. In a WSAN composed of n decrypting nodes, a traditional approach based on unicast would require $O(n)$ messages. SEA-BREW, instead, can revoke or renew multiple decryption keys by sending a single broadcast message over a WSAN. Intuitively, such a message allows all the nodes to update their keys locally. For instance, if $n = 50$ and considering a symmetric pairing with 80-bit security, the traditional approach requires 50 unicast messages of 2688 bytes each, resulting in about 131KB of total traffic. SEA-BREW, instead, requires a single 252-byte broadcast message over a WSAN. Also, our scheme allows for per-data access policies, following the *Ciphertext-Policy Attribute-Based Encryption* (CP-ABE) paradigm, which is generally considered flexible and easy to use (Bethencourt et al., 2007; Liu et al., 2013b; Ambrosin et al., 2015). In SEA-BREW, things and users can exchange encrypted data via the cloud and directly if they belong to the same WSAN. This makes the scheme suitable for both remote cloud-based communications and local delay-bounded ones. The scheme also provides a mechanism of *proxy re-encryption* (Yu et al., 2010a,b; Zu et al., 2014) by which old data can be re-encrypted by the cloud to make a revoked key unusable. This is important to protect old ciphertexts from revoked keys retroactively. We formally prove that our scheme is adaptively IND-CPA

secure under the generic bilinear group model. Furthermore, it can also be made adaptively IND-CCA secure through the Fujisaki-Okamoto transformation (Fujisaki and Okamoto, 1999). We finally show by simulations that the computational overhead is constant on the cloud server, with respect to the complexity of the access control policies.

The rest of the chapter is structured as follows. In Section 5.1 we review the current state of the art. In Section 5.2 we explain our system model; furthermore, we provide a threat model, the scheme definition, and the security definition for SEA-BREW. In Section 5.3 we show the SEA-BREW system procedures. In Section 5.4 we mathematically describe the SEA-BREW primitives, and we also show the correctness of our scheme. In Section 5.5 we formally prove the security of SEA-BREW. In Section 5.6 we evaluate our scheme both analytically and through simulations. Finally, Section 5.7 ends the chapter, answering the question inquired.

5.1 Related Work

In 2007 Bethencourt et al. (Bethencourt et al., 2007) proposed the first CP-ABE scheme, upon which we built SEA-BREW. Since then, attribute-Based Encryption has been applied to provide confidentiality and assure fine-grained access control in many different application scenarios.

With the increasing interest in ABE, researchers have focused on also improving a crucial aspect of any encryption scheme: key revocation. In the following, we show many ABE schemes that feature different key revocation mechanisms to compare SEA-BREW to them. First, we recall the notions of *direct* and *indirect* revocation, introduced by (Attrapadung and Imai, 2009). Direct revocation implies that the list of the revoked keys is somehow embedded inside each ciphertext. In this way, only users in possession of a decryption key not in such a list can decrypt the ciphertext. Instead, indirect revocation implies that the list of the revoked keys is known by the key authority only, which will release some updates for the non-revoked keys and/or ciphertexts. Such updates are not distributed or are of no use to the revoked users. In this way, only users that apply the update can decrypt the ciphertexts.

The scheme of Bethencourt et al. (Bethencourt et al., 2007) lacks functionalities for key revocation and ciphertext re-encryption, which we provide in our scheme. However, a naive indirect key revocation mechanism can be realized on such a scheme, but it requires sending a new decryption key for each user in the system, resulting in $O(n)$ point-to-point messages where n is the number of users. In contrast, SEA-BREW can revoke or renew a decryption key by sending a single $O(1)$-sized broadcast message over a wireless network, and it also provides a re-encryption mechanism delegated to the untrusted cloud server.

Attrapadung et al. (Attrapadung and Imai, 2009) proposed a hybrid ABE scheme that supports both direct and indirect revocation modes. According to that work's authors, this flexibility is a great advantage in a system because the devices can leverage the quality of both approaches depending on the situation. The indirect revocation mechanism is based on time slots. When a key revocation is performed in the middle of a time slot, it is effective only from the beginning of the next time slot; therefore, revocation is not immediate. Instead, their direct mechanism also implies the immediate key revocation. Notably, with their indirect revocation mechanism, it is possible to revoke or renew a decryption key by sending a single broadcast message over a WSAN. However, such message is usually $O(log(n))$-sized where n is the amount of the users in the system, including the ones revoked in the past. Moreover, their scheme does not provide any mechanism of re-encryption; therefore, if a revoked user somehow can get an old ciphertext, he/she can still decrypt it. Instead, SEA-BREW can revoke or renew a decryption key by sending a single $O(1)$-sized broadcast message, and it also provides a re-encryption mechanism.

Liu et al. (Liu et al., 2018) proposed a Time-Based Direct Revocable CP-ABE scheme with a Short Revocation List. Since the revocation is direct, the revocation list is embedded in the ciphertext, therefore achieving immediate key revocation. Furthermore, the authors condensed the entire revocation list into a few hundred bytes, as long as the number of total revocations does not overcome a threshold value. However, since the revocation list is destined to grow uncontrollably over time, they also propose a secret key time validation technique. This technique allows a data producer to remove a compromised decryption key from the revocation list once such a decryption key has expired. Unlike SEA-BREW, this scheme does not provide re-encryption of old ciphertexts. Furthermore, the direct revocation mechanism implies that each data producer must know the revocation list. In fact, in SEA-BREW, data producers encrypt their data without knowing any information about revoked consumers.

Touati et al. (Touati and Challal, 2015) proposed an ABE system for IoT which implements an indirect key revocation mechanism based on time slots. In their work, time is divided into slots, and policies can be modified only at the beginning of a slot. This approach is efficient only if key revocations and policy changes are known a priori. An example is an access privilege that expires after one year. Unfortunately, in many systems, there is no possibility of knowing beforehand when and which access privilege should be revoked. For example, in case a decryption key gets compromised, the system must revoke it as soon as possible. Our scheme gives this possibility.

Cui et al. (Cui et al., 2016), and Qin et al. (Qin et al., 2017) proposed two indirect revocable CP-ABE schemes which do not require communication with data producers during a revocation process. However, their schemes require all data pro-

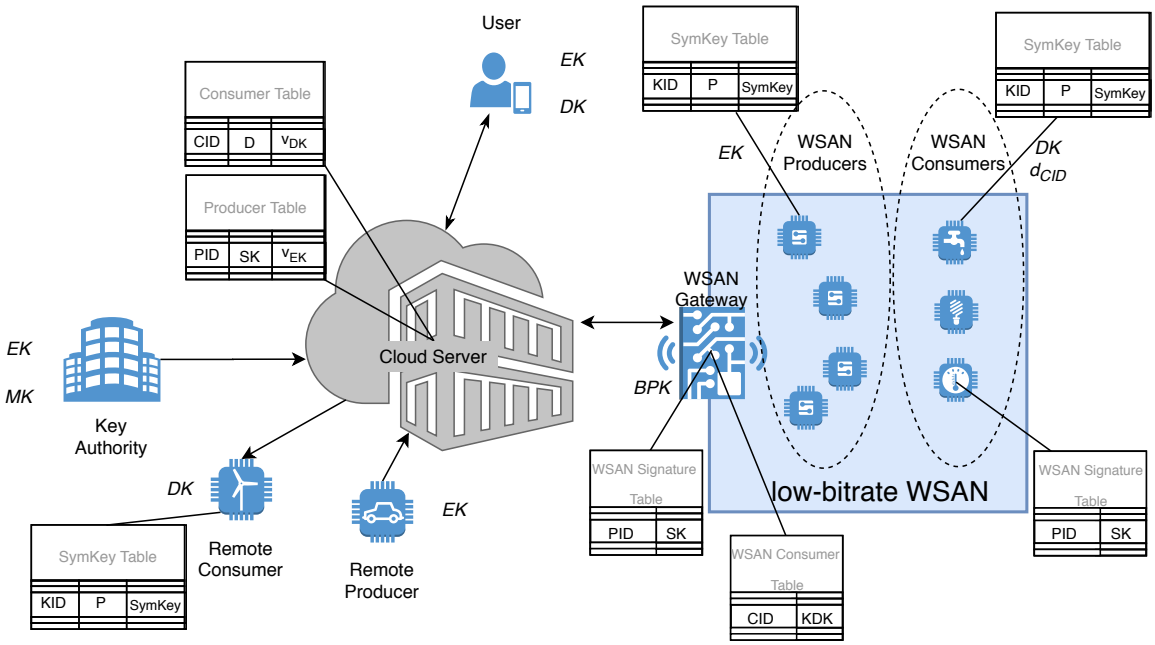


Figure 5.1. SEA-BREW system model.

ducers to be time-synchronized in a secure manner. This could be difficult to achieve and hard to implement in a WSAN where data producers are often very resource-constrained sensors. Their schemes do not provide a re-encryption mechanism nor an efficient key update distribution, unlike SEA-BREW. Furthermore, SEA-BREW has not the constraint of tight time synchronization.

Yu et al. (Yu et al., 2010a) proposed an ABE scheme to share data on a cloud server. The scheme revokes a compromised decryption key by distributing an update to non revoked users. The update is done attribute-wise: this means that only users with some attributes in common with the revoked key need to update their keys. Such an update mechanism provides immediate key revocation, as well as ciphertext re-encryption. Notably, their revocation mechanism is not efficient for WSAN, as it requires $O(n)$ different messages where n is the number of decrypting parties that need to be updated. On the other hand, SEA-BREW can revoke or renew a decryption key by sending a single $O(1)$-sized broadcast message over the wireless network.

Finally, we can see that the scheme proposed by Yu et al. (Yu et al., 2010a) is the one with the most features similar to SEA-BREW. Indeed, we will compare the performance of SEA-BREW and the scheme in (Yu et al., 2010a) in section 5.6

5.2 System Model and Scheme Definition

Figure 5.1 shows our reference system model. We assume a low-bitrate WSAN, com-

posed of a set of sensors and actuators, which upload and download encrypted data to/from a *cloud server*. Sensors and actuators access the cloud server through an Internet-connected *WSAN gateway* node belonging to the WSAN. Sensors and actuators inside the WSAN can also communicate directly without passing through the cloud server. We assume that some sensors and actuators are outside the WSAN, and they can also upload and download encrypted data to/from the cloud server, but they cannot communicate directly. In addition, human users outside the WSAN can upload and download encrypted data to/from the cloud server. The encrypted data received by an actuator could be a command that the actuator must execute, as well as a measurement from a sensor that the actuator can use to make some decisions. The cloud server is an always-online platform managed by an untrusted third-party company that offers storage and computational power to privates or other companies. Finally, a fully trusted *key authority* is in charge of generating, updating, and distributing cryptographic keys.

In the following, we will call *producers* all those system entities that produce and encrypt data. This includes sensors internal or external to the WSAN, which sense data and users that produce data or commands for actuators. Similarly, we will call *consumers* all those system entities that decrypt and consume data. This includes actuators internal or external to the WSAN, which request data and receive commands, and users that request data. For the sake of simplicity, we keep the "producer" and the "consumer" roles separated. However, SEA-BREW allows a single device or a single user to act as both. Producers that are inside the WSAN will be called *WSAN producers*, while those outside the WSAN will be called *remote producers*. Similarly, consumers that are inside the WSAN will be called *WSAN consumers*, while those outside the WSAN will be called *remote consumers*.

As a use-case example, consider a smart factory with many sensors and actuators which must communicate in a delay-bounded way to implement a real-time application (Chen et al., 2009). Given the strict requirements, sensors and actuators must communicate directly through the WSAN without losing time in remote communications with the cloud. The WSAN inside the smart factory uses IEEE 802.15.4 as a link-layer protocol, which is low-energy and low-bitrate. As a consequence, communications and key management operations must be as lightweight as possible. In addition, employees, external sensors, and external actuators involved in remote applications will upload or download data to/from the cloud server.

Each producer encrypts data by means of an *encryption key* (EK). Each consumer decrypts data by means of a *decryption key* (DK). The encryption key is public and unique for all the producers, whereas the decryption key is private and specific to a single consumer. A single piece of encrypted data is called *ciphertext* (CP). Each consumer is described by a set of attributes (γ), cryptographically embedded into its decryption key. The access rights on each ciphertext are described by an *access policy*

($\mathcal{P}$). We assume that the key authority, the cloud server, and the WSAN gateway have their own pair of asymmetric keys used for digital signature and encryption (e.g., RSA or ECIES keys). In addition, each producer and each consumer has a unique identifier called, respectively, *producer identifier* (*PID*) and *consumer identifier* (*CID*), which are assigned by the key authority. If a device acts as both producer and consumer, it will have a producer identifier and a consumer identifier.

When a decryption key needs to be revoked (e.g., because it is compromised or because a consumer has to leave the system), the key authority must ensure that it will not be able to decrypt data anymore. This is achieved by *Proxy Re-Encryption* (PRE). Re-Encryption consists in modifying an existing ciphertext such that a specific decryption key can no longer decrypt it. This is important to protect old ciphertexts from revoked keys retroactively. In SEA-BREW, as in other schemes (Yu et al., 2010a), the Re-Encryption is "proxied" because it is delegated to the cloud server, which thus acts as a full-resource proxy for the producers. Therefore, data producers do not have to do anything to protect data generated before a revocation. However, the cloud server re-encrypts blindly, that is, without accessing the plaintext of the messages. This makes our scheme resilient to possible data leakage on the cloud server. Our PRE mechanism is also "lazy", which means that the ciphertext is modified not immediately after the key revocation, but only when some consumer downloads it. This allows us to spread the computational costs sustained by the cloud server for the PRE operations over time. We implement the lazy PRE scheme by assigning a version to the encryption key, decryption key, and ciphertext. When a key is revoked, the key authority modifies the encryption key, increments its version, and uploads some update quantities to the cloud server. The set of these update quantities is called *update key*. The cloud server uses the update key to blindly re-encrypt the ABE ciphertexts and increments their version before sending them to the requesting consumers. The cloud server also uses the update key to update the encryption key used by producers and the decryption keys used by consumers. Inside the low-bitrate WSAN, instead, the update of the WSAN consumers' decryption keys is achieved with a *constant-ciphertext broadcast encryption scheme*, like the one shown in Boneh et al.'s work (Boneh et al., 2005). The broadcast encryption scheme allows the WSAN gateway to broadcast the update key encrypted in such a way as to exclude one or more WSAN consumers from decrypting it. To do this, the WSAN gateway uses a *broadcast public key* (*BPK*), and each WSAN consumer uses its own *broadcast private key* (d_{CID}). Table 5.1 lists the symbols used in the chapter.

Threat Model

In this section, we model a set of adversaries, and we analyze the security of our system against them. In particular, we consider the following adversaries: (i) an *external adversary*, which does not own any cryptographic key except the public ones;

EK	Encryption key
MK	Master key
DK	Decryption key
KDK	Key distribution key
PID	Producer identifier
SK	Signature verification key
CID	Consumer identifier
KID	Symmetric key identifier
$SymKey$	Symmetric key
$\mathcal{P}$	Access policy
γ	Attribute set
BPK	Broadcast public key
d_{CID}	Broadcast private key
CP	Ciphertext
U	Update key
M	Message

Table 5.1
Table of Symbols

(ii) a *device compromiser*, which can compromise sensors and actuators to steal secrets from them; (iii) a set of *colluding consumers*, which own some decryption keys; and (iv) a *honest-but-curious cloud server* as defined in (Yu et al., 2010a; Rasori et al., 2018; Di Vimercati et al., 2007), which does not tamper with data and correctly executes the procedures, but it is interested in accessing data. We assume that the honest-but-curious cloud server might also collude with a set of consumers who own some decryption keys. Note that the honest-but-curious cloud server models also an adversary capable of *breaching* the cloud server, meaning that he can steal all the data stored in it. To do this, he can leverage some common weaknesses, for example, buffer overflows or code injections, or hardware vulnerabilities like Meltdown or Spectre (Reidy, 2018). We assume that who breaches the cloud server only steals data and does not alter its behavior of correctly executing all the protocols, basically because he tries to remain as stealthy as possible during the attack. Note that this reflects real-life attacks against cloud servers[1]. In the following, we analyze in detail each adversary model.

The external adversary aims at reading or forging data. To do so, he can adopt several strategies. He can impersonate the key authority to communicate a false encryption key to the producer so that the data encrypted by the said producer will be accessible by the adversary. This attack is avoided because the key authority signs the encryption keys. Alternatively, the external adversary can act as a man in the

[1] `https://www.bbc.com/news/technology-41147513`

middle between the key authority and a new consumer during the decryption key distribution. The attacker wants to steal the consumer's decryption key, with which he can later decrypt data. This attack is avoided because the key authority encrypts the decryption key with asymmetric encryption. Using the encryption key, which is public, the external adversary may also try to encrypt false data and upload it to the cloud server. This attack is avoided because he cannot forge a valid signature for the encrypted data; thus, he cannot make the false data be accepted as valid by the legitimate consumers. To sum up, the external adversary cannot access legitimate data nor inject malicious data.

The device compromiser can compromise a producer or a consumer. If he compromises a producer, he gains complete control of such a device and full access to its sensed data and its private key used for signatures. He cannot retrieve any data sensed before the compromise because the producer securely deletes data after uploading it to the cloud server. Nonetheless, he can inject malicious data into the system by signing it and uploading it to the cloud server or transmitting it directly to WSAN consumers if the compromised producer belongs to the WSAN. When the key authority finds out the compromise, it revokes the compromised producer. After that, the compromised producer cannot inject malicious data anymore because the private key used for signatures is no longer considered valid by the consumers. On the other hand, if the adversary compromises a consumer, he gains full access to its decryption key. The attacker can decrypt *some* data downloaded from the cloud server and, if he compromised a consumer belonging to the WSAN, transmitted directly by WSAN producers. Notably, the adversary can decrypt only data that the compromised consumer was authorized to decrypt. When the key authority finds out the compromise, it revokes the compromised consumer. After that, the compromised consumer cannot decrypt data anymore. The reason for this is that our re-encryption mechanism updates the ciphertexts as if they were encrypted with a different encryption key.

A set of colluding consumers can somehow combine their decryption keys to decrypt some data that singularly they cannot decrypt. However, even if the union of the attribute sets of said decryption keys satisfies the access policy of a ciphertext, the colluding consumers cannot decrypt such a ciphertext. In Section 5.5 we will capture this adversary model with Game 1, and we will provide formal proof that SEA-BREW is resistant against it.

The honest-but-curious cloud server does not have access to data because it is encrypted, but it can access all the update keys and part of all the consumers' decryption keys. The update keys alone are useless to decrypt data because the cloud server lacks a (complete) decryption key. However, if the cloud server colludes with a set of consumers, then it can access all the data that the consumers are authorized to decrypt. Interestingly, if the honest-but-curious cloud server is modeling an ad-

versary capable of breaching the cloud server, recovering the breach is easy. It is sufficient that the key authority generates a new update key without revoking any consumers. This has the effect of making all the stolen update keys useless. On the other hand, in the case of an *actual* honest-but-curious cloud server, generating a new update key does not solve the problem because the cloud server knows the just generated update key, and thus it can update the revoked decryption keys. In any case, the honest-but-curious cloud server and the colluding consumers cannot combine somehow the update keys and decryption keys to decrypt some data that singularly the colluding consumers cannot decrypt. In Section 5.5 we will capture this adversary model with Game 2, and we will provide formal proof that SEA-BREW is resistant against it.

Scheme Definition

Our system makes use of a set of *cryptographic primitives* (from now on, simply primitives), which are the following ones.

(MK, EK) = **Setup**(κ): This primitive initializes the cryptographic scheme. It takes a security parameter κ as input, and outputs a *master key MK* and an associated *encryption key EK*.

CP = **Encrypt**$(M, \mathcal{P}, EK)$: This primitive encrypts a plaintext M under the policy $\mathcal{P}$. It takes as input the message M, the encryption key EK, and the policy $\mathcal{P}$. It outputs the ciphertext CP.

DK = **KeyGen**(γ, MK): This primitive generates a decryption key. It takes as input a set of attributes γ which describes the consumer, and the master key MK. It outputs a decryption key DK, which is composed of two fields for each attribute in γ, plus a field called D, useful to update such a key.

M = **Decrypt**(CP, DK): This primitive decrypts a ciphertext CP. It takes the ciphertext CP and the consumer's decryption key DK as input, and outputs the message M if decryption is successful, $\perp$ otherwise. The decryption is successful if and only if γ satisfies $\mathcal{P}$, which is embedded in CP.

The following primitives use symbols with a superscript number to indicate the version of the associated quantity. For example, $MK^{(i)}$ indicates the i-th version of the master key, $DK^{(i)}$ indicates the i-th version of a given decryption key, etc.

$(MK^{(i+1)}, U^{(i+1)})$ = **UpdateMK**$(MK^{(i)})$: This primitive updates the master key from a version i to the version $i+1$ after a key revocation. It takes as input the old master

key $MK^{(i)}$, and it outputs an updated master key $MK^{(i+1)}$, and the $(i+1)$-th version of the update key $U^{(i+1)}$. Such an update key is composed of the quantities $U_{EK}^{(i+1)}$, $U_{DK}^{(i+1)}$, $U_{CP}^{(i+1)}$, which will be used after a key revocation respectively to update the encryption key, to update the decryption keys, and to re-encrypt the ciphertexts.

$EK^{(n)} = \textbf{UpdateEK}(EK^{(i)}, U_{EK}^{(n)})$: This primitive updates an encryption key from a version i to the latest version n, with $n > i$, after a key revocation. The primitive takes as input the old encryption key $EK^{(i)}$ and $U_{EK}^{(n)}$, and it outputs the updated encryption key $EK^{(n)}$.

$D^{(n)} = \textbf{UpdateDK}(D^{(i)}, U_{DK}^{(i)}, U_{DK}^{(i+1)}, \ldots, U_{DK}^{(n)})$: This primitive updates a decryption key from a version i to the latest version n, with $n > i$, after a key revocation. What is updated is not the whole decryption key but only a particular field D inside the decryption key. This allows the cloud server to execute the primitive without knowing the whole decryption key, but only D, which alone is useless for decrypting anything. The primitive takes as input the old field $D^{(i)}$ and $U_{DK}^{(i)}, U_{DK}^{(i+1)}, \ldots, U_{DK}^{(n)}$, and it outputs the updated field $D^{(n)}$.

$CP^{(n)} = \textbf{UpdateCP}(CP^{(i)}, U_{CP}^{(i)}, U_{CP}^{(i+1)}, \ldots, U_{CP}^{(n)})$: This primitive updates a ciphertext from a version i to the latest version n, with $n > i$, after a key revocation. The cloud server executes this primitive to perform proxy re-encryption on ciphertexts. The primitive takes as input the old ciphertext $CP^{(i)}$, and $U_{CP}^{(i)}, U_{CP}^{(i+1)}, \ldots, U_{CP}^{(n)}$. It outputs the updated ciphertext $CP^{(n)}$.

The concrete construction of these primitives will be described in detail in Section 5.4.

Security Definition

We state that SEA-BREW is secure against an adaptive chosen-plaintext attack (IND-CPA) if no probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ has a non-negligible advantage against the challenger in the following game, denoted as Game 1. Note that IND-CPA security is not enough in the presence of an active adversary; however, a more robust adaptive IND-CCA security assurance can be obtained in the random oracle model through the simple Fujisaki-Okamoto transformation (Fujisaki and Okamoto, 1999), which only requires a few additional hash computations in the Encrypt and the Decrypt primitives.

Setup The challenger runs the Setup primitive, generates $EK^{(0)}$, and sends it to the adversary.

Phase 1 The adversary may issue queries for:

- *encryption key update*: the challenger runs the primitive UpdateMK. The challenger sends the updated encryption key to the adversary.
- *generate decryption key*: the challenger runs the primitive KeyGen using as input an attribute set provided by the adversary. Then, the challenger sends the generated decryption key to the adversary.
- *decryption key update*: the challenger runs the primitive UpdateDK using as input a decryption key provided by the adversary. Then, the challenger sends the updated decryption key to the adversary.
- *ciphertext update*: the challenger runs the primitive UpdateCP using as input a ciphertext provided by the adversary. Then, the challenger sends the ciphertext updated to the last version to the adversary.

Challenge The adversary submits two equal-length messages m_0 and m_1 and a challenge policy $\mathcal{P}^*$, which is not satisfied by any attribute set queried as *generate decryption key* during Phase 1. The challenger flips a fair coin and assigns the outcome to b: $b \leftarrow \{0,1\}$. Then, the challenger runs the Encrypt primitive encrypting m_b under the challenge policy $\mathcal{P}^*$ using $EK^{(n)}$ and sends the ciphertext CP^* to the adversary. The symbol n is the last version of the master key, i.e., the number of times the adversary queried for an encryption key update.

Phase 2 Phase 1 is repeated. However, the adversary cannot issue queries for *generate decryption key* whose attribute set γ satisfies the challenge policy $\mathcal{P}^*$.

Guess The adversary outputs a guess b' of b. The advantage of an adversary $\mathcal{A}$ in Game 1 is defined as $\Pr[b' = b] - \frac{1}{2}$.

We prove SEA-BREW to be secure in Section 5.5.

5.3 SEA-BREW Procedures

In the following, we describe the procedures that our system performs.

System Initialization

The system initialization procedure is executed only once to start the system, and it consists of the following steps.

Step 1. The key authority runs the Setup primitive, thus obtaining the first version of the master key ($MK^{(0)}$) and the first version of the encryption key ($EK^{(0)}$).

We indicate with v_{MK} (*master key version*) the current version of the master key. The key authority initializes the master key version to $v_{MK} = 0$, and it sends the encryption key and the master key version to the cloud server with a signed message.
Step 2. The cloud server, in turn, sends the encryption key and the master key version to the WSAN gateway with a signed message.
Step 3. The WSAN gateway generates the broadcast public key (see Figure 5.1) for the broadcast encryption scheme.

Producer Join

The consumer join procedure is executed whenever a new producer joins the system. We assume that the producer has already pre-installed its own pair of asymmetric keys that it will use for digital signatures. Alternatively, the producer can create such a pair at the first boot. We call *signature verification key* (SK, see Figure 5.1) the public key of such a pair. The procedure consists of the following steps.
Step 1. The producer sends the signature verification key to the key authority in some authenticated fashion. For example, in case the producer is a sensor, the human operator who is physically deploying the sensor can leverage a pre-shared password with the key authority.
Step 2. The key authority assigns a new producer identifier to the producer, and it sends such an identifier and the encryption key to the producer with a signed message. The encryption key embeds an *encryption key version* (v_{EK}), which represents the current version of the encryption key locally maintained by the producer. Initially, the encryption key version is equal to the master key version ($v_{EK} = v_{MK}$).
Step 3. The key authority also sends the producer's identifier, signature verification key, and encryption key version to the cloud server with a signed message. The cloud server adds a tuple $\langle PID, SK, v_{EK} \rangle$ to a locally maintained *Producer Table* (PT, see Figure 5.1). Each tuple in the PT represents a producer in the system.

If the producer is remote, then the procedure ends here. Otherwise, if the producer is inside the WSAN, then the following additional steps are performed.
Step 4. The key authority sends the producer identifier and the signature verification key to the WSAN gateway with a signed message. The WSAN gateway adds a tuple $\langle PID, SK \rangle$ to a locally maintained *WSAN Signature Table* (see Figure 5.1). Each tuple in the WSAN Signature Table represents a producer in the WSAN. Through this table, both the gateway and the consumers can authenticate data and messages generated by the producers in the WSAN.
Step 5. The WSAN gateway finally broadcasts the signed message received from the key authority to all the WSAN consumers. The WSAN consumers add the same tuple $\langle PID, SK \rangle$ to a locally maintained copy of the WSAN Signature Table.

Consumer Join

The consumer join procedure is executed whenever a new consumer, described by a given attribute set, joins the system. We assume that the consumer has already pre-installed its own pair of asymmetric keys that it will use for asymmetric encryption. Alternatively, the consumer can create such a pair at the first boot. We call *key distribution key* (*KDK*, see Figure 5.1) the public key of such a pair. The procedure consists of the following steps.

Step 1. The consumer sends the key distribution key to the key authority in some authenticated fashion.

Step 2. The key authority assigns a new consumer identifier to the consumer, and it generates a decryption key with the *KeyGen* primitive, according to the consumer's attribute set. Then, the key authority sends the consumer identifier and the decryption key to the consumer with a signed message, encrypted with the consumer's key distribution key.

Step 3. The key authority sends the consumer identifier and the field D of the decryption key to the cloud server with a signed message. The cloud server initializes a *decryption key version* (v_{DK}), which represents the current version of the consumer's decryption key, to the value of the master key version. The cloud server adds a tuple $\langle CID, D, v_{DK} \rangle$ to a locally maintained *Consumer Table* (CT, see Figure 5.1). Each tuple in the CT represents a consumer in the system.

If the consumer is remote, then the procedure ends here. Otherwise, if the consumer is a WSAN consumer, then the following additional steps are performed.

Step 4. The key authority sends the consumer identifier and the key distribution key to the WSAN gateway with a signed message.

Step 5. The WSAN gateway sends the WSAN Signature Table to the consumer with a signed message, along with the broadcast public key and the consumer's broadcast private key, which is appropriately encrypted with the consumer's key distribution key. Finally, the WSAN gateway adds a tuple $\langle CID, KDK \rangle$ to a locally maintained *WSAN Consumer Table*.

Data Upload by Remote Producers

The data upload procedure is executed whenever a producer wants to upload data to the cloud server. Remote producers and WSAN producers perform two different procedures to upload a piece of information to the cloud server. We explain them separately. The data upload procedure by remote producers consists of the following steps.

Step 1. Let $\mathcal{P}$ be the access policy that has to be enforced over the data. The remote producer encrypts the data under such a policy using the Encrypt primitive. The resulting ciphertext has the same version number of the producer's locally main-

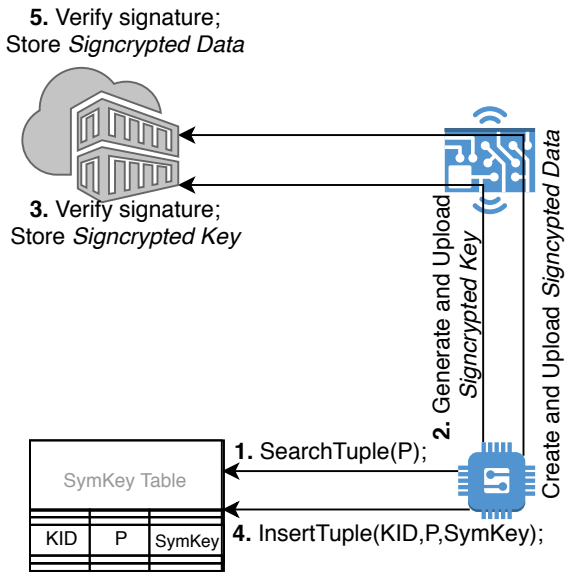


Figure 5.2. Data upload by WSAN producers procedure.

tained encryption key ($v_{CP} = v_{EK}$).
Step 2. The producer securely deletes the original data. Then it signs and uploads the ciphertext to the cloud server, along with its producer identifier.
Step 3. The cloud server verifies the signature, and then it stores the ciphertext. Finally, if the ciphertext version is older than the master key version, the cloud server executes the remote producer update procedure (see Section 5.3).

Data Upload by WSAN Producers

SEA-BREW aims at saving bandwidth in the WSAN also during data upload. However, encrypting data directly with the *Encrypt* primitive introduces too much overhead in terms of data size, as it happens in the typical ABE scheme. Therefore, we want to obtain the access control mechanism provided by the *Encrypt* primitive, and at the same time, we want a small ciphertext typical of symmetric-key encryption. We achieve this by encrypting a symmetric key using the *Encrypt* primitive and then using such a symmetric key to encrypt all the data that must be accessible with the same access policy, as described in Chapter 4. To do this, each WSAN producer maintains a *SymKey Table* (see Figure 5.1), which associates policies $\mathcal{P}$ to symmetric keys *SymKey*. More specifically, the SymKey Table is composed of tuples in the form $\langle KID, \mathcal{P}, SymKey \rangle$, where KID is the *symmetric key identifier* of $SymKey$. The symmetric key identifier uniquely identifies a symmetric key in the whole system. The data upload procedure by WSAN producers consists of the following steps (Figure 5.2).
Step 1. Let $\mathcal{P}$ be the access policy that has to be enforced over the data. The producer

searches for a tuple inside its SymKey Table associated with the policy. If such a tuple already exists, then the producer jumps directly to Step 4; otherwise, it creates it by continuing to Step 2.
Step 2. The producer randomly generates a symmetric key and a symmetric key identifier. The symmetric key identifier must be represented on a sufficient number of bits to make the probability that two producers choose the same identifier for two different symmetric keys negligible. The producer then encrypts the symmetric key under the policy using the Encrypt primitive, and it signs the resulting ciphertext together with the key identifier. The result is the *signcrypted key*. The producer uploads the signcrypted key and its producer identifier to the cloud server.
Step 3. The cloud server verifies the signature, and then it stores the signcrypted key in the same way it stores ordinary encrypted data produced by remote producers.
Step 4. The producer inserts (or retrieves, if steps 2 and 3 have not been executed) the tuple $\langle KID, \mathcal{P}, \text{SymKey} \rangle$ into (from) its SymKey Table, and it encrypts the data using the symmetric key associated to the policy. Then, the producer signs the resulting ciphertext together with the symmetric key identifier. The result is the *signcrypted data*. The producer uploads the signcrypted data and its producer identifier to the cloud server, and it securely deletes the original data.
Step 5. The cloud server verifies the signature, and then it stores the signcrypted data.

Data Download

The data download procedure is executed whenever a consumer wants to download data from the cloud server. Consumers perform two different procedures to download information from the cloud server, depending on whether such information has been produced by a remote producer or by a WSAN producer. We explain them separately. The download procedure of data produced by remote producers consists of the following steps.
Step 1. The consumer sends a data request along with its consumer identifier to the cloud server.
Step 2. The cloud server checks in the CT whether the decryption key version of the consumer is older than the master key version and, if so, it updates the decryption key by executing the remote consumer update procedure (see after). The cloud server identifies the requested ciphertext and checks whether its version is older than the master key version. If so, the cloud server updates the ciphertext by executing the UpdateCP primitive (see Section 5.4).
Step 3. The cloud server signs and sends the requested data to the consumer.
Step 4. The consumer verifies the server signature over the received message. Then, it executes the Decrypt primitive using its decryption key.

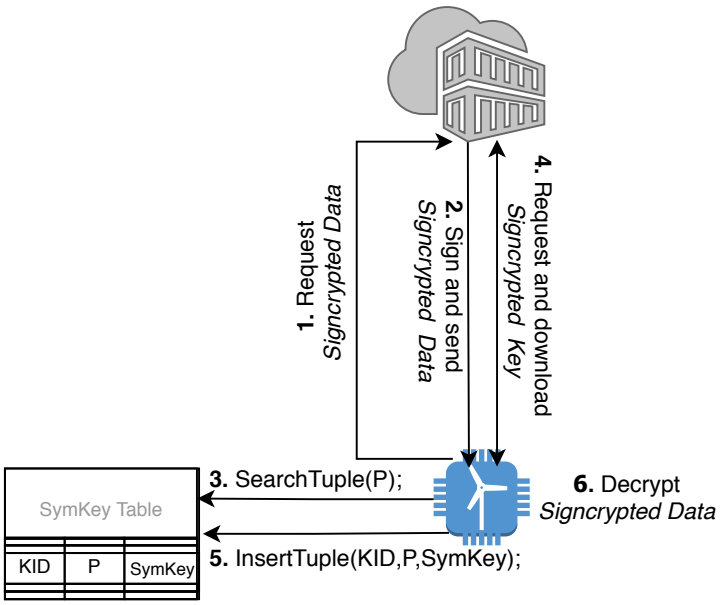


Figure 5.3. Download signcrypted data procedure.

Now consider the case in which a consumer requests data produced by a WSAN producer. Each consumer maintains a SymKey Table (see Figure 5.1), which associates policies $\mathcal{P}$ to symmetric keys *SymKey*. The download procedure of data produced by WSAN producers consists of the following steps (Figure 5.3).

Step 1. The consumer sends a data request along with its consumer identifier to the cloud server.

Step 2. The cloud server signs and sends the requested signcrypted data to the consumer.

Step 3. The consumer searches for a tuple with the same key identifier as the one contained in the received signcrypted data inside its SymKey Table. If such a tuple already exists, then the consumer jumps directly to Step 6; otherwise, the consumer creates it by continuing to Step 4.

Step 4. The consumer performs a data download procedure, requesting and obtaining the signcrypted key associated to the received symmetric key identifier.

Step 5. The consumer decrypts the signcrypted key thus obtaining the symmetric key. It adds the tuple $\langle KID, \mathcal{P}, \text{SymKey} \rangle$ to its SymKey Table.

Step 6. The consumer decrypts the signcrypted data with the symmetric key.

Direct Data Exchange

The direct data exchange procedure is executed whenever a producer wants to transmit data to consumers in a low-latency fashion inside the WSAN. The producer broadcasts the data directly to the authorized consumers in an encrypted form to obtain low latency instead of uploading such data to the cloud server. Further-

more, to save WSAN bandwidth, we want the data exchanged to be encrypted with symmetric-key encryption, under the form of signcrypted data as it happens for data uploaded by WSAN producers. We assume that the producer already has a tuple associated with the policy it wants to apply to ease the reading. Otherwise, the producer should previously perform a data upload procedure to the cloud in which it uploads the signcrypted key it will use.

The procedure consists of the following steps.

Step 1. Let $\mathcal{P}$ be the access policy that has to be enforced over the data. The producer retrieves the symmetric key associated with such policy inside its SymKey Table. The producer encrypts the data with the symmetric key and signs it together with the symmetric key identifier. It thus obtains the signcrypted data.

Step 2. The producer broadcasts the signcrypted data in the WSAN and securely deletes the original data.

Step 3. The interested consumers perform Steps 3-6 of the download procedure of data produced by WSAN producers.

Producer Leave

The producer leave procedure is executed whenever one or more producers leave the system. This happens if producers are dismissed from the system or the private keys they use for signatures are compromised. In all these cases, the private keys of the leaving producers must be revoked, so the cloud server no longer accepts that data signed with such keys. The procedure consists of the following steps.

Step 1. The key authority communicates to the cloud server the identifiers of the leaving producers with a signed message.

Step 2. The cloud server removes the tuples associated with such identifiers from the PT.

If at least one leaving producer was a WSAN producer, the following additional steps are performed.

Step 3. The key authority communicates the identifiers of the leaving WSAN producers to the WSAN gateway with a signed message.

Step 4. The WSAN gateway removes the tuples associated with such identifiers from the WSAN Signature Table, and it broadcasts the signed message received by the key authority to all the WSAN consumers.

Step 5. The WSAN consumers remove the tuples associated with such identifiers from their locally maintained copy of the WSAN Signature Table.

Consumer Leave

The consumer leave procedure is executed whenever one or more consumers leave the system, as depicted in figure 5.4. This happens in the case that consumers are

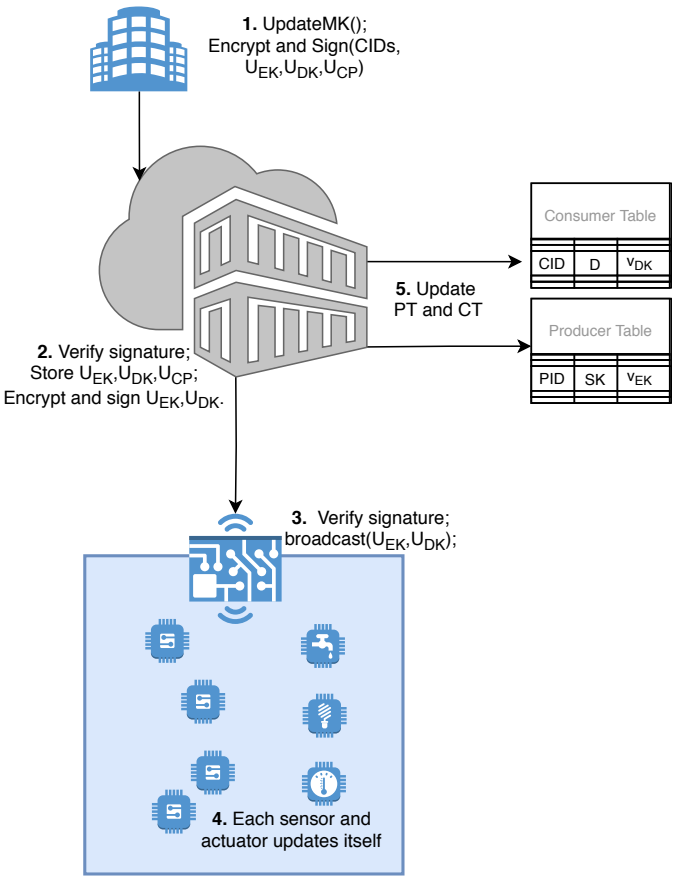


Figure 5.4. Consumer leave procedure.

dismissed from the system or their keys are compromised. In all these cases, the decryption keys of the leaving consumers must be revoked so that they cannot decrypt data anymore. The procedure consists of the following steps.

Step 1. The key authority increases the master key version, and it executes the UpdateMK primitive on the old master key, thus obtaining the new master key and the quantities $U_{EK}^{(v_{MK})}$, $U_{DK}^{(v_{MK})}$, and $U_{CP}^{(v_{MK})}$. Then, the key authority sends the identifiers of the leaving consumers and the quantities $U_{EK}^{(v_{MK})}$, $U_{DK}^{(v_{MK})}$, and $U_{CP}^{(v_{MK})}$ to the cloud server with a signed message, encrypted with the cloud server's public key.

Step 2. The cloud server verifies the signature, decrypts the message, retrieves the consumer identifier from the message, and removes the tuples associated with those identifiers from the CT. Note that the cloud server could now re-encrypt all the ciphertexts by using the quantity $U_{CP}^{(v_{MK})}$ just received. However, the re-encryption of each ciphertext is deferred to the time at which a consumer requests it (Lazy PRE). Then, the cloud server signs and encrypts $U_{EK}^{(v_{MK})}$ and $U_{DK}^{(v_{MK})}$ with asymmetric encryption, and it sends them to the gateway.

Step 3. The gateway broadcasts the quantity $U_{EK}^{(v_{MK})}$ and $U_{DK}^{(v_{MK})}$ over the local low-bitrate WSAN, so that all the producers and consumers that belong to it can imme-

diately update their encryption key and decryption key, respectively. To do this, the gateway sends a single broadcast message, composed as follows. The gateway encrypts the $U_{DK}^{(v_{MK})}$ quantity with the broadcast public key, in such a way that all the WSAN consumers except the leaving ones can decrypt it. This allows the gateway to share said quantity only with the WSAN consumers, excluding the compromised ones if there are any. The gateway then signs the concatenation of the quantity $U_{EK}^{(v_{MK})}$, and the quantity $U_{DK}^{(v_{MK})}$ (encrypted), and broadcasts said message over the WSAN.

Step 4. Each producer updates its encryption key upon receiving the broadcast message; each consumer then decrypts the received message using its broadcast private key d_{CID}, and executes the UpdateDK primitive using its old decryption key and the just received $U_{DK}^{(v_{MK})}$. The WSAN producers and the consumers delete their SymKey Tables.

Step 5. The cloud server updates inside the PT the versions of the encryption keys of all the WSAN producers, and inside the CT the versions of the decryption keys of all the WSAN consumers.

Note that SEA-BREW updates all the devices inside the low-bitrate WSAN with a single $O(1)$-sized broadcast message (Step 3). This makes SEA-BREW highly scalable in the number and size of messages necessary to manage decryption keys. Note also that, regarding remote consumers and remote producers, the computational load of the consumer leave procedure is entirely delegated to the cloud server, leaving the producers and consumers free of heavy computation. This enables SEA-BREW to run on a broader class of sensors and actuators.

Remote Producer Update

The producer update procedure is executed during the data upload procedure by remote producers (see Section 5.3), and it consists of the following steps. **Step 1.** The cloud server signs and sends the last quantity U_{EK} received from the key authority to the remote producer that must be updated.

Step 2. The producer verifies the signature and retrieves U_{EK}. Then, it executes the UpdateEK primitive using its encryption key and the received quantity U_{EK} as parameters.

Step 3. The cloud server updates the producer's encryption key version to v_{MK} inside PT.

Remote Consumer Update

The consumer update procedure is executed as specified in the data download procedure (see Section 5.3), and it consists in the following steps.

Step 1. The cloud server executes the UpdateDK primitive using the consumer's

decryption key and the last $(v_{MK} - v_{DK})$ quantities U_{DK}s received from the key authority. The cloud server signs the output of that primitive, $D^{(v_{MK})}$, and sends it to the consumer.
Step 2. The consumer verifies the signature and replaces the old field D of its decryption key with the received quantity.
Step 3. The cloud server updates the consumer's decryption key version to v_{MK} inside CT.

5.4 Concrete Construction

We now explain in detail how the CP-ABE primitives previously introduced at the beginning of Section 5.2 are realized.

$(MK^{(0)}, EK^{(0)}) = \text{Setup}(\kappa)$
The Setup primitive is executed by the key authority. This primitive computes:

$$EK^{(0)} = \{\mathbb{G}_0, g, h = g^{\beta}, l = e(g,g)^{\alpha}, v_{EK} = 0\}; \tag{5.1}$$

$$MK^{(0)} = \{\beta, g^{\alpha}, v_{MK} = 0\}, \tag{5.2}$$

where $\mathbb{G}_0$ is a multiplicative cyclic group of prime order p with size κ, g is the generator of $\mathbb{G}_0$, $e : \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_1$ is an efficiently-computable bilinear map with bi-linearity and non-degeneracy properties, and $\alpha, \beta \in \mathbb{Z}_p$ are chosen at random.

$CP = \text{Encrypt}(M, \mathcal{P}, EK^{(v_{EK})})$
The Encrypt primitive is executed by a producer. From now on, $\mathcal{P}$ is represented as a *policy tree*, which is a labeled tree where the non-leaf nodes implement *threshold-gate operators* whereas the leaf nodes are the attributes of the policy. A threshold-gate operator is a Boolean operator of the type k-of-n, which evaluates to true iff at least k (*threshold value*) of the n inputs are true. Note that a 1-of-n threshold gate implements an OR operator, whereas an n-of-n threshold gate implements an AND operator. For each node x belonging to the policy tree the primitive selects a polynomial q_x of degree equal to its threshold value minus one ($d_x = k_x - 1$). The leaf nodes have threshold value $k_x = 1$, so their polynomials have degree equal to $d_x = 0$. The polynomials are chosen in the following way, starting from the root node R. The primitive assigns an index arbitrarily to each node inside the policy tree. The index range varies from 1 to *num*, where *num* is the total number of the nodes. The function index(x) returns the index assigned to the node x. Starting with the root node R the primitive chooses a random $s \in Z_p$ and sets $q_R(0) = s$. Then, it randomly chooses d_R other points of the polynomial q_R to define it completely. Iteratively, the primitive

sets $q_x(0) = q_{\text{parent}(x)}(\text{index}(x))$ for any other node x and randomly chooses d_x other points to completely define q_x, where parent(x) refers to the parent of the node x. At the end, the ciphertext is computed as follows:

$$\begin{aligned} CP = \{\mathcal{P}, \tilde{C} = Me(g,g)^{\alpha s}, C = h^s, v_{CP} = v_{EK} \\ \forall y \in Y: \quad C_y = g^{q_y(0)}, C'_y = H(\text{att}(y))^{q_y(0)}\}, \end{aligned} \tag{5.3}$$

where Y is the set of leaf nodes of the policy tree. The function att(x) is defined only if x is a leaf node, and it denotes the attribute associated with the leaf. H is a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_0$ that is modeled as a random oracle. The encryption key version v_{EK} is assigned to the ciphertext version v_{CP}.

$DK = \text{KeyGen}(MK^{(v_{MK})}, \gamma)$
The KeyGen primitive is executed by the key authority. This primitive randomly selects $r \in \mathbb{Z}_p$, and $r_j \in \mathbb{Z}_p$ for each attribute in γ. It computes the decryption key DK as:

$$\begin{aligned} DK = \{D = g^{\frac{(\alpha+r)}{\beta}}, v_{DK} = v_{MK} \\ \forall j \in \gamma: \quad D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j}\}. \end{aligned} \tag{5.4}$$

$M = \text{Decrypt}(CP, DK)$
The Decrypt primitive is executed by a consumer. This primitive executes the sub-function *DecryptNode* on the root node. $DecryptNode(DK, CP, x)$ takes as input the consumer's decryption key, the ciphertext and the node x. If the node x is a leaf node, let $i = \text{att}(x)$ and define the function as follows. If $i \in \gamma$, then:

$$DecryptNode(DK, CP, x) = \frac{e(D_i, C_x)}{e(D'_i, C'_x)}. \tag{5.5}$$

Otherwise, if $i \notin \gamma$, then $DecryptNode(DK, CP, x) = \perp$. When x is not a leaf node, the primitive proceeds as follows. First of all, let $\Delta_{i,S}$ be the Lagrange coefficient for $i \in \mathbb{Z}_p$ and let S be an arbitrary set of element in $\mathbb{Z}_p : \Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$. Now, for all nodes z that are children of x, it calls recursively itself and stores the result as F_z. Let S_x be an arbitrary k_x-sized set of children z such that $F_z \neq \perp \ \forall z \in S_x$. Then, the function computes:

$$F_x = \prod_{z \in S_z} F_z^{\Delta_{i,S'_x}(0)} = e(g,g)^{r \cdot q_x(0)}. \tag{5.6}$$

where $i = \text{index}(z)$, and $S_x = \text{index}(z) : z \in S_x$. The Decrypt$(CP, DK)$ primitive first calls $DecryptNode(DK, CP, R)$ where R is the root of the policy tree extracted by $\mathcal{P}$ embedded in CP. Basically, the sub-function navigates the policy tree embedded inside the ciphertext in a top-down manner, and if γ satisfies the policy tree, it returns

$A = e(g,g)^{rs}$. Finally, the primitive computes:

$$M = \tilde{C}/(e(C,D)/A). \tag{5.7}$$

$(MK^{(v_{MK}+1)}, U^{(v_{MK}+1)}) = \text{UpdateMK}(MK^{(v_{MK})})$
The UpdateMK primitive is executed by the key authority. This primitive increments v_{MK} by one, chooses at random a new $\beta^{(v_{MK})} \in \mathbb{Z}_p$, and computes:

$$\begin{aligned} U_{CP}^{(v_{MK})} &= \frac{\beta^{(v_{MK})}}{\beta^{(v_{MK}-1)}}; \\ U_{EK}^{(v_{MK})} &= g^{\beta^{(v_{MK})}}; \\ U_{DK}^{(v_{MK})} &= \frac{\beta^{(v_{MK}-1)}}{\beta^{(v_{MK})}}; \\ U^{(v_{MK})} &= \{U_{CP}^{(v_{MK})}, U_{EK}^{(v_{MK})}, U_{DK}^{(v_{MK})}\}. \end{aligned} \tag{5.8}$$

Then it updates the master key as:

$$MK^{(v_{MK})} = \{\beta^{(v_{MK})}, g^{\alpha}, v_{MK}\}. \tag{5.9}$$

In order to avoid ambiguities, we specify that the first ever update key is $U^{(1)}$ and not $U^{(0)}$ as the value v_{MK} is incremented *before* the creation of U. The careful reader surely have noticed that U_{CP} and U_{DK} are reciprocal. In practice, we can use only one of these quantities and compute the other by inverting it. In this chapter, we chose to keep those quantity separated for the sake of clarity.

$EK^{(v_{MK})} = \text{UpdateEK}(EK^{(v_{EK})}, U_{EK}^{(v_{MK})})$
The UpdateEK primitive is executed by the producers. Regardless the input encryption key's version, this primitive takes as input only the last update key generated, namely $U_{EK}^{(v_{MK})}$. The primitive substitutes the field h inside the encryption key with the last update quantity, and updates the encryption key version to the latest master key version, thus obtaining:

$$EK^{(v_{MK})} = \{\mathbb{G}_0, g, h = U_{EK}^{(v_{MK})}, l = e(g,g)^{\alpha}, v_{EK} = v_{MK}\}. \tag{5.10}$$

$D^{(v_{MK})} = \text{UpdateDK}(U_{DK}^{(v_{DK}+1)}, \ldots, U_{DK}^{(v_{MK})}, D^{(v_{DK})})$
The UpdateDK primitive is executed by the cloud server and by the WSAN consumers. The decryption key on input has been lastly updated with $U_{DK}^{(v_{DK})}$, and the overall latest update is $U_{DK}^{(v_{MK})}$, with, $v_{MK} > v_{DK}$. This primitive computes:

$$\begin{aligned} U'_{DK} &= U_{DK}^{(v_{DK}+1)} \cdot \ldots \cdot U_{DK}^{(v_{MK})}; \\ D^{(v_{MK})} &= (D^{(v_{DK})})^{U'_{DK}}. \end{aligned} \tag{5.11}$$

$CP^{(v_{MK})} = \text{UpdateCP}(CP^{(v_{CP})}, U_{CP}^{(v_{CP}+1)}, \ldots, U_{CP}{}^{(v_{MK})})$
The UpdateCP primitive is executed by the cloud server. The ciphertext on input has been lastly re-encrypted with $U_{CP}^{(v_{CP})}$, and the overall latest update is $U_{CP}^{(v_{MK})}$, with, $v_{MK} > v_{CP}$. This primitive computes the re-encryption quantity U'_{CP} as the multiplication of all the version updates successive to the one in which the ciphertext has been lastly updated.

$$U'_{CP} = U_{CP}^{(v_{CP}+1)} \cdot \ldots \cdot U_{CP}^{(v_{MK})}. \tag{5.12}$$

Then, re-encryption is achieved with the following computation:

$$C^{(v_{MK})} = (C^{(v_{CP})})^{U'_{CP}}. \tag{5.13}$$

Finally, the primitive outputs the re-encrypted ciphertext CP' as:

$$\begin{aligned} CP^{(v_{MK})} = \{&\mathcal{P}, \tilde{C}, C^{(v_{MK})}, v_{CP} = v_{MK}, \\ &\forall y \in Y : \quad C_y = g^{q_y(0)}, C'_y = H(\text{att}(y))^{q_y(0)}\}. \end{aligned} \tag{5.14}$$

Correctness.

In the following, we show the correctness of SEA-BREW.

Decrypt equation (5.6):

$$\begin{aligned} F_x &= \prod_{z \in S_z} F_z^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_z} (e(g,g)^{r \cdot q_z(0)})^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_z} (e(g,g)^{r \cdot q_{\text{parent}(z)}(\text{index}(z))})^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_z} e(g,g)^{r \cdot q_x(i) \cdot \Delta_{i,S'_x}(0)} \\ &= e(g,g)^{r \cdot q_x(0)}. \end{aligned} \tag{5.15}$$

Decrypt equation (5.7):

$$\begin{aligned} \tilde{C}/(e(C,D)/A) &= \tilde{C}/(e(h^s, g^{\frac{\alpha+r}{\beta}})/e(g,g)^{rs}) \\ &= Me(g,g)^{\alpha s}/\left(e(g,g)^{\beta s \cdot \frac{\alpha+r}{\beta}}/e(g,g)^{rs}\right) \\ &= \frac{Me(g,g)^{\alpha s}}{e(g,g)^{\alpha s}} = M. \end{aligned} \tag{5.16}$$

UpdateDK equation (5.11):

$$D^{(v_{MK})} = (D^{(v_{DK})})^{U'_{DK}} = g^{\frac{r+\alpha}{\beta^{(v_{DK})}} \cdot \frac{\beta^{(v_{DK})}}{\beta^{(v_{MK})}}} = g^{\frac{r+\alpha}{\beta^{(v_{MK})}}}. \tag{5.17}$$

UpdateCP equation (5.13):

$$C^{(v_{MK})} = (C^{(v_{CP})})^{U'_{CP}} = g^{s\beta^{(v_{CP})}.\frac{\beta^{(v_{MK})}}{\beta^{(v_{CP})}}} = g^{s\beta^{(v_{MK})}}. \tag{5.18}$$

5.5 Security Proofs

In this section, we provide formal proofs of two security properties of our scheme related to two adversary models described in Section 5.2. Namely, we prove our scheme to be adaptively IND-CPA secure against a set of colluding consumers (Theorem 1), and against an honest-but-curious cloud server colluding with a set of consumers (Theorem 2).

Theorem 1. *SEA-BREW is secure against an IND-CPA by a set of colluding consumers (Game 1) under the generic bilinear group model.*

Proof. Our objective is to show that SEA-BREW is not less secure than the CP-ABE scheme by Bethencourt et al. (Bethencourt et al., 2007), which is proved to be IND-CPA secure under the generic bilinear group model. To do this, we prove that if there is a PPT adversary $\mathcal{A}$ that can win Game 1 with non-negligible advantage ϵ against SEA-BREW, then we can build a PPT simulator $\mathcal{B}$ that can win the CP-ABE game described in (Bethencourt et al., 2007) (henceforth, Game 0) against the scheme of Bethencourt et al. with the same advantage. We will denote the challenger of Game 0 as C. We describe the simulator $\mathcal{B}$ in the following.

Setup In this phase C gives to $\mathcal{B}$ the public parameters EK of Game 0, that will be exactly $EK^{(0)}$ in Game 1. In turn, $\mathcal{B}$ sends to $\mathcal{A}$ the encryption key $EK^{(0)}$ of Game 1.

Phase 1 Let us denote with the symbol n the latest version of the master key at any moment. In addition, let us denote with the symbol k a specific version of a key or a ciphertext lower than n, so that $k < n$ at any moment. The queries that an adversary can issue to the simulator are the following.

- *encryption key update*: $\mathcal{B}$ chooses $U_{DK}^{(n+1)}$ at random from $\mathbb{Z}_p$. Then, $\mathcal{B}$ computes

$$h^{(n+1)} = (g^{\beta^{(n)}})^{\frac{1}{U_{DK}^{(n+1)}}}, \tag{5.19}$$

  and sends $EK^{(n+1)}$ to $\mathcal{A}$. Finally, $\mathcal{B}$ increments n. Please note that $\mathcal{B}$ does not know $\beta^{(i)}, \forall i \in [0, n]$, but it does not need to. $\mathcal{B}$ needs to know only the relationship between any two consecutive versions, which are exactly:

$$U_{DK}^{(i)} = \frac{\beta^{(i-1)}}{\beta^{(i)}}, \forall i \in [1, n] \tag{5.20}$$

- *generate decryption key*: when $\mathcal{A}$ issues a query for $DK_j^{(n)}$ (i.e., a decryption key with a given attribute set γ_j, and latest version n) to $\mathcal{B}$, $\mathcal{B}$ in turn issues a query for DK_j to $\mathcal{C}$, and receives $DK_j^{(0)}$. Then $\mathcal{B}$ upgrades such a key to the latest version n executing the primitive UpdateDK, using as input said key and $U_{DK}^{(i)}, \forall i \in [1, n]$. Finally $\mathcal{B}$ sends to $\mathcal{A}$ the desired decryption key $DK_j^{(n)}$.
- *decryption key update*: when $\mathcal{A}$ issues a query for upgrading an existing decryption key $DK_w^{(k)}$, $\mathcal{B}$ upgrades such a key to the last version n executing the primitive UpdateDK, using as input said key and $U_{DK}^{(i)}, \forall i \in [k, n]$. Finally $\mathcal{B}$ sends to $\mathcal{A}$ the updated decryption key $DK_w^{(n)}$.
- *ciphertext update*: when $\mathcal{A}$ issues a query for upgrading an existing ciphertext $CP^{(k)}$, $\mathcal{B}$ upgrades such a ciphertext to the latest version n executing the primitive UpdateCP, using as input said ciphertext and $(U_{DK}^{(i)})^{-1}, \forall i \in [k, n]$. Finally $\mathcal{B}$ sends to $\mathcal{A}$ the updated ciphertext $CP^{(n)}$.

Challenge $\mathcal{A}$ submits two equal length messages m_0 and m_1 and a challenge policy $\mathcal{P}^*$ to $\mathcal{B}$, which in turn forwards them to $\mathcal{C}$. $\mathcal{C}$ responds with CP^* to $\mathcal{B}$, that will be exactly $CP^{*(0)}$ of Game 1. Then, $\mathcal{B}$ upgrades such a ciphertext to the latest version n executing the primitive UpdateCP, using as input said ciphertext and $(U_{DK}^{(i)})^{-1}, \forall i \in [1, n]$. Finally $\mathcal{B}$ sends to $\mathcal{A}$ the updated challenge ciphertext $CP^{*(n)}$.

Phase 2 Phase 1 is repeated.

Guess $\mathcal{A}$ outputs b' to $\mathcal{B}$, which forwards it to $\mathcal{C}$. Since a correct guess in Game 1 is also a correct guess in Game 0 and vice versa, then the advantage of the adversary $\mathcal{A}$ in Game 1 is equal to that of the adversary $\mathcal{B}$ in Game 0. Namely, such an advantage is $\epsilon = O(q^2/p)$, where q is a bound on the total number of group elements received by the $\mathcal{A}$'s queries performed in Phase 1 and Phase 2, which is negligible with the security parameter κ. Please note that, in the encryption key update query, the adversary $\mathcal{A}$ cannot distinguish an $U_{DK}^{(i)}$ provided by $\mathcal{B}$ from one provided by the real scheme. Indeed, even if the generation of such a quantity is different, its probability distribution is uniform in $\mathbb{Z}_p$ as in the real scheme. This allows the simulator $\mathcal{B}$ to answer all the other queries in Phase 1 and Phase 2 in a way that is indistinguishable from the real scheme.This concludes our proof.

■

We now consider an honest-but-curious cloud server colluding with a set of consumers. We state that a scheme is secure against an IND-CPA by an honest-but-curious cloud server colluding with a set of consumers if no PPT adversary $\mathcal{A}$ has a non-negligible advantage against the challenger in the following game, denoted

as Game 2. Game 2 is the same as Game 1 except that: (i) for every *encryption key update* query in Phase 1 and Phase 2, the adversary is also given the update quantities $U_{DK}^{(i)}, \forall i \in [1, n]$; and (ii) during Phase 1 and Phase 2 the adversary can issue the following new type of query.

- *generate decryption key's D field*: the challenger runs the primitive KeyGen using as input an attribute set provided by the adversary. Then, the challenger sends the field D of the generated decryption key to the adversary.

Note that differently from the *generate decryption key* query, when issuing a *generate decryption key's D field* query the adversary is allowed to submit an attribute set that *satisfies* the challenge policy $\mathcal{P}^*$.

Theorem 2. *SEA-BREW is secure against an IND-CPA by an honest-but-curious cloud server colluding with a set of consumers (Game 2) under the generic bilinear group model.*

Proof. We prove that if there is a PPT adversary $\mathcal{A}$ that can win Game 2 with non-negligible advantage ϵ against SEA-BREW, then we can build a PPT simulator $\mathcal{B}$ that can win Game 1 against SEA-BREW with the same advantage. We can modify the simulator $\mathcal{B}$ used in the proof of Theorem 1 to prove this theorem. In the Phase 1 and Phase 2, $\mathcal{B}$ additionally gives to $\mathcal{A}$ the update quantities $U_{DK}^{(i)}, \forall i \in [1, n]$, which $\mathcal{B}$ creates at each *encryption key update* query. During Phase 1 and Phase 2, when $\mathcal{A}$ issues a *generate decryption key's D field* query, $\mathcal{B}$ treats it in the same way of a *generate decryption key* query with an empty attribute set $\gamma = \{\emptyset\}$. Note indeed that a decryption key component D_{γ_j} is indistinguishable from a *complete* decryption key with *no attributes*. Hence, we can say that the advantage of $\mathcal{A}$ in Game 2 is the same as that of $\mathcal{B}$ in Game 0. Namely, such an advantage is $\epsilon = O(q^2/p)$, which is negligible with the security parameter κ. ■

5.6 Performance Evaluation

In this section, we analytically estimate the performances of SEA-BREW compared to: (i) the Bethencourt et al.'s scheme (Bethencourt et al., 2007) provided with a simple key revocation mechanism, denoted as "BSW-KU" (Bethencourt-Sahai-Waters with Key Update); and (ii) Yu et al. scheme (Yu et al., 2010a), denoted as "YWRL" (Yu-Wang-Ren-Lou). We considered these two schemes for different reasons. BSW-KU represents the simplest revocation method built upon the "classic" CP-ABE scheme of Bethencourt et al. Thus the performance of this revocation method constitutes the baseline reference for a generic revocable CP-ABE scheme. On the other hand, YWRL represents a KP-ABE counterpart of SEA-BREW since it natively supports an immediate key revocation and a Lazy PRE mechanism.

The revocation mechanism of BSW-KU works as follows. The producer leave procedure works the same way as SEA-BREW: the WSAN gateway simply broadcasts a signed message containing the producer identifier to all the WSAN consumers, which removes the tuples associated with such an identifier from their locally maintained copy of the WSAN Signature Table. The consumer leave procedure requires the WSAN gateway to send a signed broadcast message containing the new encryption key to all the WSAN producers and, in addition, an encrypted and signed message containing a new decryption key to each WSAN consumer. This procedure results in $O(n)$ point-to-point messages where n is the number of WSAN consumers. In contrast, SEA-BREW can perform both a consumer leave procedure by sending a single $O(1)$-sized signed broadcast message over the WSAN.

WSAN Traffic Overhead

In this section, we analytically estimate the traffic overhead that the key revocation mechanism of SEA-BREW generates in the WSAN, compared to the simple key revocation mechanism of BSW-KU. In both SEA-BREW and BSW-KU schemes, for implementing $\mathbb{G}_0$, $\mathbb{G}_1$, and the bilinear pairing, we consider a supersingular elliptic curve with embedding degree $k = 2$ defined over a finite field of 512 bits. For the signatures of the unicast and broadcast messages, we consider a 160-bit ECDSA scheme. Moreover, for the selective broadcast encryption used in the SEA-BREW scheme, we consider the Boneh et al. scheme (Boneh et al., 2005) with the same supersingular elliptic curve as above. This gives both schemes an overall security level of 80 bits. We assume that, in both SEA-BREW and BSW-KU schemes, all elliptic-curve points are represented in compressed format (Cohen et al., 2005) when they are sent over wireless links. This allows us to halve their size from 1024 bits to 512 bits. We further assume a low-bitrate WSAN composed of one gateway, 50 consumers, and 50 producers. An attribute set of 20 attributes describes each consumer. We assume that the consumer identifiers and the producer identifiers are both 64-bit long.

Table 5.2 shows the traffic overhead of consumer leave and producer leave procedures of SEA-BREW and BSW-KU schemes. In SEA-BREW, the broadcast message sent by the WSAN gateway during the consumer leave procedure is composed by the ECDSA signature (40 bytes), U_{EK} (64 bytes), and U_{DK} encrypted with the broadcast public key (148 bytes). Here we assumed that U_{DK} is encrypted through one-time pad with a key encrypted by the Boneh et al.'s broadcast encryption scheme (Boneh et al., 2005), so it is composed of 20 bytes (the one-time-padded U_{DK}) plus the broadcast encryption overhead (128 bytes). As can be seen from the table, inside a low-bitrate WSAN, SEA-BREW produces the same traffic overhead as the BSW-KU scheme when performing the producer leave procedure. However, the overhead is merely 0.2% of that produced by the BSW-KU scheme when performing a consumer leave procedure. Indeed, SEA-BREW can revoke or renew multiple de-

	Size of broadcast message (bytes)	Number/size of unicast messages (bytes)	Total (bytes)
SEA-BREW			
consumer leave	252	-	252
producer leave	48	-	48
BSW-KU			
consumer leave	256	50×2,688	134,656
producer leave	48	-	48

Table 5.2
Traffic overhead of key revocation procedures in the WSAN.

cryption keys by sending a single 252-byte (considering 80-bit security) broadcast message over the WSAN, opposed to the one 256-byte broadcast message plus 50 unicast messages of 2688-byte each (total: ~131KB of traffic) necessary to update a network with 50 consumers (each of them described by 20 attributes) in a traditional CP-ABE scheme. With bigger WSANs (more than 50 consumers) or bigger attribute sets (more than 20 attributes), the advantage of SEA-BREW compared to the BSW-KU scheme grows even more. Moreover, SEA-BREW also provides a re-encryption mechanism delegated to the untrusted cloud server, absent in the BSW-KU scheme.

Computational Overhead

In Table 5.3 we compare the computational cost of the primitives of SEA-BREW with those of BSW-KU and YWRL, in terms of number and type of needed operations. In the table, the symbol $\mathcal{A}_{rev}$ indicates the set of attributes that have been revoked, therefore the attributes that need to be updated in ciphertexts and decryption keys. The symbol $|\mathcal{P}|$ is the number of attributes inside the policy $\mathcal{P}$, and the same applies for $|\gamma|$. The expression $|\gamma \cap \mathcal{A}_{rev}|$ is the number of attributes belonging to both γ and $\mathcal{A}_{rev}$, and the same applies to $|\mathcal{P} \cap \mathcal{A}_{rev}|$. The operations taken into account are pairings, exponentiations in $\mathbb{G}_0$, and exponentiations in $\mathbb{G}_1$. In all three schemes, we consider the worst-case scenario for the Decrypt primitive, which corresponds to a policy with an AND root having all the attributes in γ as children. This represents the worst-case since it forces the consumer to execute the DecryptNode subprimitive on every node of the policy, thus maximizing the computational cost.

From the table, we can see that SEA-BREW and BSW-KU pay the flexibility of the CP-ABE paradigm in terms of computational cost, especially concerning the Encrypt and Decrypt operations. However, this computational cost is the same as that in Bethencourt et al.'s scheme (Bethencourt et al., 2007), which has proven to

Primitive	Pairings	$\mathbb{G}_0$ exp.'s	$\mathbb{G}_1$ exp.'s
SEA-BREW			
Encrypt	-	$2\|\mathcal{P}\|$	1
KeyGen	-	$2\|\gamma\|+1$	-
Decrypt	$2\|\mathcal{P}\|+1$	-	$\|\mathcal{P}\|+2$
UpdateCP	-	1	-
UpdateDK	-	1	-
BSW-KU			
Encrypt	-	$2\|\mathcal{P}\|$	1
KeyGen	-	$2\|\gamma\|+1$	-
Decrypt	$2\|\mathcal{P}\|+1$	-	$\|\mathcal{P}\|+2$
UpdateCP		(not available)	
UpdateDK	-	$2\|\gamma\|+1$	-
YWRL (Yu et al., 2010a)			
Encrypt	-	$\|\gamma\|$	1
KeyGen	-	$\|\mathcal{P}\|$	-
Decrypt	$\|\mathcal{P}\|$	-	$\|\mathcal{P}\|$
UpdateCP	-	$\|\gamma \cap \mathcal{A}_{rev}\|$	-
UpdateDK	-	$\|\mathcal{P} \cap \mathcal{A}_{rev}\|$	-

Table 5.3
Comparison between SEA-BREW, BSW-KU, and YWRL schemes in terms of the computational cost of the primitives. For the YWRL scheme, the UpdateCP and the UpdateDK primitives correspond respectively to the AUpdateAtt4File and AUpdateSK of the original paper.

be supportable by mobile devices (Ambrosin et al., 2015) and constrained IoT devices (Girgenti et al., 2019), as presented in Chapter 6. Note that our UpdateCP and UpdateDK primitives have a cost that is independent of the number of attributes in the revoked decryption key. Such primitives require a single $\mathbb{G}_0$ exponentiation and many $\mathbb{Z}_p$ multiplications equal to the number of revocations executed from the last update of the ciphertext or the decryption key. However, the latter operations have a negligible computational cost compared to the former; therefore, we can consider both primitives constant-time.

Since modern cloud services typically follow a "pay-as-you-go" business model, in order to keep the operational costs low, it is essential to minimize the computation burden on the cloud server itself. We investigated by simulations the cloud server computation burden of our Lazy PRE scheme compared to the YWRL one, which represents the current state of the art. We can see from Table 5.3 that in both SEA-BREW and YWRL, the cloud performs only exponentiations in $\mathbb{G}_0$.

The reference parameters for our simulations are the following ones. We simulated a system of 100k ciphertexts stored on the cloud server over an operation period of 1 year. We fixed an attribute universe of 200 attributes. We fixed a number

of 15 attributes embedded in policies and attribute sets. We modeled the requests with a Poisson process with an average of 50k daily requests. Finally, we modeled that several consumer leave procedures are executed at different instants, following a Poisson process with an average period of 15 days. To obtain more meaningful statistical results, we performed 100 independent repetitions of every simulation.

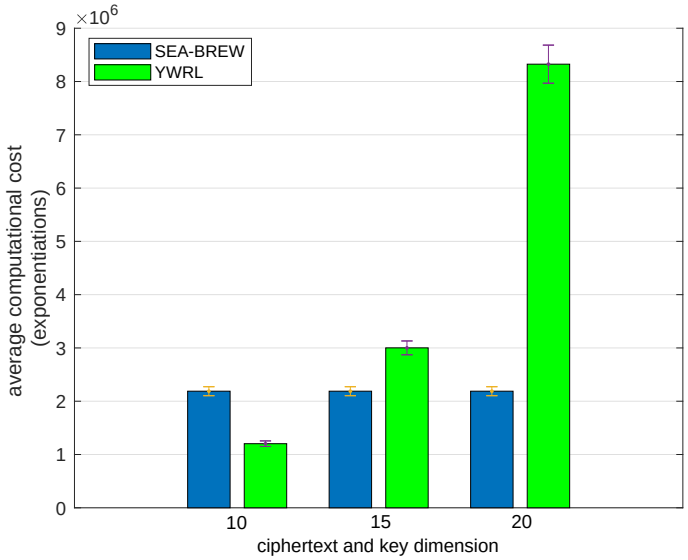


Figure 5.5. Average number of exponentiations over a year, varying policies, and attributes sets dimension. 95%-confidence intervals are displayed in error bars.

Fig. 5.5 shows the average number of exponentiations in $\mathbb{G}_0$ performed by the cloud server, with respect to the number of attributes in ciphertexts and decryption keys, which is a measure of the complexity of the access control mechanism.

As we can see from the figure, SEA-BREW scales better than the YWRL as the access control complexity grows. This is because in the YWRL scheme every attribute has a singular and independent version number, and the revocation of a decryption key requires updating all the single attributes in the key. The cloud server re-encrypts a ciphertext with many operations equal to the attributes shared between the ciphertext and the revoked key. Such a number of operations grows linearly with the average number of attributes in ciphertexts and decryption keys. On the other hand, in SEA-BREW, the version number is the same for all the attributes, and the revocation of a decryption key requires updating only it. The cloud server re-encrypts a ciphertext with an operation whose complexity is independent of the number of attributes in the ciphertext and the revoked key.

Fig. 5.6 shows the average number of exponentiations in $\mathbb{G}_0$ performed by the cloud server with respect to the average daily requests, which is a measure of the system load. The number of attributes in ciphertexts and decryption keys is fixed to

15.

Fig. 5.6 shows the average number of exponentiations in $\mathbb{G}_0$ performed by the cloud server with respect to the average daily requests, which is a measure of the system load. The number of attributes in ciphertexts and decryption keys is fixed to 15. As we can see from the figure, the computational load on the cloud server

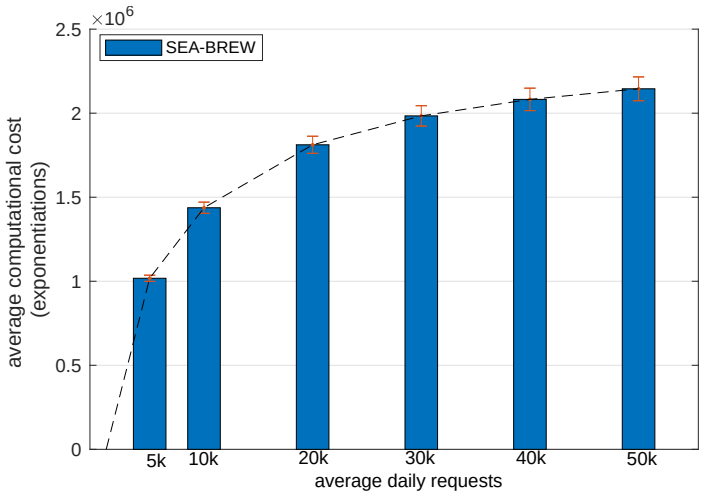


Figure 5.6. Average number of exponentiation over a year, varying the average daily requests.

grows sub-linearly with respect to the increase of requests. This behavior allows SEA-BREW to scale well also with a high number of requests.

5.7 Answer

How can We Improve the Original CP-ABE?

We efficiently provide new features! SEA-BREW can revoke or renew multiple decryption keys by sending a single broadcast message over a WSAN. Non-revoked users can upgrade their keys with a single message that does not need confidentiality. Moreover, old ciphertexts are re-encrypted to prevent revoked keys from accessing them.

Chapter 6

How Much Classical ABE Schemes Actually Impact Constrained IoT Devices?

Recent advancements in wireless communication standards and embedded computing are fostering the creation of novel smart computing systems, which are rapidly getting real in heterogeneous contexts, from personal to industrial.

The majority of IoT devices are resource-constrained, i.e., characterized by scarce capabilities and features. IoT devices are typically implemented through low-cost embedded systems that have reduced computing and storage capabilities and are often battery-powered. The scarcity of resources on those devices is currently driving the definition of specific network protocols that can accommodate the reduced features offered by them. An example is the Constrained Application Protocol (CoAP) (Shelby et al., 2014), which is an application protocol tailored to allow applications to communicate with constrained devices.

To compensate for the limited capabilities of IoT devices, more complex architectures are usually put in place to allow the implementation of advanced services on top of the functionalities they offer. IoT systems are usually implemented in a multi-layered fashion, in which IoT devices are integrated into cloud-computing platforms. Intermediate devices such as gateways or brokers are usually installed to implement functionalities like protocol translation, data dispatching, or to support the execution of simple applications that require proximity with the IoT devices due to time constraints.

In this complex architecture, data generated by IoT devices can be processed by multiple heterogeneous entities, which can be either different applications interested in analyzing the data or the above-mentioned intermediate entities. In this context, novel encryption mechanisms are required to enforce security and guarantee fine-grained access control over data. The latter, in particular, is a critical re-

quirement to tune the amount of information that can be accessed by each entity handling the data. For instance, while applications should have complete access to the data generated by IoT devices, an intermediate entity, like a broker, should have access only to the minimum set of information required to implement its functionalities (Zickau et al., 2016). Current security methods adopted in IoT are based solely on encrypted channels between the broker and the IoT devices, plus optionally an access control mechanism enforced by the broker. However, secure channels do not prevent the broker from accessing all data in the clear, and thus, an attacker that compromises the broker can jeopardize the confidentiality of the whole communication system.

Instead, ABE allows the broker to manage only encrypted data so that an attacker compromising the broker cannot break data confidentiality. ABE adoption is foreseen as a crucial technique to handle many security issues in different scenarios, ranging from healthcare systems to online social networks.

The academic literature has partially assessed the feasibility of adopting ABE in different contexts, from fully-fledged embedded IoT systems (Ambrosin et al., 2016) to smartphones (Ambrosin et al., 2015). Its adoption in *constrained* IoT devices, instead, has not been investigated so far. Understanding the feasibility limits of adopting ABE in constrained IoT devices allows us to understand its current applicability to a vast range of IoT applications. In this chapter, we carry out an extensive evaluation of ABE performance in constrained IoT devices. Specifically, we assess the performance of different ABE schemes on different devices with different memory and computational capabilities. We implemented two representative ABE schemes and tested their performance on two popular IoT platforms, the ESP32, and the RE-Mote. We selected these two IoT platforms for the evaluation as they are representative of the devices currently available in the market. The ABE schemes have been configured considering a worst-case scenario, often adopted in literature, in order to check the feasibility of adopting ABE on constrained devices in the most challenging conditions. Our performance evaluation shows that ABE significantly impacts the lifetime of battery-powered devices, especially when a high number of attributes (i.e., 20-50) is used in ciphertexts. However, if we assume to employ fewer attributes (up to 10) and leverage hardware elliptic-curve cryptographic acceleration, which is present on some platforms (e.g., RE-Mote), ABE can indeed be used by devices with very limited memory and computing power. We also obtain a significant yet tolerable battery lifetime reduction.

The worst-case configuration considered in our experiments is often adopted in literature; however, it might not entirely represent the real working conditions of sensors employing ABE, as ABE configuration adopted in real use cases can often give better performance than the worst-case configuration. For this reason, we propose a novel benchmark method that allows us to estimate the average performance

with better accuracy than the worst-case analysis. Our benchmark method applies to any ABE scheme, and it provides a more realistic performance evaluation because it captures the average case. We exploited such a method to complete our evaluation. We show that the worst-case analysis significantly overestimates the processing time and energy. For example, with RE-Mote under some configurations, the energy consumption estimated from the average case is 67% less than that estimated from the worst case.

The rest of the chapter is organized as follows. Section 6.1 overviews related work. Section 6.2 introduces a set of reference use cases and threat models. Section 6.3 introduces the hardware platforms and the methodology adopted in our experiments. Section 6.4 presents the experimental results for the ESP32 and the RE-Mote platforms. Section 6.5 presents the novel benchmark method for estimating the average-case time and energy consumption and the results obtained with it. Finally, Section 6.6 ends the chapter, answering the question inquired.

6.1 Related Work

The application of ABE schemes to implement fine-grained access control and confidentiality has been already proposed in different contexts, but none of those studies focused on assessing the cost of introducing ABE in practice. Instead, an evaluation of the adoption of ABE has been carried out in the following works.

In (Wang et al., 2014), the authors made the first benchmark of a KP-ABE scheme and a CP-ABE scheme in terms of execution time, energy consumption, memory usage, data overhead. Their benchmark is carried out on a PC-class device (Intel Quad-Core i7 @ 1.60GHz) and a mobile device (Intel Atom Z2460 @ 1.60GHz smartphone). In (Ambrosin et al., 2015), the authors evaluated the feasibility of adopting ABE on smartphone devices. Specifically, the authors developed an ABE library for the Android operating system and then they evaluated its performance through real experiments. In (Kuehner and Hartenstein, 2016), the authors carried out a comprehensive analysis of different ABE schemes with respect to their application for decentralized and secure data sharing. In particular, they performed a realistic estimation of the resource consumption and workload exploiting real-world system traces. Their evaluation considered heterogeneous devices, namely a laptop and a smartphone.

The results of (Ambrosin et al., 2015; Wang et al., 2014; Kuehner and Hartenstein, 2016) confirmed the possibility of using ABE on laptops and smartphones, showing that such devices have an acceptable amount of resources to implement ABE schemes and the resulting energy cost is acceptable. The following works focused instead on more constrained devices. In (Zickau et al., 2016), the authors surveyed the various existing implementations of ABE schemes and, as a side con-

tribution, they performed a benchmark on a single-board computer (Raspberry Pi 2 @ 900MHz) of various ABE schemes with the Charm library, used for fast prototyping of cryptographic schemes. Similarly, in (Ambrosin et al., 2016) the authors assess the feasibility of using ABE in single-board computers, namely Raspberry Pi and Intel Edison. Experimental results demonstrate that exploiting ABE in such systems is feasible, although they also highlight that future works should improve its efficiency. Notably, the authors of (Zickau et al., 2016) and (Ambrosin et al., 2016) do not focus on *actually* constrained IoT devices, but on more powerful platforms that have resources comparable with smartphone devices. Specifically, single-board computers like Raspberry Pi and Intel Edison have enough resources to run a fully-fledged operating system. However, it is not clear from their results whether ABE schemes are feasible on far more constrained devices, which is the focus of our work.

This chapter considers constrained devices with significantly less memory/computing capabilities, i.e., boards equipped with a microcontroller with less than 1MB of RAM. Those devices, very popular in IoT solutions, cannot support the execution of a fully-fledged OS and usually run an ad-hoc OS with limited features. Our analysis shows that, if we assume to employ simple access policies and leverage hardware elliptic-curve cryptographic acceleration, ABE can indeed be adopted on devices with very limited memory and computing power.

As a final note, all the previous papers (Wang et al., 2014; Zickau et al., 2016; Ambrosin et al., 2015, 2016; Kuehner and Hartenstein, 2016) evaluate the decryption performance considering the worst-case scenario of policies using only "AND" operators. This chapter shows that when relying on worst-case decryption, the processing time and energy consumption are significantly overestimated. We thus propose a novel benchmark method that allows us to evaluate the average decryption performance instead of the worst-case one, complementing our experimental analysis.

6.2 Use Case

A popular application scenario for ABE that involves constrained devices is the medical field. Future medical systems will broadly adopt Wireless Body Area Networks (WBANs) (Picazo-Sanchez et al., 2014) to collect data. Patients that require continuous monitoring, for instance, will be equipped with wearable and/or implantable sensors, which collect biometric parameters for real-time monitoring, e.g., to ensure a rapid response in case of an emergency or automate the administration of treatments. These WBANs produce highly sensible data consumed by other constrained devices, e.g., an insulin pump that analyzes data from other biomedical sensors to select the proper dose, or by humans, e.g., a doctor that remotely checks the status of a patient. In this context, data should be protected from unauthorized

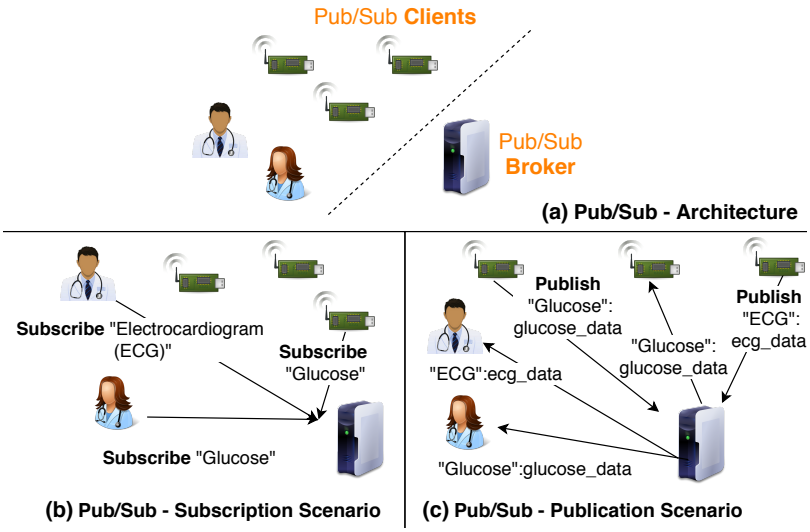


Figure 6.1. Publish/subscribe architecture and mechanism.

access through encryption. However, since multiple recipients are involved, fine-grained access control is mandatory to regulate which piece of information can be accessed by which user or device of the system. For instance, a glucose sensor can be programmed to encrypt its measurements to allow only the insulin pump and the patient's physician to access them.

In such applications, the information is often shared using a *publish/subscribe* system. Publish/subscribe is a common information-flow pattern adopted by different IoT application protocols, such as the Message Queue Telemetry Transport (MQTT) protocol (Hunkeler et al., 2008) and the Constrained Application Protocol (CoAP) (Koster et al., 2019), to decouple the producer of information to the consumer. The overall architecture of a publish/subscribe system is depicted in Fig. 6.1(a). On one side, we have a set of constrained IoT devices, e.g., sensors or actuators, and users that behave as publish/subscribe clients and produce and consume messages, e.g., periodic updates on a physical measurement. On the other side, we have a broker, a full-resource device responsible for receiving, storing, and dispatching messages. An IoT device or a user that is interested in receiving messages on a given *topic* contacts the broker to issue a subscription to that topic (Fig. 6.1(b)). Whenever an IoT device generates new data for a given topic, it sends a message to the broker. The broker is responsible for dispatching messages to all the subscribers (Fig. 6.1(c)). This approach allows us to overcome the main limitations that characterize constrained IoT devices. Firstly, their limited memory and computational power capabilities allow them to interact only with one application at a time. The adoption of a broker, instead, allows such devices to be used by multiple applications simultaneously, thanks to the dispatching capabilities of the broker. Secondly, the publish/subscribe architecture facilitates communication with battery-

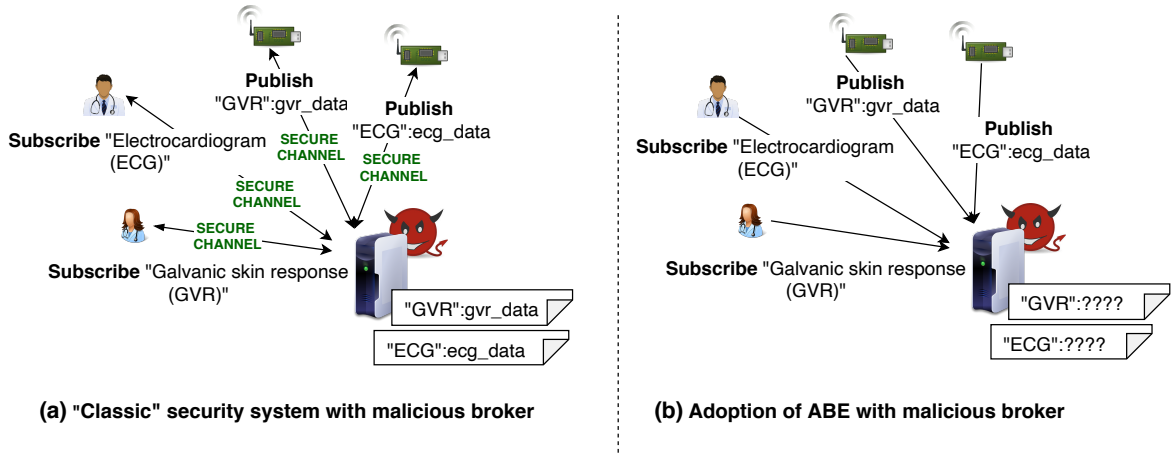


Figure 6.2. Malicious broker with traditional ABAC mechanism (a) and with ABE (b).

powered IoT devices. To minimize energy consumption, such devices should often operate in power-saving mode in which they turn off their radio. In a publish/subscribe architecture, the broker can store the generated messages, thus allowing the IoT devices to go in sleep mode without compromising information availability.

The core of this architecture is the broker, which can access all the messages. Such entity is often outsourced, i.e., it is deployed on external infrastructure, e.g., a cloud computing platform, or is completely operated by external entities, e.g., cloud computing providers, which offer MQTT brokers as a service to customers. For those reasons, a broker is not only subject to external attackers, but even an untrusted third party could directly manage it.

In this context, we consider an adversary capable of compromising the broker. However, it is worth highlighting that the same considerations apply if the broker's owner tries to access data dispatched by the broker itself. Thus, hereafter we refer to a broker that is either compromised or owned by an untrusted third party server as a *malicious broker*. A traditional *Attribute-Based Access Control* (Hu et al., 2015) (ABAC) mechanism enforced by the broker would have exposed data to a confidentiality risk in case of a malicious broker. This is because those "classic" ABAC security systems are based entirely on secure channels (e.g., TLS), in which the broker establishes a secure channel with each entity involved in the architecture. In such systems, the broker can access all the messages; thus, it is a single point of trust, as shown in Fig. 6.2(a).

ABE is a possible implementation of the ABAC methodology. In fact, with ABE, a malicious broker cannot do much since it stores and dispatches messages in an encrypted fashion so that the broker is not able to decrypt them (see Fig. 6.2(b)). ABE itself gives such resistance to this adversary: ABE ensures that the broker has access only to metadata, e.g., the data type or the topic, but not to the data itself, which is stored and dispatched by the broker in an encrypted form.

Note that ABE does not preclude the possibility to use *also* secure channels, for example, as a means of authenticating the messages. In addition to this, ABE enforces fine-grained access control on encrypted data, thus preventing malicious or compromised applications from accessing unauthorized data.

6.3 Experimental Setup

In this section, we present the experimental setup adopted for our performance evaluation. Specifically, in the following, we first introduce our reference ABE schemes, the adopted hardware and software platforms, and then present the methodology.

Reference ABE Schemes

In this chapter, we focus on two representative ABE schemes, namely: (i) the Goyal-Pandey-Sahai-Waters scheme (Goyal et al., 2006b)[1] (throughout referred to as "GPSW", for brevity), which has been the first proposed KP-ABE scheme in the literature; and (ii) the Bethencourt-Sahai-Waters scheme (Bethencourt et al., 2007) ("BSW"), which has been the first proposed CP-ABE scheme in the literature.

In both considered schemes, the most expensive operation performed by the encryption algorithm is the *point-scalar multiplication*, which is an elliptic-curve operation (see (Hankerson et al., 2006) for details). In particular, the GPSW scheme performs one point-scalar multiplication for each attribute in the attribute set. The BSW scheme performs two point-scalar multiplications for each leaf in the policy tree, and for each internal node of the policy tree, the BSW scheme creates a random polynomial of zero degree if the node is an OR operator, or degree equal to the number of children minus one if the node is an AND operator. Furthermore, the BSW scheme also performs a hashing of each attribute name on an *elliptic-curve group* as a first stage of the encryption operation. Since this hashing operation has a non-negligible impact on encryption performance, and since it can be easily precomputed given the set of attribute names that the encrypting device uses, we chose *not* to include the hash operations in our performance evaluation. In both GPSW and BSW schemes, the most expensive operation performed by the decryption algorithm is the *bilinear pairing*, which is an expensive cryptographic operation (see again (Hankerson et al., 2006) for details). We recall that the decryption algorithm does not have to visit all the tree nodes, but only a subset of them that is necessary to reach the root. The GPSW scheme performs one bilinear pairing for each visited leaf in the policy tree.

[1] In the cited paper, the authors present two schemes, offering a small and a large attribute universe, respectively. We refer to the first one, the most lightweight of the two, thus suitable for very constrained devices.

The BSW scheme performs two bilinear pairings for each visited leaf in the policy tree.

Hardware and Software Platforms

As an example of constrained IoT devices, we exploited the ESP32 and the RE-Mote boards in our experiments. We have chosen those two IoT platforms as they are representative of two different categories of IoT devices currently available in the market, i.e., very constrained devices (like the RE-Mote board) designed to operate on batteries characterized by scarce memory/computing capabilities and equipped with an ultra-low-power wireless transceiver (e.g., IEEE 802.15.4) and constrained devices with slightly more memory and computing capabilities equipped with a WiFi transceiver (like the ESP board).

The ESP32 (Espressif, 2020) is an IoT platform produced by Espressif Systems that is growing in popularity due to its low cost, high availability, and rich set of features. A dual-core Xtensa LX6 microprocessor at 240 MHz designed to have ultra-low-power consumption is the system's core. The board is equipped with 520 KB of SRAM and 448 KB of programmable ROM. It includes both WiFi and Bluetooth connectivity to accommodate a wide range of IoT use cases. The chip includes cryptographic hardware acceleration support for AES, SHA2, and RSA algorithms. Unfortunately, it does not provide hardware acceleration for elliptic-curve cryptography (ECC) algorithms, the most burdensome ones in ABE. The board is natively supported by FreeRTOS[2]. FreeRTOS is a popular Operating System (OS) for embedded devices that supports a wide range of microcontrollers. It is written in the C language and provides support for multi-threaded programming. Compared with fully-fledged OSs, FreeRTOS lacks support for many advanced features and only provides basic support for memory management and networking operations. Basic support for cryptographic operations is included, such as the popular wolfSSL library.

The RE-Mote (Zolertia, 2020a) is a platform jointly designed by universities and industrial partners and produced by Zolertia, which targets industrial-grade design and ultra-low power consumption. The board is equipped with the Texas Instruments CC2538 ARM Cortex-M3 System on Chip (SoC) working at 32 MHz, and it can provide ECC hardware acceleration, i.e., it can accelerate point-scalar multiplication and point addition. Note that many other SoCs provide ECC hardware acceleration for standard elliptic curves only, which cannot be used for ABE, and thus they are useless for our aims. In contrast, the CC2538 SoC can accelerate the *generic* elliptic curve, including those suitable for ABE (*paring-friendly curves*), and thus it can be exploited to boost ABE operations. The RE-Mote board has native

[2]FreeRTOS, https://www.freertos.org, accessed: 2019-12-06

Table 6.1
ESP32 and RE-Mote specifics.

	ESP32	RE-Mote
CPU	Tensilica Xtensa dual-core LX6 microprocessor, operating at 240 MHz	ARM Cortex-M3, operating at 32 MHz
Radio	Wi-Fi: 802.11 b/g/n - Bluetooth: v4.2 BR/EDR and BLE	IEEE 802.15.4 (ISM 2.4-GHz and 863-950-MHz) & Zigbee
RAM	448 KB flash and 520 KB RAM	512 KB flash and 32 KB RAM

support for different OSs for IoT devices, including the Contiki-NG OS[3]. Compared to ESP32, RE-Mote offers a comparable programmable ROM size (512 KB), but it has a smaller SRAM size (32 KB), making storing cryptographic information on the device challenging.

Table 6.1 lists the specifications of the two adopted boards.

To carry out our performance evaluation, two existing libraries for Linux OS implementing the two considered ABE schemes have been ported to FreeRTOS and Contiki-NG, namely, the *libcelia* library[4] which implements the GPSW scheme, and the *libbswabe* library[5] which implements the BSW scheme. We configured both libraries to use pairing-friendly elliptic curves with embedding degree 2 and effective security strength of 80 bits, equivalent to a 1024-bit RSA encryption. The libraries have been modified to suit the features offered by FreeRTOS and Contiki-NG. Specifically, the major modifications consisted into: (i) removing any usage of GLib, which is unavailable in both OS, and (ii) adapting the code to use the wolfSSL library on FreeRTOS and mbedTLS library on Contiki-NG[6] instead of the more popular OpenSSL, which is not supported on FreeRTOS and Contiki-NG.

Methodology

To assess the performance of the two considered ABE schemes on ESP32 and RE-Mote, two simple main programs, one for each library, have been developed to perform a sequence of operations. After the initial **Setup** algorithm, which generates a master key and an encryption key, the program generates a decryption key with the **KeyGen** algorithm. Then, it creates a random 4-byte string that emulates the message to be transmitted. After that, the program encrypts the message and subsequently decrypts it. For each operation, the program prints a message over the serial connection. This allows us to measure the time required to perform every operation. We adopted a high-precision USB power meter to measure the energy consumption. Specifically, we used the AVHzY USB Power Meter Tester[7], which supports automatic data collection from an attached PC and allows measurements

[3]Contiki-NG, https://contiki-ng.org/, accessed: 2019-12-06

[4]Libcelia library: https://bit.ly/33lESZN, accessed: 2019-12-06

[5]Libbswabe library: https://bit.ly/2QRtEJV, accessed: 2019-12-06

[6]Two different TLS/SSL libraries for the two ESP32 and RE-Mote implementations have been considered considering their availability on the FreeRTOS and Contiki-NG OSs, respectively.

[7]Power Meter Tester product page: https://goo.gl/vQDyac

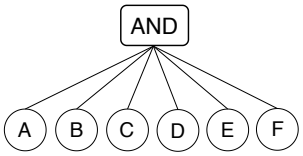


Figure 6.3. Example of flat policy.

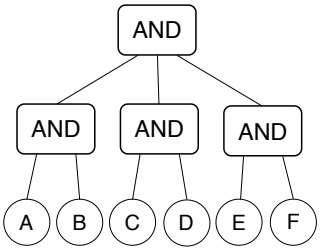


Figure 6.4. Example of 3-level policy.

with a resolution of 10^{-15} mWh. The comparison between the log from the board and the readings from the power meter allowed us to measure the energy consumed for each specific operation.

To evaluate the two ABE schemes, we considered the following metrics:

- *Encrypton/decryption time* (s), defined as the time required to execute an **Encrypt**/**Decrypt** algorithm.
- *Encrypton/decryption energy consumption* (mWh), defined as the overall energy consumed by the board to execute an **Encrypt**/**Decrypt** algorithm.

An increasing number of attributes, from 5 to 50, has been considered for encryption. Such a number represents the number of leaves in the policy tree for the CP-ABE scheme (BSW) or the size of the attribute set for the KP-ABE scheme (GPSW). These two quantities do not have the same meaning because the leaves of a policy tree represent the formal arguments of the policy, whereas the attribute set represents the actual arguments used to evaluate a policy. However, they both give a measure of the complexity of the access control rules involved in an application. For each experimental scenario, ten independent replicas of the experiment have been executed. Our results report the average value of the measurements and the 95% confidence interval. Note that the number of replicas and the number of configurations in terms of attribute number considered in our experiments are limited. This is because some steps for the execution of the experiments cannot be automated, thus greatly increasing the time required for their execution. Regarding the number of replicas, ten replicas should, however, be enough to get statistically sound averages due to the small variability of the results. This is also suggested by the very small confidence intervals, which are almost unnoticeable in the plots.

The number of operations performed by the **Decrypt** algorithm grows up linearly with the number of visited leaves and the number of visited internal nodes (including the root). This means that policies with the same number of leaves but a different "shape" can perform differently. We shaped the access policies as *flat policies*, with a single internal node (the root) associated with an AND operator and many child nodes, one for each attribute. Fig. 6.3 shows an example of flat policy.

Flat policies represent the worst case for decryption algorithms. Indeed, with a flat policy, the **Decrypt** algorithm is forced to visit all the leaves of the policy tree, and the leaf visit is generally the most expensive operation in decryption. In some experiments, we also used access policies shaped as *3-level policies*. A 3-level policy is semantically equivalent to a flat policy, but it has an additional intermediate level between the leaves and the root. The nodes in this intermediate level are AND operators and they have no more than two leaves as children. Hence, the root has only $\lceil n/2 \rceil$ child nodes, where n is the number of leaves, opposed to n child nodes of a flat policy. Fig. 6.4 shows an example of 3-level policy, which is equivalent to the flat policy shown in Fig. 6.3. The 3-level policies are useful to test algorithms whose efficiency depends on the average number of children of internal nodes. Indeed, the internal nodes in a 3-level policy have in general fewer children than those in the equivalent flat policy.

Drawbacks of Flat and 3-Level Policies

Intuitively, results measured by using flat and 3-level policies give an *overestimation* of ABE resource consumption since both types of policy are, by definition, the decryption's worst-case scenario. However, policies should be much more diverse in the real world, as they reflect the complexity of a human's access rights (KP-ABE), or they describe the vast range of entities that can access a single piece of information (CP-ABE). In practice, this variety and flexibility are rendered by building access policies also using OR gates. Such gates drastically diminish the number of nodes and leaves that need to be evaluated to satisfy a policy, therefore decreasing time, resources, and energy depleted by the sensors. Now, we anticipate the basic idea behind the *average-case scenario* (analyzed in Section 6.5), which allows us to estimate decryption performance more realistically. The average-case performance is measured by randomly generating many access policy/attribute set couples –ensuring for each couple that the generated attribute set satisfies the generated policy–, and then performing a decryption operation upon each couple. As in the worst case, we analyze the decryption time and the energy consumption needed to perform the decryption operation so that we can compare the results with the ones obtained considering the worst-case scenario. To do this, we perform decryption over several thousands of different access policy/attribute set couples. The average-case scenario and the results achieved accordingly to it are thoroughly described in Section 6.5.

6.4 Experimental Results

In this section, we present the results of our experiments. We first discuss the results obtained with the ESP32 boards and then analyze the results obtained with the RE-

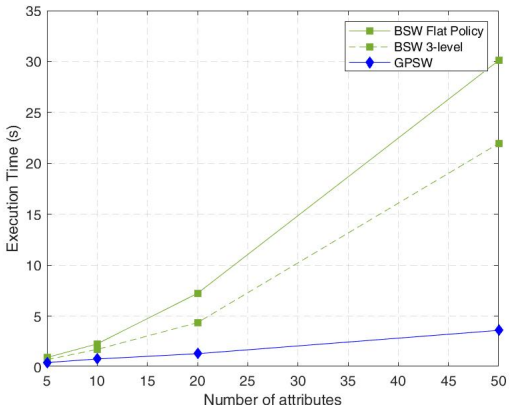


Figure 6.5. Encryption time. ESP32

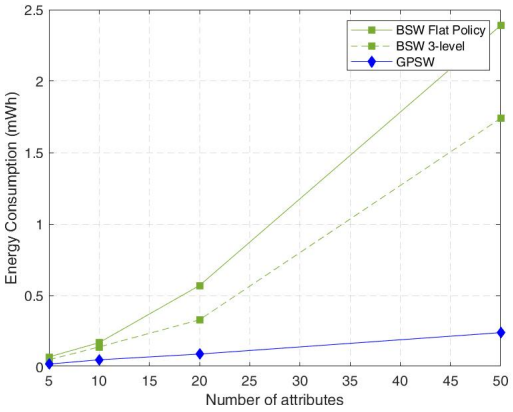


Figure 6.6. Encryption energy consumption. ESP32

Mote boards.

Results with ESP32 Boards

Encryption Time and Energy Consumption

Figs. 6.5 and 6.6 show the encryption time and the encryption energy consumption of the two considered schemes, as a function of the number of involved attributes. As we said before, such "involved attributes" assume a different meaning in CP-ABE schemes (i.e., BSW) and in KP-ABE schemes (i.e., GPSW). In this figure and the following ones, we put them in the same X-axis for the mere reason of saving space, but this should not be interpreted as a comparison between CP-ABE and KP-ABE schemes. The BSW scheme is quite expensive in encryption in terms of both time and energy consumption. This is because it performs two point-scalar multiplications for each leaf in the policy tree, and it generates a random polynomial for each internal node.

Despite many previous papers on ABE (Wang et al., 2014; Ambrosin et al., 2015, 2016) report an encryption time linear on the number of leaves for the BSW scheme, we experienced an over-linear time. This is mainly due to the random polynomial generation that the BSW scheme performs for each internal node of the policy tree. The complexity of generating such a random polynomial grows in an over-linear fashion with respect to the number of children of the internal node. To confirm this, we re-shaped the flat policy into an equivalent 3-level policy, in which each internal node has fewer children, and run an additional set of experiments with all the considered ABE schemes. In Figs. 6.5 and 6.6 we report the results of the BSW scheme with a 3-level policy. The results obtained with GPSW are omitted because both execution time and energy consumption for encryption do not deviate from the results obtained with the flat policy since, in KP-ABE, the policy is not involved in the

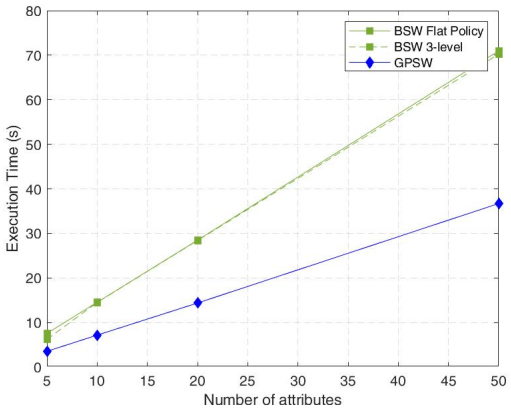


Figure 6.7. Decryption time. ESP32

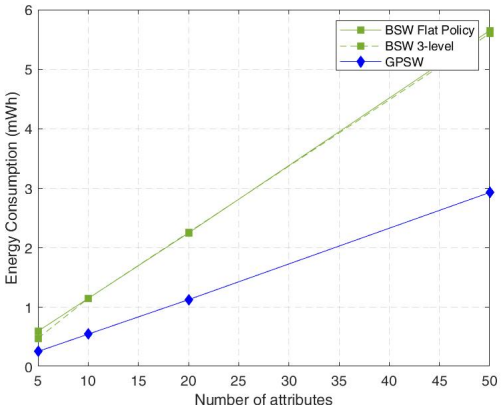


Figure 6.8. Decryption energy consumption. ESP32

encryption process. This is coherent with the fact that they do not generate random polynomials. As can be seen, the encryption time and the encryption energy consumption of the BSW scheme with a 3-level policy decrease sensibly compared to the flat policy. This result seems counter-intuitive since, with a 3-level policy, there are more polynomials to generate. However, it confirms that the over-linear cost of BSW encryption is due to the random polynomial generations, which grow in an over-linear fashion with respect to the number of children of each internal node. This also suggests that shaping the policies on many levels is a good practice to improve the performance of BSW encryption. The reason why the over-linear behavior does not appear in other previous performance evaluations of ABE available in the literature (Wang et al., 2014; Ambrosin et al., 2015, 2016) is that these papers also include the hashing of the attribute names within the encryption operation. Our performance evaluation does not include the hash operations because the hashes of the attributes' names can be precomputed, leading to a noticeable time saving.

Decryption Time and Energy Consumption

Regarding the decryption, Figs. 6.7 and 6.8 show the decryption time and energy consumption of the two considered schemes, as a function of the number of involved attributes, with flat policies. As expected, the BSW is quite an expensive scheme for decryption, too, because it performs two bilinear pairings for each visited node in the policy tree. No over-linear trend has been observed in BSW decryption, coherently with the fact that no random polynomials are generated. The GPSW scheme performs only one bilinear pairing per visited node, therefore it is not surprising that its consumption(both in terms of time and energy) is roughly half of the BSW scheme.

Figs. 6.7 and 6.8 show also the decryption time and the decryption energy consumption of the BSW scheme both with flat and 3-level policies. As it can be seen,

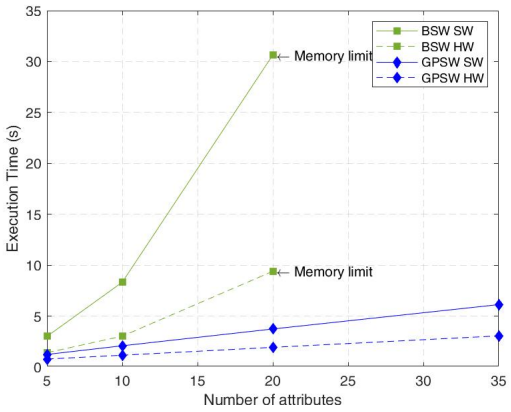


Figure 6.9. Encryption time. RE-Mote

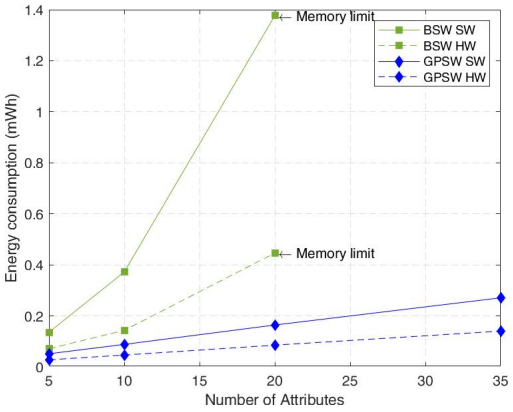


Figure 6.10. Encryption energy consumption. RE-Mote

the decryption time and energy consumption seem independent of the shape of the policy tree. This confirms that shaping the policies on many levels is a good practice in the BSW scheme because it improves encryption performance without decreasing the decryption performance.

Results with RE-Mote

Encryption Time and Energy Consumption

Figs. 6.9 and 6.10 show, respectively, the encryption time and energy consumption of the two considered ABE schemes, vs. the number of involved attributes, when using flat policies. Different from ESP32, RE-Mote provides ECC hardware acceleration, which can improve ABE performance. Therefore, we carried out experiments with and without ECC hardware acceleration. In the former case, the hardware support has been exploited to implement the ECC operations. In the latter one, such operations have been implemented via software, as for the experiments carried out with ESP32. The results obtained with hardware acceleration are represented with dashed lines. As can be seen, in some of the experiments, the encryption operation fails as a certain number of attributes is reached. Specifically, the BSW scheme failed when more than 20 attributes were adopted, while the GPSW scheme failed with more than 35 attributes. This is due to the limited SRAM available on the RE-Mote platform, which was insufficient to accommodate all the data structures required for the encryption operations.

As expected, the performance of the two ABE schemes exhibits the same trend as that obtained with ESP32 boards, both in terms of energy consumption and encryption time. By enabling hardware acceleration, both the encryption time and the corresponding energy consumption reduce significantly. For instance, if we consider the points of maximum reduction, the execution time is reduced by approximately

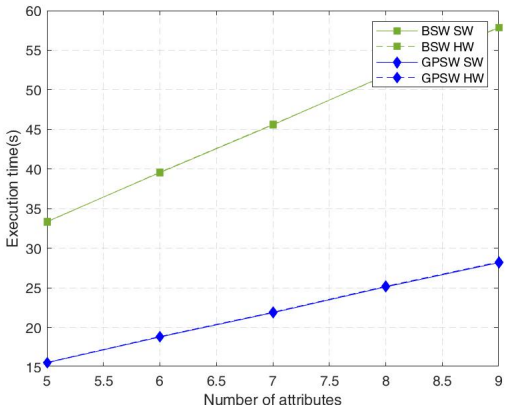


Figure 6.11. Decryption time. RE-Mote

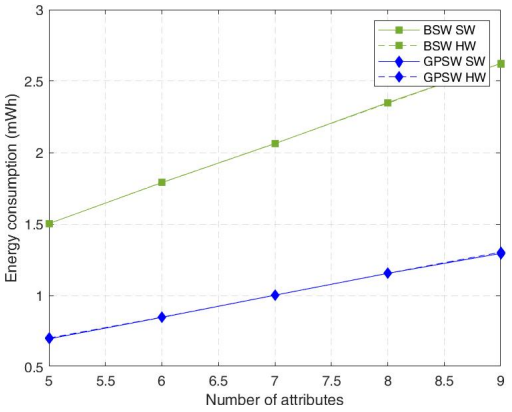


Figure 6.12. Decryption energy consumption. RE-Mote

70% for BSW obtained with 20 attributes, and by approximately 50% for GPSW obtained with 35 attributes. If we compare the results obtained with RE-Mote without ECC hardware acceleration and with ESP32, we can notice that the encryption time and the encryption energy consumption are higher with RE-Mote. This is expected as the microcontroller installed in RE-Mote has a lower frequency than the one installed in ESP32, thus resulting in higher times (and consequently higher energy consumption) for encryption.

Decryption Time and Energy Consumption

Figs. 6.11 and 6.12 show the decryption time and the decryption energy consumption, respectively, for RE-Mote when using flat policies. The GPSW scheme does not benefit from hardware acceleration. As it can be seen, it does not obtain any performance gain via ECC hardware acceleration compared to the software implementation of such operations. The BSW scheme does not benefit from hardware acceleration either. This is because the BSW and the GPSW schemes use burdensome bilinear pairing operations in decryption, which are not accelerated in RE-Mote. Though some prototypes of bilinear pairing hardware accelerators have been developed in literature (see (Salman et al., 2017) for an example), none of them is commercially available to the best of authors' knowledge. Therefore, we can expect that decryption operations in pairing-based schemes[8] do not benefit from hardware acceleration in any platform currently available in the market.

In decryption also, the maximum number of attributes is limited by the constrained memory of RE-Mote. In particular, the number of attributes for which the decryption can be performed does not exceed 9 and 10, depending on the scheme. When compared to encryption, the maximum number of attributes is significantly

[8]We recall that no EC-based or RSA-based Pairing-Free ABE scheme is secure (Herranz, 2020).

reduced in decryption. This can be explained by considering the larger data structures required by decryption operations with all the considered ABE schemes.

When devices with low memory are considered, the maximum number of attributes that can be employed is bounded to the amount of memory available. This is further exacerbated when data decryption has to be implemented on an IoT device, e.g., in an actuator, as the maximum number of attributes is significantly lower in this case for both schemes. Furthermore, the CP-ABE scheme can also be used by IoT-constrained devices.

Regarding the hardware acceleration in decryption, it is not beneficial since the dominant cost of decryption is, by far, the bilinear pairing.

Battery Lifetime Analysis

To analyze better the energy consumption that comes from the adoption of ABE schemes and to obtain a key performance indicator that directly evaluates the feasibility of adopting ABE in a real IoT scenario, we also estimated the lifetime of a battery-powered sensor node, i.e., the time of operation for the sensor node to exhaust its battery. The battery lifetime resulting from Figs. 6.13 and 6.14 is a simplified measure that does not consider the energy consumption for all the operations performed by a sensor node. Nevertheless, it can be helpful as a high-level comparison between different platforms. The evaluation is performed analytically by exploiting both the measurements from real experiments and the energy consumption data from the datasheet of ESP32 (Espressif, 2020) and RE-Mote (Zolertia, 2020b).

We consider a scenario in which a battery-powered sensor periodically collects a sample of a physical measure (e.g., the temperature in a room), encrypts it using ABE, and then transmits the encrypted message over a wireless connection. For the sake of brevity, we assume that the device only produces and encrypts data (i.e., it is a sensor).

In our analysis, both the ESP32 and the RE-Mote boards are assumed to be powered by two AA 1.5 V batteries, which can provide 2.85 Ah each. The evaluation of the energy consumption of the sensors included: (i) the overall energy consumed by the sensors for data encryption; (ii) the energy consumed for the transmission of the encrypted message over WiFi (for ESP32) or IEEE 802.15.4 (for RE-Mote); (iii) the energy consumed in between two subsequent transmissions, assuming that the sensors remain idle. We used the measurements obtained from the real experiments for the first component, while for the latter two, we exploited the power consumption values from the datasheet. To this aim, for ESP32, we assumed that the radio transceiver transmits data at 1 Mbps using DSS (Discrete Spread Spectrum) modulation, which results in a power consumption of 240 mW. Instead, RE-Mote transmits data at 250 Kbps, using DSS modulation again, resulting in a power consumption of 72 mW. Moreover, when idle, we assumed that both the boards enter a light-sleep

mode, which results in a power consumption of 2.64 mW for ESP32 and of 1.8 mW for RE-Mote (according to the corresponding datasheets). Finally, a value of 60 seconds is considered for the sampling period.

Figs. 6.13 and 6.14 show the estimated battery lifetime for the ESP32 and the RE-Mote boards, respectively, expressed in days with respect to the number of involved attributes. As we said in the previous section, such "involved attributes" assume a different meaning in CP-ABE schemes (BSW) and in KP-ABE schemes (GPSW). In this figure and the following ones, we put them in the same X-axis for the mere reason of saving space, but this should not be interpreted as a comparison between CP-ABE and KP-ABE schemes.

Analysis with the ESP32 board

In Fig. 6.13, the horizontal red line corresponding to 134 days represents the battery lifetime of an ESP32 sending data in the clear, i.e., without the cost of any encryption. With reasonably small attribute sets, i.e., ten attributes, a battery lifetime up to 62 days (54% decrease compared to no-encryption scenario) can be obtained with the GPSW scheme. On the other hand, with 10-attribute access policies, the BSW scheme experiences a lifetime of around 50 days. As expected, as the number of involved attributes increases, the battery lifetime reduces proportionally to the two ABE scheme's energy consumption. With an attribute set of 50 attributes, the resulting battery lifetime of GPSW is 25 days. On the other hand, with 50-attribute policies, BSW depletes the battery after a few days of use, but we can see that using 3-level policies can improve the performance, although slightly. The difference between the flat policy and the 3-level policy in BSW is that the over-linear cost of BSW encryption is due to the random polynomial generations, which grow in an over-linear fashion with respect to the number of children of each internal node. It also suggests that shaping policies on many levels while maintaining the same logical meaning is a good practice to improve performance.

Analysis with the RE-Mote board

Fig. 6.14 shows the battery duration of RE-Mote with flat policies. Again, the horizontal red line (about 198 days) represents the battery lifetime of a RE-Mote sensor sending data in the clear. It can be seen that, with ten involved attributes, the battery lifetime is up to 78 days (61% decrease) with hardware-accelerated GPSW and up to 35 days (82% decrease) with hardware-accelerated BSW. As for ESP32, the battery lifetime reduces proportionally with the number of attributes. If we compare the results with and without hardware acceleration, we can notice that hardware acceleration helps in improving the battery lifetime in both schemes. This is because the hardware support reduces the energy consumption for encryption.

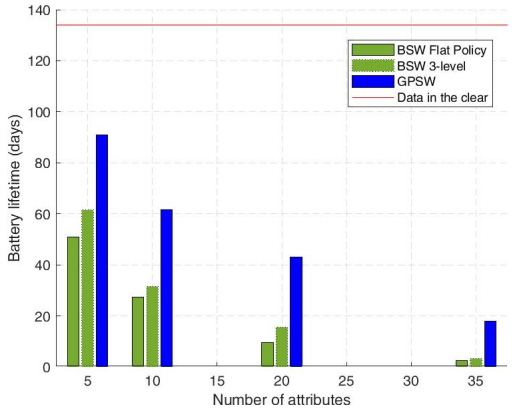


Figure 6.13. Battery lifetime. ESP32

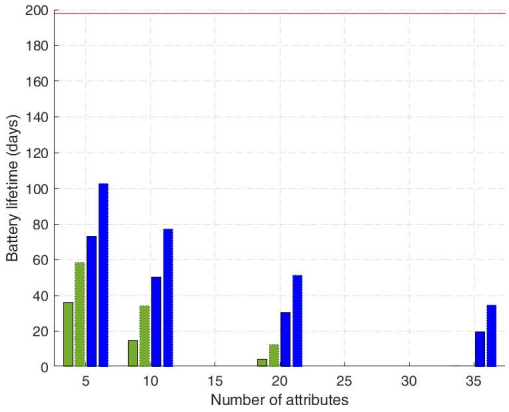


Figure 6.14. Battery lifetime. RE-Mote

Final Remarks

In our performance evaluation, we used two different IoT boards, namely ESP32 and RE-Mote. The results show that the CP-ABE scheme (BSW) is the most expensive for both platforms in terms of execution time and energy consumption, and this applies to both encryption and decryption operations. If we compare the results of the ESP32 and the RE-Mote (Figs. 6.5-6.12) without the use of hardware acceleration (only available on RE-Mote), we can conclude that the power consumption of BSW is much higher with the RE-Mote board. This is because their energy consumption is twice the one of the ESP32 board, and they also have a less powerful microcontroller; thus, encryption operations take more time to complete. On the other hand, the RE-Mote board can take advantage of the hardware acceleration for encryption operations, but in the case of the CP-ABE scheme, even though the gain obtained is significant (i.e., 70%), the power consumption is still in the same order of the ESP32 that cannot exploit hardware acceleration.

The hardware acceleration allows the RE-Mote to have encryption execution time and energy consumption in the same order as the ESP32, and, when possible, it should be exploited to fill the performance gap given by low-frequency microcontrollers. For decryption operations with the RE-Mote board instead, the hardware acceleration is not beneficial since the bilinear pairing is not accelerated. It is worth highlighting that due to the memory limit in the RE-Mote board, the decryption cannot be performed if the number of attributes is greater than 10. Thus, the RE-Mote board should not be used if the ABE policy's attributes are greater than 10.

To sum up, although the adoption of ABE has a noticeable cost in terms of energy consumption, the lifetime reduction can still be considered acceptable if we use small attribute sets in KP-ABE schemes, i.e., up to 20 attributes, and small policies in CP-ABE schemes, i.e., up to 5-attribute policies. When a higher number of attributes is considered, instead, the resulting battery lifetime is shortened significantly down to a value that could be unacceptable in scenarios in which a frequent battery re-

placement is not feasible or desirable. In any case, the use of hardware ECC acceleration, which is available on some platforms like RE-Mote, helps prolong the battery lifetime. We finally remark that some optimization techniques not considered in our analysis could be adopted to improve the sensor battery lifetime further. For example, it is often the case that all the pieces of data periodically transmitted by the sensor must be encrypted with the same attributes, as presented in (La Manna et al., 2019).

6.5 Average Decryption Performance Evaluation

In this section, we explain the needs and motivations that led us to develop this evaluation framework, the main idea behind it, how it works, and what we want to achieve. This section is organized in the following way: in Section 6.5 we present the limitations of the main ABE benchmarks proposed in the literature; in Section 6.5 we show in detail the construction and use of our framework; in Section 6.5 we argue the plausibility of the synthetic policies; and finally in Section 6.5 we show the results obtained evaluating the average case with the proposed method.

Motivation: Worst Case vs. Average Case

A recurring problem while benchmarking ABE schemes is how to evaluate the decryption performance correctly. Indeed, while the encryption performance, i.e., time and energy consumption, is quite predictable given the number of attributes in the ciphertext, the decryption performance highly depends on unpredictable factors, namely the structure of the access policy and the attribute set. To evaluate the decryption performance, it is a common practice in the literature (Wang et al., 2014; Ambrosin et al., 2015, 2016) to consider the worst case, which corresponds to the flat policy we used in this chapter. However, this worst case is infrequent in practice: while it can be helpful to validate the feasibility of adopting ABE in constrained devices by assessing its performance in the worst-case scenario, this configuration might be infrequent in a real scenario. Indeed, in a real scenario, access policies are supposed to have OR operators, and they are supposed to be structured in trees of many levels. However, it is extremely difficult to find large datasets related to *real* access policies and attribute sets used in companies and organizations, upon which evaluate the average case. This is more than understandable since those are security-critical information belonging to the company. For these reasons, we propose a method that allows us to create *synthetic* access policies in order to measure the decryption performance of an ABE scheme in the average case. We believe this approach to be more realistic than testing them in the worst-case scenario, as done in many previous works (Wang et al., 2014; Ambrosin et al., 2015, 2016) and can be

used to complement the experiments and provide a more comprehensive evaluation.

Framework Construction

Broadly speaking, the basic idea is to generate many random but realistic access policies and, for each generated policy, generate a random attribute set that fulfills such a policy. Then, for a KP-ABE scheme, we generate a decryption key associated with each access policy and a ciphertext associated with each attribute set. For a CP-ABE scheme, we do *vice versa*. Finally, we benchmark the decryption of the ABE scheme with the generated decryption keys and ciphertexts, and we average the results, thus obtaining the decryption performance in the average case.

More in detail, we generate a random policy having n attributes according to the following steps.

1. We assign a conventional name to each of the n attributes, say A, B, etc., we build a leaf for each attribute, and we fill a *parent-less node set* $\mathcal{N}$ with all these leaves. Through the algorithm, the parent-less node-set will contain the nodes of the policy we are creating that do not have a parent node yet. Since the random policy is still a collection of unstructured leaf nodes at this stage, the parent-less node-set contains all of them. The random policy will be a tree at the end of the algorithm, and the parent-less node-set will contain only the root.

2. We randomly select each node in the parent-less node set with a given probability p_c, and we define the set of selected nodes as $\mathcal{N}_c$. If $\mathcal{N}_c$ contains less than two nodes, we repeat the node selection procedure until at least two nodes have been selected.

3. We build a new node having the nodes in $\mathcal{N}_c$ as children, and we assign a conventional name to it, say N_1. The nodes created in the successive iterations of the algorithm will be N_2, N_3, etc. We choose the Boolean operation associated with the new node to be AND with a given p_{AND} probability, or OR with $1 - p_{AND}$ probability.

4. We modify the parent-less node-set by removing the nodes in $\mathcal{N}_c$ and adding the new node.

5. If the parent-less node-set now contains only one node ($|\mathcal{N}| = 1$), we terminate the algorithm. The random policy is fully built. Otherwise, we repeat from Step 2.

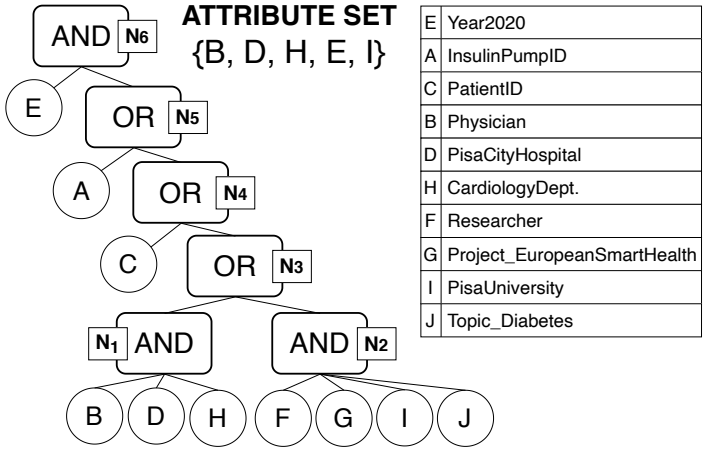


Figure 6.15. Example of random policy and random fulfilling attribute set, and a possible attribute interpretation considering a medical scenario.

The parameter $p_c \in (0, 1]$ influences how many children each node has on average, and thus the average height of the generated policy tree. The higher p_c is, the more nodes will be selected at each algorithm iteration so that the final policy will be more "flat". By choosing $p_c = p_{AND} = 1$, the method always generates flat policies, which represent the worst case for decryption.

After the random policy is generated, we need to generate a random attribute set that fulfills the policy to perform a decryption operation. We do it in the following way.

1. We randomly select some attributes in the n attributes used in the previous algorithm. Each attribute has a probability to be selected p_a.

2. If the selected attributes fulfill the synthetic policy, we terminate the algorithm. The attribute set is fully built. Otherwise, we repeat from Step 1.

The parameter $p_a \in (0, 1]$ influences how many attributes are included in the attribute set. The higher p_a is, the larger the attribute set will be on average. Fig. 6.15 shows an example of random policy with 10 attributes and $p_c = p_{AND} = 0.5$, and a random attribute set that fulfills it, both created with the aforementioned method. Note that the created policy has many levels and different Boolean operators, which recalls real-world access policies written by system administrators. Note also that the created attribute set is suboptimal to decrypt since, for example, the attribute set $\{A, E\}$ can fulfill the same policy by visiting fewer leaves and fewer internal nodes. Suboptimal attribute sets are extremely likely to appear in a real-world scenario.

Realism of the Synthetic Policies

The best way to corroborate the plausibility of synthetic access policies would be to analyze a dataset of real access policies. In this case, we can extract and compare various parameters like the depth of the policy tree, the frequency of the AND/OR gate, the distributions of specific attributes, and many others. Unfortunately, as we said, companies and organizations are reluctant to disclose such precious information. However, we provide an example to support the idea that any synthetic policy can be mapped to a real world-policy. We do so by showing that it is possible to give realistic meaning to the randomly generated policy of Fig. 6.15, considering the medical use case introduced in Section 6.2. Suppose that Pisa's city hospital provides a patient with a smart sensor that continuously monitors the blood glucose level. The same patient has a smart insulin pump that can be activated by the data received from such a smart sensor. Data produced by the smart sensor can also be read by the patient and any physician working at Pisa's city hospital inside the cardiology department. Moreover, the patient also gives his consent to process personal data to Pisa University researchers who participate in the "EuropeanSmartHealth" project in the "Diabetes" work package. Finally, keys are renewed every year for security reasons, so each key has an attribute associated with the current year. Considering the CP-ABE paradigm, the policy that the smart sensor must enforce on its produced data has the shape of the random policy in Fig. 6.15 (refer to the table on the right of the figure for the attribute meanings). Furthermore, the randomly generated attribute set describes a physician working at the Pisa city hospital inside the cardiology department, also affiliated with the University of Pisa. This simple example demonstrates that the generated policies and attribute sets, though random, can have a realistic interpretation.

Simplified Method for Memory-Constrained Devices

The method mentioned above allows us to evaluate the decryption performance of generic ABE schemes in the average case. However, it requires loading the device under test with a high number of policies and associated attribute sets. This is hardly feasible with memory-constrained devices like ESP32 and RE-Mote. Fortunately, from our experience, we noticed that the decryption efficiency is analytically predictable given the number of basic cryptographic operations performed (e.g., pairings, modular exponentiations, etc.) and their processing time. We thus used a simplified method that first involves a benchmark of the basic cryptographic operations, second, the random generation of many couples of access policies and attribute sets with the previously explained method, and finally, the analytical computation of the decryption performance with every such random couple.

To better understand how the decryption performance can be analytically com-

	Pairing	Mod. exp.	Mod. mul.	Point-scalar mul.	Point add.
Time (SW)	3093 ms	44 ms	1 ms	167 ms	5 ms
Time (HW)	3093 ms	44 ms	1 ms	74 ms	2 ms
Energy (SW)	140 μWh	2 μWh	0.05 μWh	10 μWh	0.2 μWh
Energy (HW)	140 μWh	2 μWh	0.05 μWh	3 μWh	0.1 μWh
GPSW decryption	n_L	$n_L + n_N - 1$	$n_L - 1$	-	-
BSW decryption	$2n_L + 1$	$n_L + n_N - 1$	$n_L + 1$	-	-

Table 6.2
Processing time of basic crypto operations on RE-Mote, and number of basic crypto operations needed in decryption by the different schemes.

puted, remind that the decryption algorithm visits the policy tree in a bottom-up order, from the leaves to the root. The algorithm visits only those nodes that are strictly necessary to visit the root. For each visited node, the algorithm executes some basic elliptic-curve operations. In both examined schemes, the number of the various basic operations are expressible in terms of the number of visited leaves (n_L) and the number of visited internal nodes (n_N). Table 6.2 shows the processing times and the energy consumption of the basic cryptographic operations involved in decryption by the two schemes, measured on a RE-Mote with or without hardware acceleration. Each value has been obtained by averaging on 100 independent replicas of the experiment. The table also shows the number of different basic operations needed in decryption by the different schemes. Since we are interested in the difference between a performance evaluation based on the worst-case and one based on the average case, we restrict our analysis to only one examined board: the RE-Mote board. Performing a similar analysis on the ESP32 board is straightforward. We omit it for the sake of brevity. Based on the numbers of Table 6.2, we applied the simplified method to benchmark the average decryption of the two schemes on RE-Mote.

Figs. 6.16 and 6.17 show respectively the resulting average decryption time and average energy consumption, compared to the worst-case ones, as they have been measured with the experiments of the previous sections (see Fig. 6.11). Each plot point relative to the average decryption time has been computed by averaging 100,000 generated couples of random policies and attribute sets. Note that only considering flat policies, like practically all the literature about ABE scheme performance evaluation does (Zickau et al., 2016; Ambrosin et al., 2016, 2015; Wang et al., 2014; Kuehner and Hartenstein, 2016), greatly overestimates the processing time of the average decryption operation. The average decryption time also shows a slight sublinear trend with respect to the number of attributes in the policy.

This is due to the presence of OR operators within the generated policies, which are instead absent in a flat policy. Indeed, to visit an OR node, the decryption algorithm just needs to visit only one of its children. The decryption algorithm will much

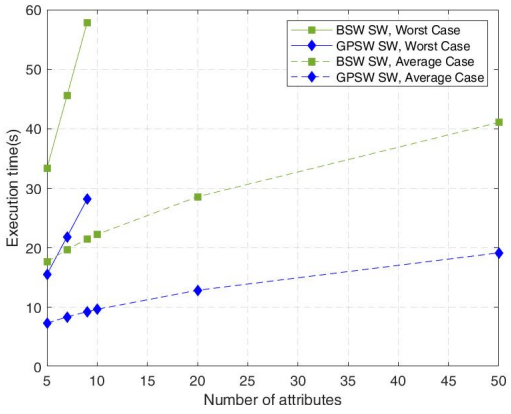


Figure 6.16. Average- and worst-case decryption time on RE-Mote.

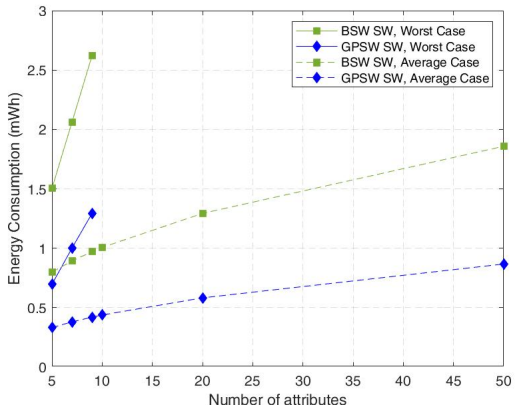


Figure 6.17. Average- and worst-case decryption energy consumption on RE-Mote.

probably visit the *most convenient* child, which is the one having fewer descendant leaves, to save time. As the attributes of the policy grow, the number of OR operators also grows, thus allowing the decryption algorithm to further save time by visiting the most convenient child each time.

We can conclude that the current literature significantly overestimates the processing time and energy by always considering the worst-case decryption.

However, such estimations are too harsh when considering the applicability of ABE in an IoT context. Indeed, realistic policies can be shaped around the needs and the capability of the devices at disposal, as we showed before. We think that our framework is better in correctly estimating the feasibility of ABE in an IoT context since it also considers the flexibility and the expressiveness of such a technique. Indeed, flexibility and expressiveness are two major features that are hard to quantify, but they significantly impact the overall system. For example, with the GPSW scheme and 9-attribute policies, the energy consumption on RE-Mote of the average-case decryption is 67% less than that of the worst-case decryption. This shows that if policies are well-thought and well-managed, ABE is far more performing than it is believed to be so far.

6.6 Answer

How Much Classical ABE Schemes Actually Impact Constrained IoT Devices?

A lot, but not as much as we thought it does. In fact, we showed that ABE schemes are feasible on IoT devices both in terms of computational capabilities and battery usage. Indeed, with careful engineering of policies and attribute sets, the battery duration of constrained devices that use ABE can last up to a month.

Chapter 7

What is the Most Suitable ABE Scheme for my System?

In this chapter, we survey ABE schemes particularly suitable for IoT applications, focusing on specific features that are desirable in an IoT context. Instead of proposing a simple list of schemes, we try to systematize the literature by identifying *performance indicators* (PIs) that are of particular interest in IoT and can be used to estimate how a given ABE scheme fits for a given IoT application. In particular, we identify six PIs, namely, the *data producer CPU efficiency*, the *data producer bandwidth efficiency*, the *key authority bandwidth efficiency*, the *data producer storage efficiency*, the *data consumer CPU efficiency*, and the *data consumer bandwidth efficiency*. These six indicators have different importance depending on different scenarios. However, we think that in the vast majority of IoT applications the first three indicators are more important (hence *KPIs* Key Performance Indicators), and the latter three are slightly less impactful (hence *APIs* Accessory Performance Indicators). Nonetheless, in this chapter, we analyze strategies that improve the performance of every PI. Therefore, the readers that need to choose an ABE scheme that fits their needs, have the tool to prioritize any of the six PIs as they please. Please note that this chapter is intended to be *selective*, in the sense that we focus only on those schemes that are promising from the point of view of one or more KPIs or APIs, and thus, are more likely to be employed in typical IoT applications. Moreover, we select only schemes provided with formal security proof. By doing this, we intentionally leave out many schemes whose potential in IoT is not sufficient or whose security is not proved. This chapter enables researchers and practitioners to get up to speed quickly on the characteristics of the different ABE schemes present in the literature and understand their suitability for the particular IoT applications, obviating the need to read many cryptographic papers. As a further contribution, we employ thorough simulations to assess the efficiency of a subset of representative schemes. Based on such simulations, we observe that no scheme excels in all three performance indicators at once,

but some simultaneously perform well in two indicators.

In the past, other surveys on ABE (Oberko et al., 2021; Al-Dahhan et al., 2019; Edemacu et al., 2019; Lee et al., 2013; Liu et al., 2016; Moffat et al., 2017; Pang et al., 2014; Qiao et al., 2014; Zhang et al., 2020) have been published, but they differ from ours for various reasons. Some of the existing surveys (Lee et al., 2013; Pang et al., 2014; Qiao et al., 2014) are outdated; As ABE is a trending topic, a lot of new and more sophisticated schemes are proposed every year. Some surveys do not carry out exhaustive research on ABE, limiting their contribution to analyzing schemes with specific characteristics. For example, Liu et al. (Liu et al., 2016) and Al-Dahhan et al. (Al-Dahhan et al., 2019) survey only revocable ABE schemes, while Moffat et al. (Moffat et al., 2017) do not consider revocable schemes. Recent surveys (Moffat et al., 2017; Zhang et al., 2014a; Edemacu et al., 2019) do not explore novel strategies, e.g., ABE schemes with asymmetric pairings, and they do not quantitatively estimate the key performance indicators of the suitable schemes through experiments. In the present chapter, we do not survey lattice-based ABE schemes, for example, (Li et al., 2019; Dai et al., 2018). Indeed, although they are interesting from a post-quantum perspective, they seem too burdensome for the class of constrained IoT devices that we consider in this chapter.

The rest of the chapter is organized as follows. Section 7.1 introduces the typical IoT architecture on which it is possible and fruitful to apply ABE techniques, introduces the key performance indicators, and advocates why they deserve priority. Sections 7.2, 7.3, and 7.4 survey the different strategies employed in the literature to improve the KPIs. In particular, Section 7.2 focuses on data producer CPU efficiency, Section 7.3 on key authority bandwidth efficiency, and Section 7.4 on data producer bandwidth efficiency. In Section 7.5, we simulate some promising ABE schemes that excel in at least one KPI, and we discuss the results. Section 7.6 surveys the different strategies employed in the literature to improve the APIs. Finally, Section 7.7 ends the chapter, answering the question inquired.

7.1 ABE in IoT

The typical IoT architecture (Fig. 7.1) includes many *data producers* (or simply *producers*), which produce information, many *data consumers* (or simply *consumers*), which consume such information, and some *data storage*, on which information is either temporarily or permanently stored.

Data producers are typically sensing devices that measure physical quantities or detect events from the environment. They are often constrained and battery-powered and have limited computing and connectivity capacities. In some cases, data producers are more resourceful devices, for example, single-board computers (e.g., Raspberry Pi) or mobile devices. Data consumers are typically devices that dis-

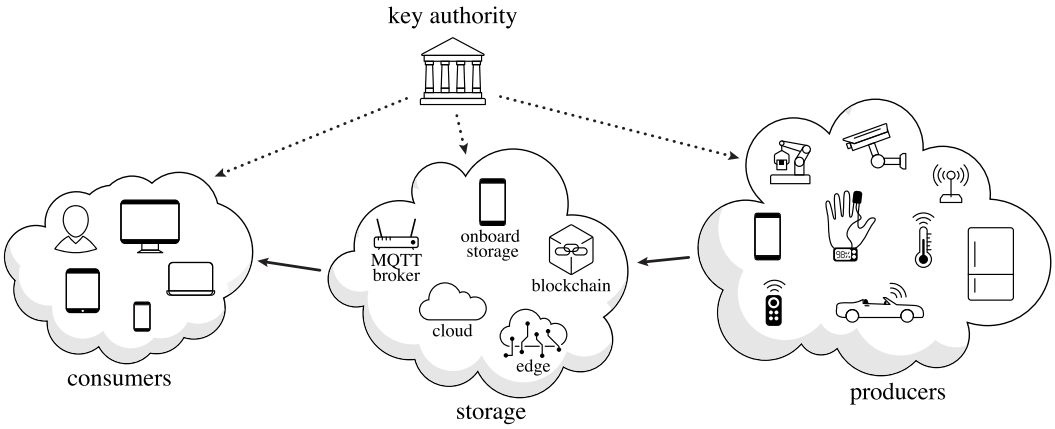


Figure 7.1. ABE architecture.

play information to users or actuators who automatically undertake actions based on such information. They can be smartphones, smartwatches, tablets, or even full-fledged computers with higher computing and connectivity capabilities than the average data producer. In some cases, data consumers are more constrained devices, for example, single-board computers or even battery-powered actuators. Data storage can be implemented in various ways, including at one extreme high-end mainframes providing cloud services to subscribers, and at the other extreme the data producers themselves in case they temporarily store sensed data locally instead of transmitting it immediately (*onboard storage*). Between these two extremes, data storage can be edge nodes, MQTT broker devices, etc. In addition, many recent IoT systems store data in blockchain data structures, like Ethereum (Arena et al., 2019). The typical ABE scheme in the literature considers all data storage as *untrusted* for several reasons. Indeed, cloud servers are often managed by third-party companies based in foreign countries. Furthermore, edge servers and cloud servers are Internet-connected and constantly exposed to cyberattacks, both software and hardware (Lipp et al., 2018; Kocher et al., 2020). Onboard storage is considered untrusted because producers are often easy to hack or physically accessible and unattended, e.g., in wireless sensor networks. Finally, data is inherently public in the case of on-blockchain storage, so it must be encrypted to preserve its secrecy (Arena et al., 2019).

For all the reasons set out above, it is essential to encrypt the data when at rest in the data storage. ABE technology is very effective for all those systems required to preserve data confidentiality and enforce an access control mechanism. Some ABE schemes in the literature consider data storage as *semi-trusted*, which may assume different meanings but generally implies that data storage can perform *some* malicious actions (e.g., attempting to decrypt data), but not others (e.g., actively sending malicious messages). The *honest-but-curious* trust model used in some papers (Yu et al., 2010a; Rasori et al., 2020) falls into this category. Note that we do not consider

data storage those nodes that are transparent from the point of view of data connections, for example, network gateways or Internet routers. Of course, these devices are untrusted, but they can be prevented from accessing data by simply establishing end-to-end secure channels, for example, with the DTLS protocol (Rescorla and Modadugu, 2012).

If ABE is employed, a fourth entity is necessary in the architecture: the *key authority*, which is trusted by all the other entities. The key authority has the responsibility of creating, distributing, updating, and revoking ABE keys. We refer to all these activities with the general term *key management*. The key authority is typically a PC-class device that is not constantly connected to the Internet but rather goes online only when key management procedures must be fulfilled.

Performance Indicators

We identify three KPIs that an ABE scheme must offer to be best suited for most common IoT applications (see Fig. 7.2). These three primary properties are (i) producer CPU efficiency, (ii) producer bandwidth efficiency, and (iii) key authority bandwidth efficiency. We further identify three accessory performance indicators, which are desirable only if they do not jeopardize one or more KPIs. In the typical case, accessory performance indicators are (i) producer storage efficiency, (ii) consumer CPU efficiency, and (iii) consumer bandwidth efficiency. Of course, every

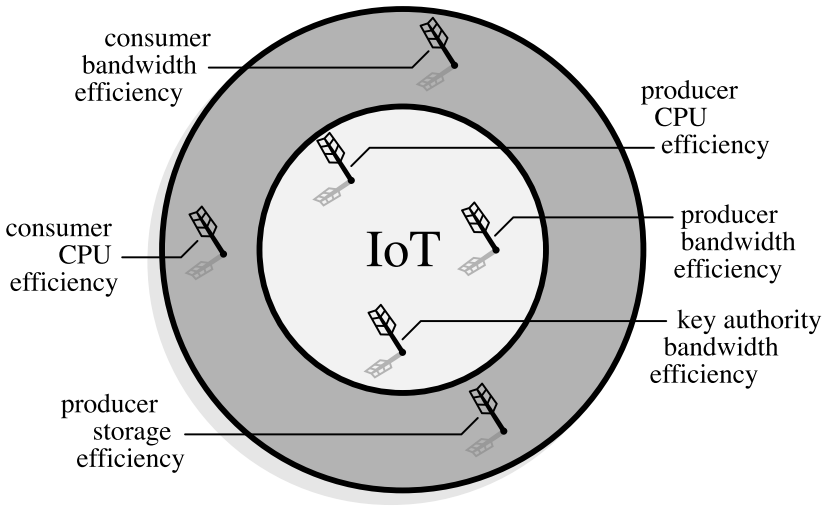


Figure 7.2. Performance indicators.

choice about a set of key and accessory performance indicators is debatable, and one should analyze the specific IoT application to identify key indicators case by case. Nevertheless, we feel that our choices fit the *typical* IoT application, and they are also reasonable for many specific, less typical ones. In the following, we will explain our rationale behind such a claim.

The producer CPU efficiency is a key indicator for the typical IoT application because constrained devices with limited processing capabilities usually produce data. Moreover, such devices are often battery-powered, so they must employ low-power communication protocols, which have low bit rates. This also makes the producer bandwidth efficiency a key indicator. On the other hand, the CPU and bandwidth efficiency of the consumers are not as important as those of the producers because data is usually consumed by users equipped with resourceful devices. Of course, in those specific applications in which consumers are small battery-powered actuators, their efficiency may rise in importance. However, considering the typical case, we decided to categorize the consumer CPU and bandwidth efficiency as accessory performance indicators. Notably, we consider the producer storage efficiency only as an accessory indicator because many ABE schemes take up very little storage on the producer, i.e., a few hundreds of bytes. For those schemes in which the producer storage load is more pronounced, there are techniques to sensibly alleviate it (see Section 7.6). Note that the producer storage capacity may become the real bottleneck of the overall system efficiency if onboard storage is the only storage media used since possibly a large amount of encrypted data is stored locally. In this specific case, the producer storage efficiency may become a key performance indicator. Regarding the key authority, we believe that its CPU efficiency is not a key indicator since the key authority is typically a PC-class device with high computational capabilities. The only scenario in which the key authority CPU load might be relevant is that of a scheme using the naive revocation technique, described in Section 2.2. Rather, we consider the key authority bandwidth efficiency a key indicator mainly for scalability issues. Indeed, in many ABE schemes, the key authority traffic grows with the number of consumers in the system, which tends to be quite large in IoT applications, i.e., hundreds to thousands of devices. Finally, in the specific case of blockchain data storage, it is worth noting that the storage efficiency may rise in importance as a performance indicator. Indeed, the typical blockchain, e.g., Ethereum, makes users pay each kilobyte of storage space in cryptocurrency. However, the blockchain storage efficiency can be enhanced by improving both the key authority bandwidth (i.e., smaller key update material) and the producer bandwidth (i.e., smaller ciphertexts). Therefore, we decided to neglect such a performance indicator because the strategies to improve it coincide with the strategies adopted to improve two other key indicators, explained respectively in Sections 7.3 and 7.4.

Cited Schemes and Naming Convention

From now on, we will refer to the concrete ABE schemes proposed by the literature with the initials of the authors' surnames, the publication year, and an optional final number discriminating different schemes within the same paper. For example, "SW05-2" refers to the second scheme presented by Sahai and Waters in their 2005

paper (Sahai and Waters, 2005), "BSW07" refers to the (unique) scheme presented by Bethencourt, Sahai, and Waters in their 2007 paper (Bethencourt et al., 2007), and so on. Table 7.1 contains the names of every ABE scheme cited in the present chapter, with its precise reference and its basic characteristics.

7.2 Producer CPU Efficiency

In this section, we focus on the CPU efficiency of ABE schemes from the point of view of the data producers. It is worth noting that the encryption operations dominate the CPU load required by a scheme on the producers. Indeed, the producers are also involved in key management processes, but these typically do not require producers to perform any computation. Instead, in this case, producers are required to download new public parameters or lists of revoked consumers. In the typical ABE scheme, encryption requires point-scalar multiplications and, for large-universe schemes, hash functions whose output is in an elliptic-curve group. Hence, a preliminary observation regarding encryption efficiency is that large-universe schemes are typically less performant than small-universe ones. This is because hash functions over elliptic-curve groups are quite burdensome for resource-constrained devices (see Table 7.3 of Section 7.5). The first straightforward method to save producer CPU is thus to use small-universe schemes. Of course, the additional cost of large-universe schemes can be alleviated by precomputing the hashes of the attributes used in encryption and avoiding encrypting with attributes other than the precomputed ones. However, this partially nullifies the advantage of being large-universe since the producer can encrypt only with a predefined set of attributes, just like it happens for small-universe schemes. In many IoT applications, it is not possible to fix a static set of attributes for each producer to encrypt with, so other more flexible strategies to improve CPU efficiency are needed.

Another simple technique, featured in LPD21, is to lower the producer CPU load is to use the *digital envelope*, that is, to let producers encrypt a symmetric key with ABE with a specific attribute set (KP-ABE) or a specific policy (CP-ABE). Then, all the plaintexts that must be protected by such an attribute set/policy are efficiently encrypted with such a symmetric key. This technique can be applied to every ABE scheme, and it is particularly advantageous in case producers must encrypt many times using the same attribute set/policy. Nonetheless, it makes the key management more complex because symmetric keys must be stored by producers and consumers, and possibly revoked by the key authority.

Besides these basic techniques, we identified three main strategies to improve the producer CPU efficiency: (i) encryption outsourcing (Touati et al., 2014; Touati and Challal, 2016; Hohenberger and Waters, 2014); (ii) adopting alternative mathematics (Yao et al., 2015; Odelu and Das, 2016; Odelu et al., 2017); and (iii) adopting

Table 7.1
Cited ABE Schemes

Scheme Name	Reference	Paradigm	Universe	Access Structure Expressiveness
SW05-1	Sahai and Waters (2005), Sec. 4.1	KP-ABE	small	k-of-n
SW05-2	Sahai and Waters (2005), Sec. 6.1	KP-ABE	large	k-of-n
GPSW06-1	Goyal et al. (2006b), Sec. 4.2	KP-ABE	small	full monotonic
GPSW06-2	Goyal et al. (2006b), Sec. 5.1	KP-ABE	large	full monotonic
GPSW06-3	Goyal et al. (2006b), App. A.1	KP-ABE	small	full monotonic
BSW07	Bethencourt et al. (2007), Sec. 4.2	CP-ABE	large	full monotonic
OSW07	Ostrovsky et al. (2007), Sec. 3.1	KP-ABE	large	full non-monotonic
BGK08-4	Boldyreva et al. (2008), Sec. 6	KP-ABE	small	full monotonic
AI09	Attrapadung and Imai (2009), Sec. 4	KP-ABE	large	full monotonic
EMNOS09	Emura et al. (2009), Sec. 4	CP-ABE	small	AND_{m}
HLR10	Herranz et al. (2010), Sec. 3	CP-ABE	small	k-of-n
LOSTW10-1	Lewko et al. (2010), Sec. 2.3	CP-ABE	small	full monotonic
YWRL10	Yu et al. (2010a), Sec. IV	KP-ABE	small	full monotonic
ZH10	Zhou and Huang (2010), Sec. 3	CP-ABE	small	$\text{AND}^{*}_{\pm}$
ALP11	Attrapadung et al. (2011), Sec. 5	KP-ABE	large	$\text{AND}_{\pm}$
HN11	Hur and Noh (2011), Sec. 4	CP-ABE	large	full monotonic
W11-1	Waters (2011), Sec. 3	CP-ABE	small	full monotonic
W11-2	Waters (2011), Sec. 5	CP-ABE	small	full monotonic
W11-3	Waters (2011), Sec. 6	CP-ABE	small	full monotonic
W11-4	Waters (2011), App. A	CP-ABE	large	full monotonic
W11-5	Waters (2011), App. B	CP-ABE	large	full monotonic
GZCMZ12-1	Ge et al. (2012), Sec. 3.1	CP-ABE	small	k-of-n
GZCMZ12-2	Ge et al. (2012), Sec. 3.2	CP-ABE	small	k-of-n
LGRDY12	Li et al. (2012), Sec 3.2	CP-ABE	small	AND_{m}
SSW12-1	Sahai et al. (2012), Sec. 8	KP-ABE	small	full monotonic
SSW12-2	Sahai et al. (2012), Sec. 10	CP-ABE	small	full monotonic
RW13-1	Rouselakis and Waters (2013), Sec. 4	CP-ABE	large	full monotonic
RW13-2	Rouselakis and Waters (2013), App. C	KP-ABE	large	full monotonic
ZHW13	Zhou et al. (2013), Sec. 4	CP-ABE	small	$\text{AND}^{*}_{\text{m}}$
DJ14	Doshi and Jinwala (2014), Sec. 4	CP-ABE	small	AND_{m}
HW14-1	Hohenberger and Waters (2014), Sec. 3	KP-ABE	large	full monotonic
HW14-2	Hohenberger and Waters (2014), Sec. 4	CP-ABE	large	full monotonic
PYS14	Phuong et al. (2014), Sec. 3	CP-ABE	small	$\text{AND}^{*}_{\pm}$
TCB14	Touati et al. (2014), Sec. IV	CP-ABE	large	full monotonic
ZCLLL14	Zhang et al. (2014a), Sec. 5.3	CP-ABE	small	$\text{AND}_{\pm}$
ZZCLL14	Zhang et al. (2014b), Sec. 4	CP-ABE	small	$\text{AND}^{*}_{\text{m}}$
CGW15-1	Chen et al. (2015), App. B.1	KP-ABE	small	full monotonic
CGW15-2	Chen et al. (2015), App. B.2	CP-ABE	small	full monotonic
PYSC15-1	Phuong et al. (2015), Sec. 3	KP-ABE	small	$\text{AND}^{*}_{\pm}$
PYSC15-2	Phuong et al. (2015), Sec. 4	CP-ABE	small	$\text{AND}^{*}_{\pm}$
YCT15	Yao et al. (2015), Sec. 4.2	KP-ABE	small	full monotonic
CDLQ16	Cui et al. (2016), Sec. 4	CP-ABE	large	full monotonic
AHMTY16	Attrapadung et al. (2016), Sec 3	KP-ABE	large	full monotonic
OD16	Odelu and Das (2016), Sec 3	CP-ABE	small	$\text{AND}_{\pm}$
TC16	Touati and Challal (2016), Sec. IV	KP-ABE	small	full monotonic
AC17-1	Agrawal and Chase (2017), Sec. 3	CP-ABE	large	full monotonic
AC17-2	Agrawal and Chase (2017), App. B	KP-ABE	large	full monotonic
AC17-3	Agrawal and Chase (2017), App. D	CP-ABE	large	full monotonic
AC17-4	Agrawal and Chase (2017), App. E	CP-ABE	small	full monotonic
AC17-5	Agrawal and Chase (2017), App. F	KP-ABE	small	full monotonic
LYHZS17	Li et al. (2017), Sec. 7	CP-ABE	large	full monotonic
ODKCJ17	Odelu et al. (2017), Sec. IV	CP-ABE	small	$\text{AND}_{\pm}$
QZZC17	Qin et al. (2017), Sec. 3	CP-ABE	large	full monotonic
HWY18	Huang et al. (2018), Sec. 5	CP-ABE	large	full monotonic
JSMG18-1	Jiang et al. (2018), Sec. 4	CP-ABE	small	$\text{AND}_{\pm}$
JSMG18-2	Jiang et al. (2018), Sec. 5	CP-ABE	small	$\text{AND}_{\pm}$
LYZL18	Liu et al. (2018), Sec. 4	CP-ABE	small	full monotonic
XYM19	Xu et al. (2019), Sec. 5	CP-ABE	large	full monotonic
LPD21	La Manna et al. (2021), Sec. 5	CP-ABE	large	full monotonic
RPDY21	Rasori et al. (2021), Sec. 4	KP-ABE	small	full monotonic

Type III pairings (Chen et al., 2015; Agrawal and Chase, 2017).

Encryption Outsourcing

The schemes adopting this strategy reduce the producer CPU load sensibly, but they need specific architectural or usage features, for example, the presence of full-resource neighbors or the presence of users that periodically load producers with pre-computed quantities. Touati et al. (Touati et al., 2014) propose a CP-ABE scheme[1] (TCB14) that collaboratively accomplishes the encryption of a message. To do that, the data producer needs to establish secure channels with at least two trusted full-resource neighbors to which delegate burdensome operations. The neighbors compute partial results and send them to the producer, which combines them, creating the final ciphertext. The authors of (Touati and Challal, 2016) propose a similar KP-ABE scheme (TC16). The offloading technique of TCB14 and TC16 greatly unburdens producers, but it needs multiple resourceful devices in the neighborhood, which could be missing. Moreover, the outsourcing system heavily impacts bandwidth so that producers may spend more time and energy communicating than what they save in processing.

Another efficient solution is proposed by Hohenberger and Waters in (Hohenberger and Waters, 2014). The authors propose a KP-ABE scheme (HW14-1) and a CP-ABE one (HW14-2), based on the RW13-2 and RW13-1 schemes, respectively. Both the schemes split the encryption algorithm into two phases. In the first phase (offline phase), all the burdensome operations are pre-processed, whereas, in the second phase (online phase), light operations are performed to generate the actual ciphertext. This solution is helpful if data producers are mobile devices that experience battery charging cycles, e.g., smartphones. The offline phase is executed while the device is charging, and when the data is ready –and the device is possibly not charging– the online phase is executed. Note that HW14-1 and HW14-2 do not improve CPU efficiency strictly speaking, because they do not outsource encryption, but rather rationalize the CPU usage cycles. However, it is worth noting that such schemes can be seamlessly adapted to outsource encryption. Indeed, the offline phase can be outsourced to some trusted resourceful device, and the resulting pre-processed quantities can be loaded on the producers through some secure channel. We will explore this possibility in the experimental section (Section 7.5). Note that HW14-1 and HW14-2 allow the producer to decide the message and the attributes used in encryption in the online phase when the full-resource device is offline. TCB14 and TC16 do not provide this feature.

[1]Despite (Touati et al., 2014) and (Touati and Challal, 2016) do not provide formal security proofs, we include them anyway in the present chapter because similar security proofs of the base schemes BSW07 (Bethencourt et al., 2007) and GPSW06-1 (Goyal et al., 2006b) should apply under the assumption of secure channels between the producer and the full-resource neighbors.

To sum up, encryption outsourcing does not have general applicability in IoT. Outsourcing is possible only if there are trusted full-resource devices close to the producer. This happens, for example, in the case of a network of producers administered by a unique entity, in which one or more full-resource gateways are present. However, these full-resource nodes would become a single point of trust of the whole system, and their compromise might have a devastating effect on data confidentiality.

Alternative Mathematics

Some studies propose ABE schemes that do not employ pairing operations (*pairing-free schemes*). These schemes employ different cryptographic mathematics, usually ECC (Yao et al., 2015; Odelu and Das, 2016) or RSA (Odelu et al., 2017). Pairing-free schemes allow for the fastest encryption in the literature. Indeed, RSA-based schemes employ extremely simple modular mathematics, which can also be hardware accelerated in modern IoT devices (Zolertia S.L., 2017). ECC-based schemes can employ small and standard elliptic curves, for example, the P-192 one for 96-bit security or the P-256 one for 128-bit security (Kerry, 2013). These curves do not support an efficiently-computable pairing operation, but they are generally more efficient than pairing-friendly curves with the same level of security. This is because they can be represented on the shortest possible number of bits, for example, 160 bits for obtaining 80-bit security. Also, operations on the standard curves are hardware-accelerated in modern IoT devices (Zolertia S.L., 2017). Prominent pairing-free ABE schemes in the literature are YCT15 and OD16, which employ ECC mathematics, and ODKCJ17, which employs RSA mathematics.

Unfortunately, the security of such ABE schemes is debated in the cryptography community. Some of them, i.e., YCT15, OD16, and ODKCJ17, have been successfully cryptanalized by successive papers: respectively (Tan et al., 2019) and (Herranz, 2020) for YCT15, and (Herranz, 2017) for OD16 and ODKCJ17. A recent paper (Herranz, 2020) cryptanalized several other pairing-free ABE schemes. Herranz (Herranz, 2017) provided a simple argument motivating the reason why a secure RSA/ECC ABE scheme should not exist. Indeed, since attribute-based encryption is a generalization of identity-based encryption, if one could design a secure RSA/ECC ABE scheme, this could be easily converted into a secure RSA/ECC IBE scheme. However, designing such an IBE scheme has shown to be an extremely hard problem. In practice, such a problem has been unsolved since 1984, when the IBE problem was firstly stated by Shamir (Shamir, 1985). This argument raises doubts about the security of all the RSA/ECC ABE schemes published until now, although they are usually accompanied by formal security proofs. We chose not to neglect RSA/ECC schemes in this chapter so that the reader is aware of the problems that

come with this strategy. However, we do not endorse such schemes and therefore we will not simulate them in Section 7.5.

Type III Pairings

The majority of ABE schemes have been designed, proved for security, and benchmarked with Type I pairings. This is probably for historical reasons since the first pairing-based cryptographic schemes were designed with this type of pairings (Joux, 2000; Boneh and Franklin, 2001). However, using Type III pairings allows us to speed up some cryptographic operations. This is because Type III pairings permit smaller representations of $\mathbb{G}_1$ elements with the same security level (Agrawal and Chase, 2017), thus leading to faster operations on them. Fortunately, many existing ABE schemes, including the classic SW05-1, GPSW06-1, and BSW07, are easily "portable" to Type III pairings. By doing so, their security proofs are invalidated, but there are formal methods to convert a security proof with Type I pairings to an equivalent one with Type III pairings (Akinyele et al., 2015). Notably, there is no unique way to convert a scheme from Type I to Type III pairings. Broadly speaking, this is because each Type I pairing $e(A, B)$ (with $A, B \in \mathbb{G}$) employed in the scheme can be converted in two different ways: (i) assuming $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$ and thus leaving the pairing as is, or vice versa (ii) assuming $A \in \mathbb{G}_2$ and $B \in \mathbb{G}_1$ and thus inverting the pairing to be $e(B, A)$. These choices lead to different performance in different operations. Typically, the most convenient choice for the producer efficiency is the one that converts the highest number of $\mathbb{G}$ elements to $\mathbb{G}_1$ elements in the ciphertext. In this way, the encryption performs point-scalar multiplications in $\mathbb{G}_1$, which are the efficient ones. Type III pairings also enjoy much more efficient $\mathbb{G}_1$ hash operations than Type I ones, thus they are convenient also to reduce the cost of large-universe schemes. On the negative side, using Type III pairings decreases the efficiency of pairing operations and point-scalar multiplications in $\mathbb{G}_2$, which are typically used in decryption and key generation, respectively. Thus, in those IoT applications in which the consumer CPU efficiency and/or the key authority CPU efficiency is more important than the producer's one, adopting Type III pairings could not be a convenient solution.

Some recent studies (Chen et al., 2015; Agrawal and Chase, 2017) propose ABE schemes that have been designed explicitly for Type III pairings to improve encryption efficiency. Chen et al. (Chen et al., 2015) proposed a framework for building ABE schemes, and they applied such a framework to propose two concrete schemes: CGW15-1 (KP-ABE) and CGW15-2 (CP-ABE). The authors used Type III pairings for their schemes to improve the encryption performance. Agrawal and Chase (Agrawal and Chase, 2017) proposed AC17-1 (CP-ABE, named "FAME" by the authors) and AC17-2 (KP-ABE), both employing Type III pairings. Such schemes are inspired by Chen et al.'s ones, but they provide for large universes. In the same paper, the authors also provide other schemes, i.e., AC17-3, AC17-4, and

AC17-5, Type III conversions of BSW07, W11-1, and GPSW06-3, respectively. Not all the schemes presented by Agrawal and Chase are optimized for encryption. Among these schemes, the fastest ones in encryption are AC17-5 (for the KP-ABE paradigm) and AC17-1 (for the CP-ABE one).

7.3 Key Authority Bandwidth Efficiency

Key authority bandwidth efficiency depends entirely on key management operations. A system deployed to run over the long haul must foresee that consumers' roles and privileges can change over time, consumers can join or leave the system, and consumers' keys can get compromised, either because stolen by an attacker or lost. In response to these events, the key authority should distribute new keys or revoke old ones. While key distribution for joining consumers is typically a trivial task, key revocation is more complex. In the literature, key revocation is classified into three categories: *direct*, *indirect*, and *attribute-wise*. In direct revocation, consumers' decryption keys are associated with *identifiers*, and revoking a key means disabling the consumer identifier from decryption of new ciphertexts. The list of revoked identifiers, usually referred to as *revocation list*, must be available to all the producers, which use it during encryption in such a way to exclude revoked keys from being capable of decrypting the new ciphertexts. Differently, in indirect revocation, the revocation process involves the non-revoked consumers. In particular, their decryption keys are updated in order to decrypt new ciphertexts, while revoked ones are not. Usually, producers do not participate in the revocation process, but some schemes, e.g., LPD21, require producers to update a small part of the public parameters and use them for new encryptions. The naive revocation technique presented in Section 2.2 falls in this category, but its performances are poor since the key authority generates and distributes new decryption keys to all the non-revoked consumers. Also, all the producers must obtain the new public parameters. Attribute-wise revocation can be seen as a type of indirect revocation at attribute level. To revoke a compromised key, the key authority issues a new version of the attributes present in that key. Every decryption key, except for the revoked one, is updated to the new version. Producers need to obtain the new version of the public parameters to generate new ciphertexts that the revoked key cannot decrypt. In the following, we describe existing revocation strategies and analyze some approaches proposed in the literature that attempt to limit the key authority effort to handle key revocations.

We identified three main strategies to reduce the key authority traffic: (i) adopting direct revocation (Liu et al., 2018; Phuong et al., 2015), which completely unburdens the key authority of the key revocation tasks; (ii) adopting binary tree structures within indirect revocation (Boldyreva et al., 2008; Sahai et al., 2012; Xu et al.,

2019; Cui et al., 2016; Qin et al., 2017; Attrapadung and Imai, 2009), which reduce the key authority traffic from linear to logarithmic in the number of consumers; (iii) adopting attribute-wise revocation (Yu et al., 2010a; Hur and Noh, 2011; Li et al., 2017), which makes the traffic generated by revocation tasks dependent on the number of revoked attributes instead of the number of consumers.

Direct Revocation

Direct revocation is the most effective strategy to reduce key authority traffic. When a decryption key is compromised, the key authority simply adds the identifier associated with that key to a revocation list. During encryption, producers use the revocation list as additional input to generate new ciphertexts that decryption keys associated with identifiers in the revocation list cannot decrypt. However, as already highlighted in Section 7.1, an IoT ABE scheme should weigh as few as possible on the resource-constrained producers. Direct revocation often fails in this because producers' bandwidth and encryption efficiency are inevitably reduced. Indeed, producers need to obtain an updated copy of the revocation list prior to encryption (bandwidth overhead), use the revocation list during encryption (encryption overhead), and then upload a larger ciphertext on the data storage (bandwidth overhead). As a consequence of these clear disadvantages, some studies proposed ABE schemes with direct revocation that enlighten the burden on the producers. For example, the authors of (Liu et al., 2018) proposed a direct revocable CP-ABE scheme with a short revocation list (LYZL18). To achieve this, the revocation list is condensed into a single $\mathbb{G}$ element in the ciphertext. In this scheme, decryption keys have a planned expiration date, and revoked keys –whose expiration date is over– are excluded from the revocation list to relieve encryption efficiency.

Phuong et al. (Phuong et al., 2015) proposed a direct revocable ABE scheme for both KP-ABE (PYSC15-1) and CP-ABE (PYSC15-2) paradigms. They combine ABE with broadcast encryption to impede decryption to revoked identifiers. The encryption algorithm takes as input the list of non-revoked identifiers. This is a drawback because producers must update their list when a consumer is revoked and every time a new consumer joins the system. As in the LYZL18 scheme, the list used during encryption is condensed into a single $\mathbb{G}$ element. Overall, the schemes reach good efficiency in terms of bandwidth and storage because the KP-ABE variant has short ciphertexts and constant-size decryption keys, and the CP-ABE variant has constant-size ciphertexts and short decryption keys.

In short, direct revocation is the strategy that weighs less on the key authority but inevitably hampers the other key performance indicators as it adds computational and communication overhead on the producers.

Binary Trees

Indirect revocation typically leverages an additional input to revoke consumers: time. Indirect revocation schemes are organized in time periods, and at the beginning of a new period, the key authority updates only non-revoked consumers' decryption keys. In a naive approach (Boneh and Franklin, 2001), the key authority generates a new key for each consumer at each new period and individually sends them to consumers through secure channels. Obviously, this solution does not scale well with regard to the key authority since its computational and communication costs increase linearly with the number of consumers.

Boldyreva et al. (Boldyreva et al., 2008) improved the efficiency of the keys update mechanism. In their scheme (BGK08-4), the costs for the key authority are reduced from linear to logarithmic in the number of consumers. To achieve this performance, they first proposed to use a binary tree for creating key-update material. Notably, this information, which they call *key update*, is not a secret. The key update can be published on the data storage to eliminate the need for interaction between consumers and the key authority. In this scheme, decryption keys are associated with identifiers. A consumer owns a long-term secret key linked to its identifier and a short-term decryption key valid for the current time period only. The consumer creates a new short-term decryption key at each time period by combining the key update with its long-term secret key. Only non-revoked consumers are capable of performing this operation. Indeed, the key authority generates key updates that do not allow revoked consumers to create a new valid short-term decryption key. The size of a key update is $O(R \log(N/R))$ elements in $\mathbb{G}$, where N is the total number of consumers, and $R \leq N$ is the number of revoked consumers.

Inspired by Boldyreva et al.'s work (Boldyreva et al., 2008), many indirect revocable ABE schemes using binary tree construction have been proposed (Sahai et al., 2012; Xu et al., 2019; Cui et al., 2016; Qin et al., 2017). Sahai et al. (Sahai et al., 2012) extended the concept of revocation to a broader sense and proposed an indirect revocable ABE scheme for both KP-ABE (SSW12-1) and CP-ABE (SSW12-2) paradigms. They dealt with the problem of revoking access also to previously encrypted data. In their construction, based on LOSTW10-1, the untrusted storage is enabled to re-encrypt old ciphertexts to a more restrictive policy using only public information. More precisely, a ciphertext encrypted at time t is transformed to an independent encryption of the same message under the same attribute set at time $t + 1$. Note that re-encryption is performed without accessing the message. Xu et al. (Xu et al., 2019) proposed a revocable ABE scheme (XYM19) that adds a feature to the one introduced by Sahai et al. (Sahai et al., 2012). In their construction, based on RW13-1, the authors deal with the *decryption key exposure* attack[2], which was first introduced

[2]A scheme has decryption key exposure resistance if the compromise of the short-term decryption key does not imply the compromise of the long-term secret key.

by Seo and Emura (Seo and Emura, 2013). The authors show that the performance of their scheme is very similar to that of SSW12-1.

Cui et al. (Cui et al., 2016) (CDLQ16) and Qin et al. (Qin et al., 2017) (QZZC17) also extended the techniques of Boldyreva et al. (Boldyreva et al., 2008) in the CP-ABE realm. In their construction, based on RW13-1, the untrusted storage (which does not hold any secret information) carries out the majority of decryption and revocation workload. As in (Boldyreva et al., 2008), at the beginning of a new time slot t, the key authority loads the key update on the data storage. For a consumer with identifier id, the data storage computes a *transformation* key for the consumer id and the time slot t. On a data request by a consumer, the data storage uses the transformation key to manipulate the requested ciphertext. The consumer finalizes the decryption at a low and constant cost. The difference between these schemes is that the latter has decryption key exposure resistance, which adds a moderate cost for the consumer.

In short, the binary tree is an efficient construction to achieve indirect revocation as it limits the key authority bandwidth to be logarithmic in the number of consumers.

A Hybrid Scheme

Attrapadung and Imai (Attrapadung and Imai, 2009) proposed a hybrid revocable KP-ABE scheme (AI09) that allows for both direct and indirect revocation modes. The indirect revocation technique is pretty much the one proposed in Boldyreva et al.'s work (Boldyreva et al., 2008) that we previously described. That is, the key authority publishes a key update of size $O(R \log(N/R))$ at each time period through which non-revoked consumers can update their keys. In the direct revocation, producers use the elements in the key update as additional input during encryption. If direct revocation is used to create a ciphertext at time slot t, a non-revoked consumer is not required to update its key for that time period. If indirect revocation is used to create a ciphertext at time slot t, a non-revoked consumer is required to update its key for that time period. Depending on its resources, a producer can use and switch between the direct and indirect revocation modes. However, if producers mix direct and indirect modes within the same time period, the scheme takes the worst of both worlds because it requires additional effort to handle revocation for both consumers and producers. Moreover, the key authority bandwidth always results as that of BGK08-4.

Attribute-Wise Revocation

Attribute-wise revocation is more fine-grained than the previous strategies because it can revoke the privileges of a consumer at attribute level. This means that the key

authority can invalidate just one attribute in a consumer's decryption key. With this strategy, the key authority bandwidth depends on the number of revoked attributes.

Yu et al. (Yu et al., 2010a) proposed an attribute-wise revocable KP-ABE scheme (YWRL10) based on GPSW06-1. We recall that GPSW06-1 has a small universe, and the public parameters contain one *component* (a $\mathbb{G}$ element) for each attribute in the universe. In the YWRL10 construction, all the attributes of the universe (except for one, called *dummy attribute*) are subject to versioning. When the key authority wants to revoke a set of attributes λ embedded in a decryption key, it generates a new version of the public parameters components for the attributes in λ. Then, it computes a secret quantity, i.e., an element in $\mathbb{Z}_p$ called *re-encryption key*, for each attribute in λ. Finally, it loads the updated public parameters' components and the re-encryption keys on the semi-trusted storage, thus transferring $|\lambda| \cdot (|\mathbb{G}| + |\mathbb{Z}_p|)$ bits, where the symbol $|\cdot|$ denotes the size of an element expressed in bits. Note that if the key authority wants to revoke a whole compromised key, it can revoke only the minimal set of attributes without which the embedded access policy can never be satisfied. For example, if the compromised key's access policy is A AND (B OR C), the minimal set is {A}. Depending on the shape of the access policy, this enhancement can save a lot of bandwidth overhead. The data storage is in charge of updating decryption keys of non-revoked consumers which shared at least one attribute with the revoked attribute set λ. The YWRL10 assumes the data storage as honest-but-curious. Rasori et al. (Rasori et al., 2021) proposed a scheme (RPDY21) based on the YWRL10 one, which is secure even with untrusted data storage.

Hur and Noh (Hur and Noh, 2011) proposed an attribute-wise revocable CP-ABE scheme (HN11) based on BSW07. As in YWRL10, the key authority can revoke a set of attributes of a decryption key associated with a consumer identifier. Producers are not affected by revocation, and they generate CP-ABE ciphertexts by always using the same public parameters. The key authority only communicates to the semi-trusted storage the consumer identifier and the change of access privileges it wants to actuate. For example, if the key authority wants to revoke the whole key of consumer id, it communicates all the attributes in that key and the associated consumer identifier. The data storage manipulates each ciphertext and re-encrypts it at attribute level so that only non-revoked consumers (for that attribute) can use that attribute during decryption. On the contrary, if the consumer id is revoked for the attribute i, it cannot use that attribute anymore for decrypting. This scheme applies two layers of encryption. The first layer is CP-ABE encryption, performed by producers, and the second layer is symmetric encryption. The second layer is applied by the data storage and enforces revocation at attribute level. To decrypt a ciphertext, a consumer first proves that it has access privileges for an attribute by performing symmetric decryption. Then, it uses CP-ABE decryption to retrieve the message. This scheme, too, makes use of the binary tree to manage revocation

through symmetric cryptography. The HN11 scheme suffers from a vulnerability for which a non-revoked consumer can collude with a revoked consumer and restore its access privileges. Li et al. (Li et al., 2017) proposed a scheme (LYHZS17) based on the HN11 one that fixes this vulnerability.

In short, in attribute-wise revocation, differently from indirect revocation, the key authority can stay offline and perform no task as far as no revocation occurs. Moreover, being attribute-wise revocation not time-dependent, a key revocation can come into force with immediate effect.

7.4 Producer Bandwidth Efficiency

In this section, we focus on the bandwidth efficiency of ABE schemes from the point of view of the data producers. The bandwidth overhead introduced by a scheme on the producers includes the *encryption bandwidth overhead*, i.e., the difference between the ciphertext size and the plaintext size, and the *key management bandwidth overhead*, i.e., the traffic related to key distribution and key revocation mechanisms. Indeed, the producers are also involved in key management processes to download new public parameters or even lists of revoked consumers.

A general and straightforward way to lower the encryption bandwidth overhead is to use the digital envelope technique described in Section 7.2. This is because symmetric-key encryption introduces much less encryption bandwidth overhead compared to ABE. Of course, as already highlighted, it makes the key management overhead more complex because symmetric keys must be stored by producers and consumers, and possibly revoked by the key authority.

Besides this basic technique, we identified three main strategies to lower the bandwidth overhead on the producer: (i) the use of a constant-size ciphertext (Zhou et al., 2013; Zhou and Huang, 2010; Doshi and Jinwala, 2014; Li et al., 2012; Zhang et al., 2014a,b; Ge et al., 2012; Herranz et al., 2010; Attrapadung et al., 2011; Emura et al., 2009; Phuong et al., 2014), (ii) the implementation of an efficient key management mechanism (Hur and Noh, 2011; Liu et al., 2018; Li et al., 2017), and (iii) the use of small group elements for the public parameters and ciphertexts (Agrawal and Chase, 2017).

Constant-Size Ciphertext

An effective strategy to reduce the encryption bandwidth overhead is to have ciphertexts of small or constant size. Typically, in many ABE schemes, the ciphertext size depends on the number of attributes either in the access policy (CP-ABE), e.g., BSW07, HW14-2, and LYZL18, or in the attributes set (KP-ABE), e.g., YWRL10, GPSW06-1, and AI09. Clearly, this dependency is detrimental to the producer's traf-

fic. On the other hand, schemes with constant-size ciphertexts reduce the traffic by disposing of such a dependency. However, we notice that a trade-off emerges between policy expressiveness and encryption bandwidth overhead. Usually, schemes with constant-size ciphertexts use poorly expressive access structures languages, while schemes with non-constant-size ciphertexts tend to use more expressive access structure languages, allowing the creation of access policies that cannot be built in the constant-size ciphertexts schemes. This is because, in poorly expressive access structures languages (essentially all the AND-based languages), the attributes in the ciphertext are condensed into a single $\mathbb{G}$ element.

Fig. 7.3 shows some prominent constant-size ciphertext schemes on a Cartesian plane. The x-axis denotes efficiency in terms of ciphertext size (the rightmost, the better); the y-axis denotes the expressiveness of the language (the higher, the better); schemes that are proved secure under standard assumptions are shown in bold. Moreover, the classic schemes BSW07 and GPSW06-1 are shown as a reference. We note that among the schemes with poor expressiveness, the one that features the largest ciphertext size is PYS14, which uses the $\text{AND}^*_\pm$ language and provides a ciphertext of $|\mathbb{G}_T| + 4|\mathbb{G}|$ bits.

A slightly smaller ciphertext is achieved by the scheme GZCMZ12-2, which features a ciphertext size of $|\mathbb{G}_T| + 3|\mathbb{G}| + |\mathbb{Z}_p|$ and uses the threshold monotonic (k-of$n$) language. Three schemes, namely, EMNOS09, ALP11, and ZCLLL14, which use AND_m, $\text{AND}_\pm$, and $\text{AND}_\pm$ languages, respectively, provide a ciphertext of size $|\mathbb{G}_T| + 3|\mathbb{G}|$. Among all the schemes considered for this strategy, ALP11 is the only one that follows the KP-ABE paradigm. It is interesting to point out that ZCLLL14 also provides a very efficient key revocation system concerning the producers. They must download only one $|\mathbb{G}|$ element to update the public parameters after a key revocation, instead of the whole set of public parameters needed in all the other schemes, typically $O(n)$, being n the number of attributes in the universe.

Then, a considerable number of schemes feature a ciphertext size of $|\mathbb{G}_T| + 2|\mathbb{G}|$. Among them, the schemes DJ14 and LGRDY12 provide the worst expressiveness (AND_m), whereas ZZCLL14 provides slightly better expressiveness, using AND^*_m. However, the schemes with the best expressiveness are HLR10, and GZCMZ12-1, because they use the threshold monotonic language.

Finally, the schemes that feature the smallest ciphertext size are ZH10 and ZHW13, with an overhead of only $2|\mathbb{G}|$. They are the only two schemes that do not need to embed a $\mathbb{G}_T$ element in the ciphertext. Indeed, in decryption, the data consumer combines the elements of its decryption key and the ciphertext to produce a $\mathbb{G}_T$ element, which is the symmetric key used to encrypt the actual message. However, ZH10 and ZHW13 have poor expressiveness, as they use the $\text{AND}^*_\pm$ and the AND^*_m languages, respectively.

A general technique for improving the expressiveness of the $\text{AND}_\pm$, $\text{AND}^*_\pm$, AND_m,

and AND^*_m access structure languages is to provide *redundancy* of decryption keys in KP-ABE or redundancy of ciphertexts in CP-ABE. For example, in a KP-ABE scheme, a single consumer could hold three different decryption keys. From the access structure point of view, this is like binding the consumer to an AND/OR gate access structure: the root node is an OR, and the three branches are the single decryption keys. This technique can benefit KP-ABE schemes such as ALP11: consumers simply hold two or more different keys, which can better describe their access rights. In an equivalent example for the CP-ABE paradigm, a producer could create and transmit three ciphertexts for every piece of information to be encrypted. Even though this technique aims to improve expressiveness, in CP-ABE this is detrimental to the producer bandwidth (and computation) performance, since for a single piece of data the producer must transmit (and compute) two or more different ciphertexts. We investigate this technique in our simulations in Section 7.5. Moreover, the scheme AHMTY16 proposes a full monotonic KP-ABE scheme with a trade-off between decryption key size and ciphertext size. In the extreme case of ciphertext size optimization, such scheme features a ciphertext size of $|\mathbb{G}_T| + 6|\mathbb{G}|$. It may seem more than the other schemes, however, since this scheme uses KP-ABE, the ciphertext does not need redundancy. Moreover, since it has full monotonic access structures, the decryption key does not need redundancy, either. This makes such scheme the third-best performing solution (concerning the analyzed KPI) discussed in this section: this may be a good solution if one prioritizes expressiveness over ciphertext size, while still needing them both.

Be aware that some schemes, e.g., JSMG18-1 and JSMG18-2, achieve a small ciphertext size, but they are not very bandwidth efficient. The ciphertext itself is indeed small, but the producer must create and transmit additional cryptographic material along with each ciphertext to make the scheme work. Indeed, such schemes provide an unusual technique for updating the policy of an existing ciphertext, and this feature comes with a cost in terms of producer bandwidth. They can either add (JSMG18-1) or remove (JSMG18-2) the required positive value of an attribute from the policy embedded in the ciphertext, from a minimum of 1 to a maximum of m attributes inside a single policy. However, to do so, they must upload to the data storage $(m - |\mathcal{P}|)|\mathbb{G}|$ more bits along with the ciphertext, increasing further the encryption bandwidth overhead (being $|\mathcal{P}|$ the number of attributes used inside the policy).

As a side note, the schemes EMNOS09, LGRDY12, and PYS14 are the only schemes with constant-size ciphertext that have been formally proved to be secure under standard assumptions.

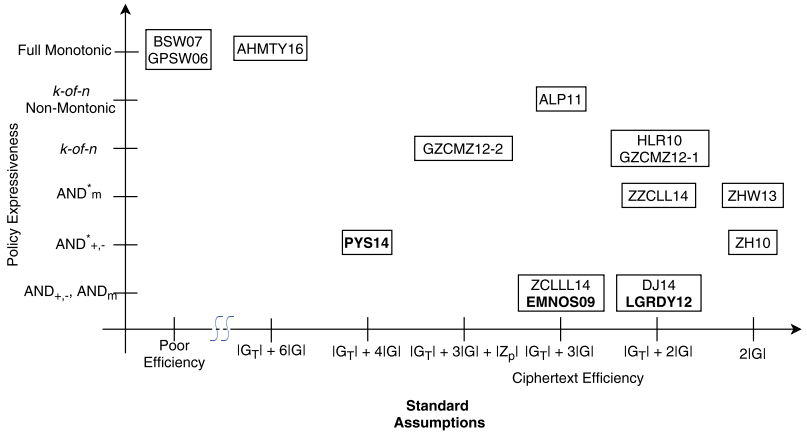


Figure 7.3. The analyzed constant-size ciphertext schemes. The classic CP-ABE (BSW07) and KP-ABE (GPSW06-1) schemes are shown as a reference. Schemes in bold have been proved secure under standard assumptions.

Efficient Key Management

As outlined in Section 7.3, there are three different key revocation mechanisms: direct, indirect, and attribute-wise. We aim to identify the mechanism that reliably impacts the least the producer bandwidth.

An example of an attribute-wise revocation that is not reliably convenient for the producer is YWRL10. Indeed, we recall that when the key authority wants to revoke a set of attributes λ embedded in a decryption key, it generates a new version of the public parameters for the attributes in λ. This means that a producer has to download a number of elements in $\mathbb{G}$ equal to λ after each revocation. Notably, the required bandwidth can be optimized if the producer maintains up-to-date only a subset of the public parameters (e.g., a sensor that encrypts sensed data always under the same attribute set). In this case, the producer has to download several elements in $\mathbb{G}$ from 0 to λ after each revocation, depending on how many attributes it uses are inside λ. This is undoubtedly the most unpredictable mechanism in terms of required bandwidth since its traffic depends on the number of attributes involved in the revocation. On the other hand, the schemes HN11 and LYHZS17 also feature an attribute-wise revocation mechanism, but the producers are not required to download anything after a revocation happens. In fact, a producer will always encrypt data using the same public parameters, and then it uploads the ciphertext on the data storage, which enforces the revocation.

In the direct revocation mechanism, usually, the producer must hold a list of identifiers of the revoked users. Typically, this is the only cost sustained by producers in terms of bandwidth for key management operations. The basic idea is to create a “trapdoor" in the ciphertext by using the identifiers of all the revoked consumers. Such a trapdoor “activates" when a revoked consumer tries to decrypt

the ciphertext with its decryption key and, therefore, its identifier. The trapdoor hides a needed quantity to decrypt the ciphertext, so this step cannot be avoided or cheated in any way. The only cost in terms of producer bandwidth is, therefore, the revocation list. The more the system runs, the more consumers will be revoked: downloading an entire revocation list each time can be detrimental for the producer bandwidth and may also impact its limited storage capabilities. The direct revocable scheme LYZL18 (see Section 7.3) keeps the revocation list short by removing expired decryption keys to lessen key management bandwidth overhead on the producer.

Finally, in the indirect revocation mechanism, the producer potentially does not have to download anything. For example, in BGK08-4 and QZZC17, the producer contributes to revocation only by encrypting the ciphertext with an additional attribute referring to the encryption time. This is a task that requires no interaction with the other parties and saves bandwidth. Therefore, this approach is very convenient for the producers since they neither have to download updated public parameters after each revocation nor have to hold an updated copy of the revocation list.

Small Group Elements

Using small group elements in the ciphertext can reduce the bandwidth overhead due to the ciphertext size. A viable strategy is to adapt ABE schemes to use Type III pairing by converting the highest number of $\mathbb{G}$ elements to $\mathbb{G}_1$ elements in the ciphertext, as discussed in Section 7.2. Smaller $\mathbb{G}_1$ elements are convenient to compute some operations in encryption more efficiently, and they also save the producer a conspicuous amount of bandwidth. Using fewer bits for a single $\mathbb{G}_1$ group element dramatically reduces the bandwidth needed to upload a ciphertext to the data storage.

For example, we can compare a ciphertext of the GPSW06-3 KP-ABE scheme with a ciphertext of the AC17-5 scheme, which represents a possible Type III conversion of GPSW06-3. To do so, we use the standard Type I (curve `a.param`) and Type III (curve `d201.param`) curves of the PBC library[3] with 80-bit security and 20 attributes in the ciphertext. The resulting encryption bandwidth overhead of GPSW06-3 is 1408 bytes, while that of AC17-5 is 654 bytes, leading to a 53.6 % overhead reduction for *every* ciphertext uploaded to the data storage.

In other schemes, the bandwidth saving is less pronounced. For example, we compare the BSW07 CP-ABE scheme with the AC17-3 scheme, which is a possible Type III conversion of BSW07, with the same curves as before and 20 attributes in the ciphertext. The resulting encryption bandwidth overhead of BSW07 is 2752 bytes, while that of AC17-3 is 2237 bytes, leading to an 18.7 % overhead reduction.

[3] https://crypto.stanford.edu/pbc

Note that, in contrast with the strategy of adopting schemes with constant-size ciphertexts (Section 7.4), this strategy can improve bandwidth efficiency without jeopardizing the expressiveness of the policies. Note also that the two strategies can be applied together to gain even more efficiency.

7.5 Experimental Evaluation

In this section, we evaluate the performance of a variety of ABE schemes that we described in the previous sections through an event-based Matlab simulator. We selected a scheme to represent each strategy discussed, except for "alternative mathematics" given its dubious security guarantees. All the simulated schemes excel in one or two KPIs, but none of them in all the three KPIs at once. The simulated KP-ABE schemes are BGK08-4, AI09, YWRL10, HW14-1, and AC17-5. The simulated CP-ABE schemes are ZH10, HW14-2, AC17-1, QZZC17, and LYZL18. Table 7.2 shows in which KPI(s) each selected scheme excels. The schemes BKG08-4 and QZZC17 implement indirect key revocation and efficient key management: producers are not affected by revocations. ZH10 is the scheme with the smallest constant-size ciphertext. AC17-1 and AC17-5 use Type III pairings and have ciphertexts with small group elements. YWRL10 implements attribute-wise revocation. AI09 implements a hybrid revocation. HW14 exploits encryption offloading to reduce producers' computational load. LYZL18 implements direct key revocation and efficient key management.

Table 7.2
KPIs of Simulated Schemes

Scheme Name	Producer CPU Efficiency	Key Authority Bandwidth Efficiency	Producer Bandwidth Efficiency
BGK08-4		✓	✓
AI09		✓	
YWRL10		✓	
ZH10			✓
HW14	✓		
AC17-1	✓		✓
AC17-5	✓		✓
QZZC17		✓	✓
LYZL18		✓	✓

Each simulated scheme comprises a key authority, a data storage, many producers, and many consumers. The simulator runs for a simulated period of time within which it randomly generates four types of events: data production, data consumption, consumer join, and key revocation. The corresponding algorithm of each scheme, e.g., **Encrypt**, **Decrypt**, etc., is simulated within these events. The simulator neither performs actual math operations nor implements some protocol for

exchanging messages between the entities. Rather, it records the number and type of math operations and eventually estimates the total computational load for each entity. Moreover, the simulator records the number and the size of the messages exchanged between the entities and estimates their experienced traffic overhead.

Simulator Description and Configuration

The simulator simulates a generic architecture as the one of Fig. 7.1, in which (i) producers produce ciphertexts and upload them on the data storage, (ii) consumers obtain ciphertexts from the data storage and decrypt them, and (iii) the key authority generates decryption keys for joining consumers and also (iv) revokes decryption keys.

The simulator defines a universe of 100 attributes for each scheme. Scheme-specific attributes, such as the dummy attribute (used in YWRL10) or time attributes (used in BGK08-4, AI09, and QZZC17), are not considered in this number but are individually added to the schemes that need them. In the simulation of KP-ABE schemes, each ciphertext is labeled with a set of 30 distinct attributes chosen randomly among those in the universe, and each decryption key embeds an access policy consisting of 10 distinct attributes chosen randomly among those in the universe. In a dual way, in the CP-ABE schemes, a ciphertext embeds an access policy of 10 distinct random attributes, and a decryption key is labeled with 30 distinct random attributes. Within the same simulation, the access policy shape is fixed for all the ciphertexts (in CP-ABE) or decryption keys (in KP-ABE). We set the access policy shape to be in Disjunctive Normal Form (DNF), with an OR at the root and three AND children with three, three, and four attributes, respectively. Fig. 7.4 shows an example.

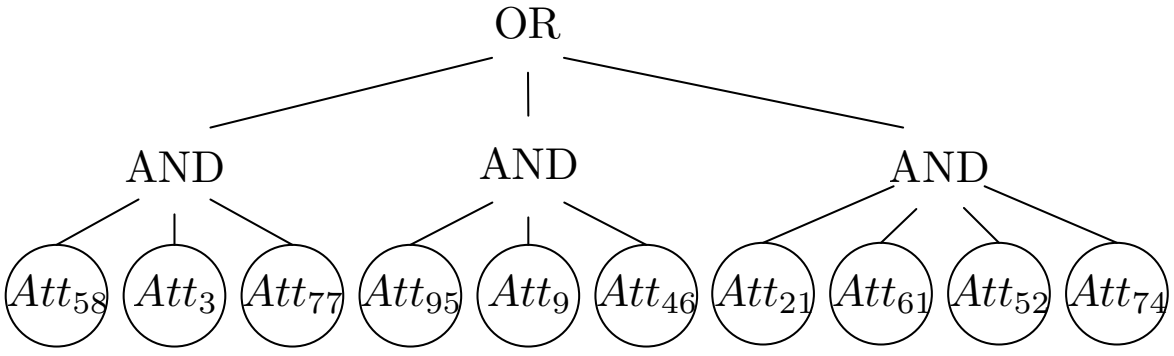


Figure 7.4. Example of simulated access policy in DNF shape.

The simulator can be configured to start with an initial number of producers and consumers. In our simulations, we start with 10 000 producers and 10 000 consumers. The number of producers remains the same throughout the simulation,

while the number of consumers varies every time a consumer join event or a key revocation event occurs. During a preliminary phase, the simulator creates an initial database of 10 000 ciphertexts and decryption keys for the consumers according to the methodology described above. As far as decryption key generation is concerned, we ensure that each decryption key can decrypt at least one of the initial ciphertexts.

After these preliminary operations, the simulator starts generating the events and recording the metrics. The events of data production, data consumption, consumer join, and key revocation are modeled as Poisson processes. More in detail, each producer generates a data production event every hour on average, each consumer generates a data consumption event every hour on average, the key authority generates a key revocation event every day on average, and a consumer join event is generated every day on average. At each data production event, we simulate, for each scheme, that the producer encrypts a new piece of data and uploads the ciphertext on the data storage. The new ciphertext is created according to the methodology described above. At each data consumption event, we simulate, for each scheme, that a random consumer downloads a random ciphertext from the data storage (among those its key is allowed to decrypt) and decrypts it. At each key revocation event, we simulate, for each scheme, that the key authority revokes a random consumer from the system. At each consumer join event, we simulate, for each scheme, that the key authority generates a new decryption key according to the methodology described above, encrypts it with the consumer's public key, and uploads it to the data storage; then, the consumer downloads and decrypts it. These events are executed until the simulated period of time, which we set to one month, is reached.

In the final phase of the simulation, the simulator averages the recorded metrics to obtain the results per single producer and consumer. The metrics, i.e., computational load and traffic overhead, are expressed in units of time and units of storage, respectively. We assume all producers to be IoT devices, i.e., Zolertia RE-Motes (Zolertia S.L., 2017). Therefore, to determine the running time of the various basic math operations, we perform benchmarks that use the PBC library on such a device. We suppose that all the simulated schemes that employ symmetric pairing use a Type I pairing with 512-bit $\mathbb{G}$ group elements and 1024-bit $\mathbb{G}_T$ group elements, while those that employ asymmetric pairing use a Type III pairing with 201-bit $\mathbb{G}_1$ group elements, 603-bit $\mathbb{G}_2$ group elements, and 1206-bit $\mathbb{G}_T$ group elements[4]. These curves give an equivalent security level of 80 bits. Table 7.3 shows the results of our benchmarks.

The simulator performs 30 repetitions with the same configuration but with differ-

[4]These curves come with the PBC library, and their parameters can be found in the files `a.param` and `d201.param`

Table 7.3
Pairing-Based Cryptography Benchmarks on Zolertia RE-Mote. For Each Operation, 100 Repetitions Are Averaged and 95 %-Confidence Intervals Are Computed

Type I pairing	time (ms)	
	Mean	95 % CI
$\mathbb{G}$ point-scalar multiplication	265	4.05×10^{-3}
$\mathbb{G}_T$ modular exponentiation	74	1.48×10^{-3}
bilinear pairing	5673	0.80×10^{-3}
hash $(\mathcal{H} : \{0,1\}^* \to \mathbb{G})^\dagger$	6088	2.17×10^{-3}

Type III pairing	time (ms)	
	Mean	95 % CI
$\mathbb{G}_1$ point-scalar multiplication	73	1.22×10^{-3}
$\mathbb{G}_2$ point-scalar multiplication	763	9.56×10^{-3}
$\mathbb{G}_T$ modular exponentiation	288	4.84×10^{-3}
bilinear pairing	9195	0.71×10^{-3}
hash $(\mathcal{H} : \{0,1\}^* \to \mathbb{G}_1)$	225	1.00×10^{-3}

† From our experiments, this value is largely independent of the input size.

ent random seeds to achieve statistically sound results. The final results are averaged, and confidence intervals are computed.

Simulated Schemes

In the following, we give details on scheme-specific configurations.

The AI09 scheme allows producers to arbitrarily choose whether to encrypt the ciphertext in "direct revocation mode" or "indirect revocation mode". We modeled this opportunity with a random choice during the data production event, and we simulated two variants of the AI09 scheme. In the first variant, called AI09(H), where (H) stands for hybrid, we set the probability of producing a ciphertext in direct revocation mode to 0.5. In the second variant, called AI09(D), where (D) stands for direct, we force the scheme to act as a pure direct revocation scheme by setting this probability to 1. We neglect the pure indirect variant because it is very similar to the BGK08-4 scheme.

For the schemes that use the binary tree, i.e., BGK08-4, QZZC17, and AI09, we create a complete binary tree of the minimum size that can accommodate the initial consumers and the joining consumers. Moreover, we set the duration of the time period to one day. At the end of each time period, which we model through a periodic event, the key authority creates a key update and stores it on the data storage.

The ZH10 scheme allows only access policies composed of an AND gate on Boolean attributes with wildcards ($\text{AND}^*_\pm$). For a fair comparison, we improve the ZH10 expressiveness using redundancy. We realize DNF-shaped policies by encrypting the same piece of data for a number of times equal to the number of AND

gates in the original DNF. Therefore, the producer creates three ciphertexts: each ciphertext specifies three or four positive attributes, and the remaining attributes of the universe are wildcards. On the other hand, a decryption key contains 30 positive attributes and 70 negative attributes. We simulate that the producer uploads three ciphertexts on the data storage, and the consumer downloads only the one it can decrypt.

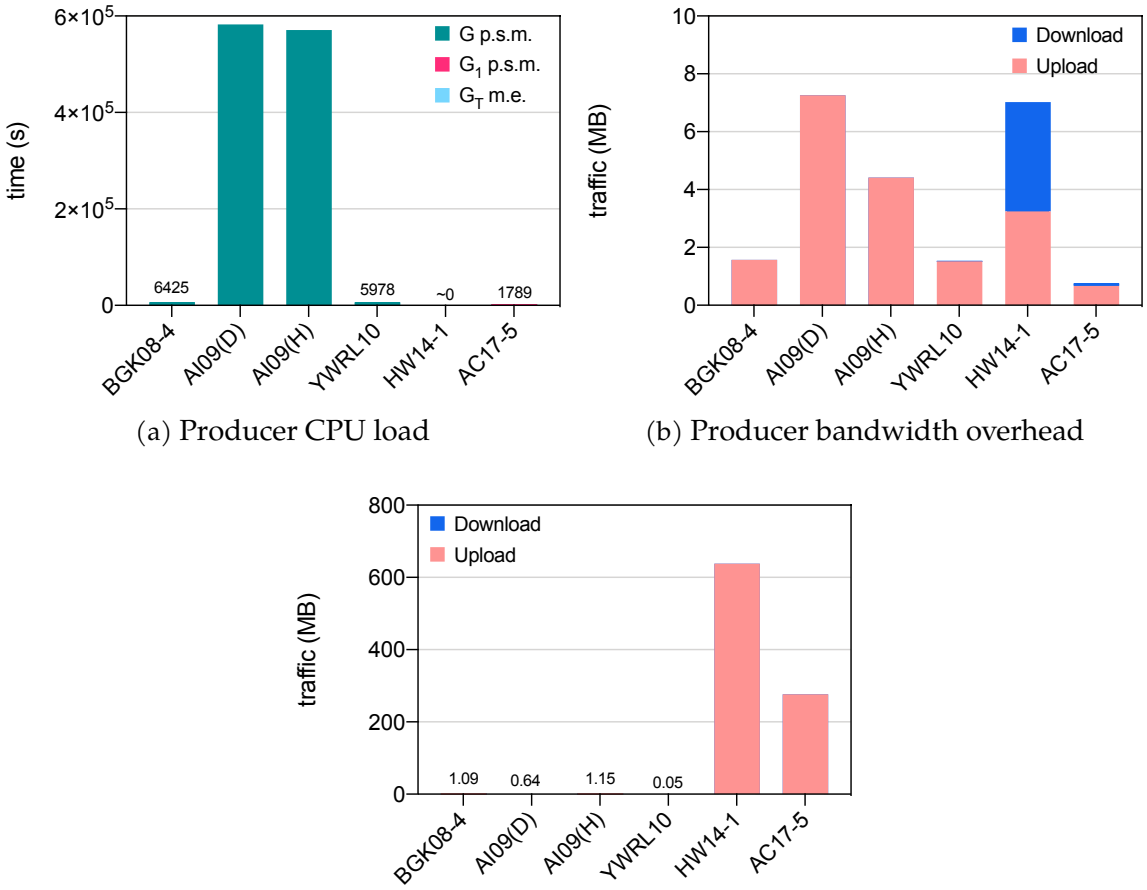
For the HW14-1 and HW14-2 schemes, we simulate that the offline phase is outsourced to a trusted resourceful device, and the resulting pre-processed quantities are transmitted to the producers through some secure channel. When the online phase –which comes at no cost for the producer– is completed, the producer uploads the ciphertext on the data storage.

As regards the schemes that do not come with a revocation mechanism, i.e., ZH10, HW14-1, HW14-2, AC17-1, and AC17-5, we implemented for them the naive revocation mechanism described in Section 2.2. For these schemes, during the consumer join event, the key authority generates as many decryption keys for the joining consumer as the number of key revocations occurred so far. All these keys have the same access policy (or attribute set), but each one has been generated with a different master key. In this way, the consumer can access ciphertexts generated before its joining time.

Discussion

In the following, we show the performance of the selected ABE schemes against the key performance indicators. We treat KP-ABE and CP-ABE schemes separately since such two paradigms are not meaningfully comparable. We first analyze the results obtained by simulating the KP-ABE schemes, and later we focus on CP-ABE.

Fig. 7.5 shows average values and 95 %-confidence intervals of the KPIs for the simulated KP-ABE schemes. We note that the best scheme concerning the producer's performance is AC17-5 which can keep both the CPU load and bandwidth overhead low by taking advantage of the asymmetric pairing. Indeed, as we notice from Table 7.3, a point-scalar multiplication in $\mathbb{G}_1$ is roughly four times faster than a point-scalar multiplication in $\mathbb{G}$. Moreover, being a $\mathbb{G}_1$ group element smaller than a $\mathbb{G}$ element, the producer bandwidth for AC17-5 is the lowest among all the schemes tested, albeit the producer must download at each revocation event the public parameters because of the naive revocation. The schemes BGK08-4 and YWRL10 turn out to be slightly less efficient about the producer. However, if we look at the key authority bandwidth efficiency, we notice that these schemes handle the key revocation very efficiently, while in the AC17-5 scheme, at each revocation, the key authority is burdened with the creation of new public parameters and new decryption keys for non-revoked consumers. The AI09 scheme, which is revocable, performs well only concerning the key authority bandwidth efficiency. Indeed, the direct revo-



(a) Producer CPU load

(b) Producer bandwidth overhead

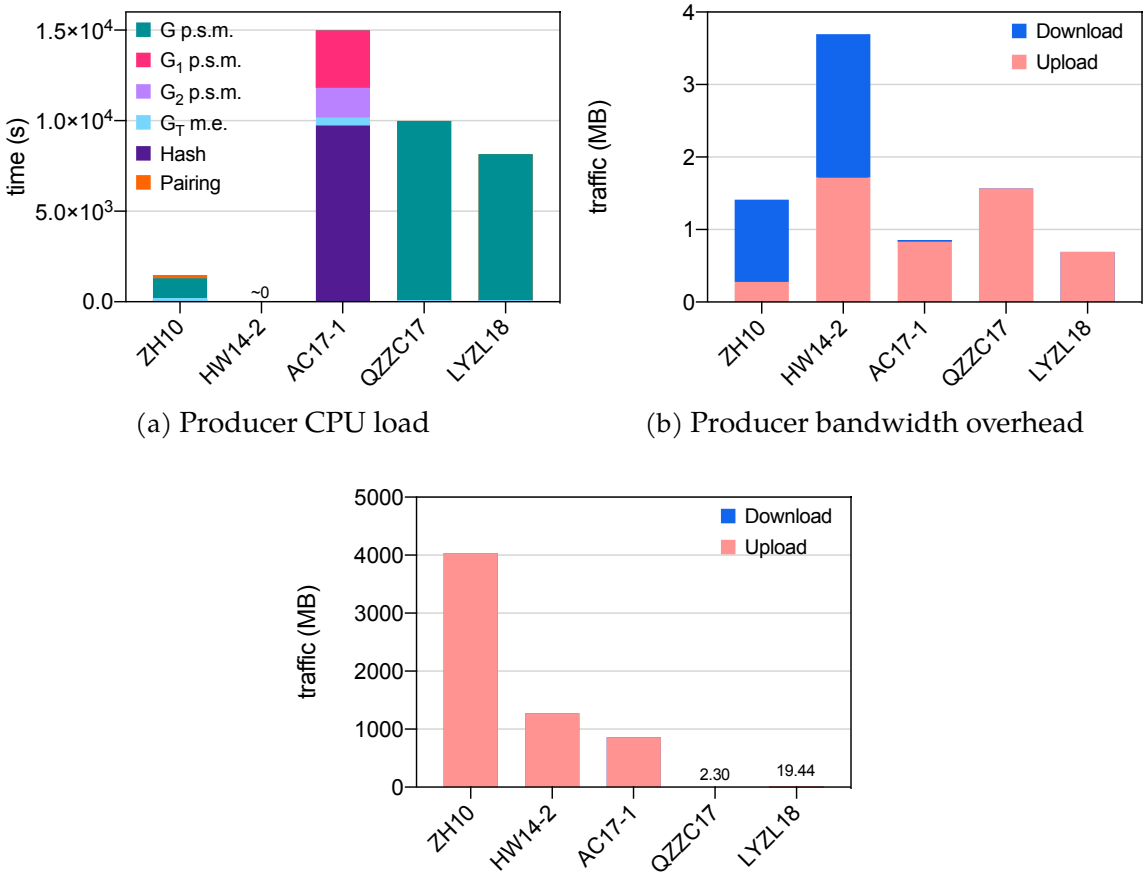
(c) Key authority bandwidth overhead

Figure 7.5. Comparison of KP-ABE schemes performance concerning the three KPIs.

cation mode used in both AI09(D) and AI09(H) reduces the producer's efficiency since it must perform more computations and generate larger ciphertexts than when indirect revocation mode is used.

In the HW14-1 scheme, the producer does not perform burdensome operations thanks to encryption outsourcing. It only executes a few modular multiplications in $\mathbb{Z}_p$, which are enough time-efficient to be negligible, therefore we do not simulate them. On the other hand, the producer must download the pre-processed quantities, which heavily impacts its bandwidth efficiency. Also, the key authority bandwidth overhead is high because of the naive revocation. Note how the key authority is efficient in terms of bandwidth in revocable schemes, namely two orders of magnitude more efficient than non-revocable schemes.

Fig. 7.6 shows average values and 95 %-confidence intervals of the KPIs for the simulated CP-ABE schemes. We note that the producer CPU load of HW14-2 is negligible as it happens for its KP-ABE counterpart HW14-1. Unlike the previous simulation, the simulated asymmetric-pairing scheme (AC17-1) is not the one



(a) Producer CPU load

(b) Producer bandwidth overhead

(c) Key authority bandwidth overhead

Figure 7.6. Comparison of CP-ABE schemes performance concerning the three KPIs.

with the lowest computational load concerning the producer. In this scheme, the computation of the hashes heavily impacts the performance, and they are not pre-computable. With a policy of 10 attributes as in our simulation, at each encryption, the producer computes 60 hashes and spends about 13.5 s just for computing hash values; the advantages of asymmetric pairing are therefore nullified. Note that this contrasts with the results in (Agrawal and Chase, 2017), which reports hashes to be very fast, and thus AC17-1 to be very efficient in encryption. Such results were obtained on a PC-class device and not on constrained devices. On the contrary, we experimentally noticed that hashes are quite slow on the Zolertia RE-Mote platform. This suggests that AC17-1, though very efficient on PCs, loses much of its efficiency when adopted in IoT applications.

In our simulations, the scheme that performs best concerning the producer efficiency is ZH10. Even though the producer generates three ciphertexts instead of one at each data production (for expressiveness fairness), ZH10 features the best producer CPU efficiency and a good producer bandwidth efficiency. In Fig. 7.6(b),

its pronounced download overhead is due to the naive revocation. Compared to the other schemes with naive revocation, i.e., HW14-2 and AC17-1, the ZH10 scheme performs worse. This is because, unlike HW14-2 and AC17-1, ZH10 is a small-universe scheme, and thus public parameters are large. If we look at Fig. 7.6(c), we notice that ZH10 has the worst key authority bandwidth efficiency. We recall that in the naive revocation, the key authority generates public parameters but also new decryption keys for non-revoked consumers. Compared to HW14-2 and AC17-1, in ZH10 the key authority must generate larger keys.

The revocable schemes LYZL18 and QZZC17 are light in terms of key authority bandwidth overhead. About the producer, the LYZL18 scheme is more efficient, albeit it implements a direct revocation mechanism, and the producers must perform additional computations to enforce revocation. However, we recall that the revocation list in the ciphertext is condensed into a single $\mathbb{G}$ element, which also helps to keep the bandwidth overhead on the producer very low (Fig. 7.6(b)). In the QZZC17 scheme, the producer experiences a slightly higher CPU load and bandwidth overhead. Nonetheless, this scheme is particularly suitable for applications in which a high efficiency on the consumer is needed (see Sections 7.6 and 7.6).

7.6 Accessory Performance Indicators

Producer Storage Efficiency

We identified in the literature three main strategies to improve the storage efficiency of the producer: (i) adopting large-universe schemes (Bethencourt et al., 2007; Ostrovsky et al., 2007; Attrapadung and Imai, 2009), (ii) storing partially the public parameters (e.g., the majority of small-universe schemes can adopt this strategy), and (iii) using storage-efficient key revocation mechanisms (Liu et al., 2018).

In an ABE scheme, the minimum amount of information a producer must store is the public parameters, whose size depends on the universe type. Usually, large-universe schemes have small public parameters, while small-universe schemes have large public parameters. In particular, in small-universe schemes, the size of the public parameters grows linearly with the number of attributes in the universe. On the contrary, in large-universe schemes, the size of the public parameters is typically constant and composed of a few group elements. For example, in the GPSW06-1 (small-universe) scheme initialized with a universe of 100 attributes, the size of the public parameters is 6528 bytes, while in the BSW07 (large-universe) scheme, the size of the public parameters is only 320 bytes[5]. Therefore, a viable strategy to keep the storage overhead low on the producer is to choose a large-universe scheme that features small and constant-size public parameters.

[5]considering a security level of 80 bits.

Regarding the small-universe schemes, we identify a general strategy that can be applied to plenty of such schemes to reduce the required storage on the producer: storing only a portion of the public parameters. In small-universe schemes, the public parameters typically include one $\mathbb{G}$ element (component) for each attribute in the universe. If the producer uses only a portion of them, e.g., it always encrypts with the same set of attributes, it can store only the components it uses for encryption. Referring to the previous example, if a producer uses only 30 attributes out of 100, the storage overhead can be reduced from 6528 to 2048 bytes. Unfortunately, this strategy is not viable for the vast majority of the schemes with constant-size ciphertext that we surveyed in Section 7.4. By their very nature, to create a ciphertext of constant size, these schemes require computations involving each part of the public parameters.

The revocation mechanism also can impact the producer storage. Specifically, in direct revocation, each producer must store the revocation list, which contains the revoked identifiers. An identifier can be expressed as a group element or as a mere progressive number. In both cases, the producer storage might be severely affected when the system reaches a large number of revoked consumers. Therefore, using an indirect revocation mechanism typically leads to a better producer storage efficiency. In direct revocation, to relieve the storage overhead on the producers, some schemes, e.g., LYZL18, embed an expiration date in the decryption keys. In this way, the expired decryption keys can be excluded from the revocation list.

Consumer CPU Efficiency

The computation efficiency of an ABE scheme on the data consumers includes the operations the consumers perform to decrypt data and those they perform for key management procedures. As already outlined, the key management operations are typically much less frequent, so, in this section, we focus on decryption efficiency to represent the overall consumer CPU efficiency. We identified in the literature three main strategies to improve the efficiency of decryption on the consumer: (i) outsourcing burdensome decryption operations (Huang et al., 2018; Cui et al., 2016; Qin et al., 2017), (ii) using constant-complexity decryption (Doshi and Jinwala, 2014; Li et al., 2012), and (iii) adopting mathematics alternative to pairings, for example, ECC (Yao et al., 2015) or RSA (Odelu et al., 2017) mathematics.

The first strategy leverages data storage with abundant computational capabilities, for example, a cloud server. This is necessary because, at each data request from the consumers, the data storage must perform some operations on the requested ciphertext before transmitting it. In the CDLQ16 and QZZC17 schemes, the data storage uses public information related to the requesting consumer id to transform the ciphertext so that only the consumer id can decrypt it. The transformed ciphertext is sent to the consumer, which finalizes the decryption at a low and constant cost,

i.e., one operation in $\mathbb{G}_T$ in CDLQ16 and two pairings in QZZC17. Note that the data storage can be untrusted since it does not hold any secret information to transform the ciphertext. Additionally, anyone can verify the correctness of the transformation by performing the same algorithm executed by the data storage. In HWY18, too, the decryption is outsourced to the data storage that transforms the ciphertext and leaves just one modular exponentiation in $\mathbb{G}_T$ to the consumer for retrieving the plaintext. In all these schemes, the decryption performed by the consumer is independent of the attributes in its decryption key and is performed in constant time.

The second strategy tries to reduce the cost of the decryption algorithm. Very often, its complexity grows linearly with the number of attributes used to satisfy the access policy. However, for many schemes with limited expressiveness (see schemes surveyed in Section 7.4), the cost of decryption is low and constant, and it is usually dominated by a few pairings. For example, the decryption cost is fixed to two pairings in the schemes DJ14, LGRDY12, EMNOS09, and ZZCLL14. In many cases, limited expressiveness results in a good CPU efficiency for consumers.

The third strategy is to adopt mathematics different from pairings, e.g., ECC (YCT15 and OD16) and RSA (ODKCJ17), which lighten the consumer CPU because they eliminate the burdensome pairing operation usually employed in decryption. However, at the time of writing, we do not suggest their employment for the security concerns explained in Section 7.2.

Consumer Bandwidth Efficiency

We state in Section 7.4 that the main tasks involving the producer bandwidth overhead are the encryption bandwidth overhead and the key management bandwidth overhead. This applies to the bandwidth overhead of the consumer as well. Two of the three strategies described in Section 7.4 that improve the producer bandwidth efficiency are also good for improving the consumer bandwidth efficiency: using schemes with constant-size ciphertexts, and using small group elements. Furthermore, we identified in the literature two additional strategies to improve the efficiency of the consumer bandwidth: (i) partially outsourcing the decryption to the data storage (Cui et al., 2016; Qin et al., 2017), and (ii) using a direct key revocation mechanism (Liu et al., 2018; Phuong et al., 2015).

In the first additional strategy, since the consumer must download the ciphertext from the data storage, we can use this intermediary to lighten the burden on the consumer. Oftentimes, the data storage (typically in the form of a cloud server) manipulates the stored ciphertexts in some way before sending them to the consumers. Indeed, in schemes like CDLQ16 and QZZC17, the cloud storage does shrink the ciphertext size before sending it to a consumer: from an arbitrary size (that grows linearly with the size of the access policy) to a constant size of $|\mathbb{G}_T|+2|\mathbb{G}|$. This strategy can be effective for the consumer's bandwidth. However, we must point out

that not always such a manipulation reduces the number of bytes that the consumer has to download. In fact, HN11 requires the data storage to do some operations over a ciphertext before sending it to the requesting consumer. Among other operations, the data storage has to prepend some additional information to the ciphertext, therefore *increasing* the number of bytes that the consumer must download.

The second additional strategy is to use a direct revocation mechanism. Indeed, the direct revocation mechanism (e.g., LYZL18) is the most effective since the consumer does not have to download any cryptographic quantity to update its decryption key. In contrast, indirect revocation and attribute-wise revocation impact the consumer, which must download key update material, either periodically or at each revocation event.

7.7 Answer

What is the Most Suitable ABE Scheme for my System?

It turns out, it depends! (Who would have guessed it?). Indeed, we argued that some features are more valuable than others in a precise context, and we also simulated the best schemes that emerged from the theoretical dissertation to compare their performance. In brief, we provided the reader with tools to choose the most suitable ABE scheme for any IoT scenario. In particular, we addressed the best strategies to improve any of the six PIs, and we provided a MATLAB program to simulate the performance of virtually any ABE scheme regarding each PI.

Chapter 8

Is it Feasible to Leverage CP-ABE in the Automotive Environment?

Over the last few decades, we have seen a complete transformation of the automotive world. Vehicles rely more and more on electronic components to provide new features to the customers. With well over 80 ECU's per vehicle (NXP, 2018), software maintenance is a severe issue. In industry, it has been estimated that the number of bugs per 1000 lines of code oscillates from 0,5 to 25 (McConnell, 2009). It would be foolish to think that a vehicle on the market has no bugs, and it would be even more foolish to assume that none of them can lead to a vulnerability issue. New attacks and exploits (Kocher et al., 2019; Lipp et al., 2018) emerge every day, and it is impossible to prevent them all. However, it is possible to find a solution to a newly discovered vulnerability and fix it with a software or firmware update in most cases. A safe way to update the software or the firmware of a vehicle is to bring it to the nearest licensed workshop: clearly, this scenario must be avoided since it can cause a disservice to the customer and extra costs for the automotive OEM (Original Equipment Manufacturer). This is a serious problem that also caught the attention of the European Processor Initiative (EPI) project committee (Union, 2019; Kovač et al., 2020) and its partners such as BMW and Elektrobit. One solution is to update the software/firmware *over the air* (OTA), with the user that can manage the update in the same way as he/she does with a smartphone or a home PC. The basic idea is that inside each vehicle there is a particular ECU, called *gateway*, that connects the outer world to all the vehicle's ECUs. For example, the gateway provides for infotainment to the vehicle's passengers or — in our case — an Internet connection with the manufacturer to download the updates. There are many state-of-the-art solutions (Asokan et al., 2018; Karthik et al., 2016; NXP, 2018) that already implement OTA software updates, and all of them focus on providing the authenticity and the integrity of the update. Confidentiality, instead, is treated as an optional security feature. Unfortunately, the *Intellectual Property* (IP) of the update is not protected

since in case the update is sent to the vehicle without any encryption, then the competitors can easily capture and analyze its content. This is not desirable, particularly if the update contains innovative countermeasures to new attacks or introduces new features for the vehicle. Another problem is that, even if the update is confidentially transmitted through the establishment of a secure channel (e.g., TLS), confidentiality is not guaranteed when the update is *at rest*. We recall that a piece of data is "at rest" whenever it is not traveling through the Internet; for example, data stored on a cloud server is at rest. This means that a manufacturer that uses secure channels to provide confidentiality to its updates cannot use a third-party untrusted server for storage and distribution since when an update is uploaded to such servers, it will not be encrypted. Furthermore, even if the manufacturer uses its own trusted cloud server to transmit the update to each vehicle, once the update arrives at the vehicle's gateway, it can still be easily captured by someone who tampered with the gateway itself. This can happen since the gateway is the only ECU directly connected to the Internet, and therefore prone to cyber-attacks and more vulnerable to tampering than all the other ECUs. Indeed, even in the Autosar *Specification of Update and Configuration Management* document (AutosarAdaptive, 2019), it is specified that it is convenient to have a dedicated ECU, different from the gateway, in charge of managing the SW update of the vehicle. A solution to the "data at rest" problem is to encrypt the update itself asymmetrically (e.g., using RSA) so that only such a dedicated ECU (not even the gateway) can decrypt it. However, this approach can be costly because the manufacturer should encrypt the update as many times as the vehicles to be updated. *Attribute-Based Encryption* (ABE) dramatically reduces the cost of multiple-receiver end-to-end encryption and solves the problem of update at rest, making it worth and efficient to provide confidentiality. Using ABE, even if an adversary successfully tampers the vehicle's gateway, the update is still encrypted and signed with long-term keys in possession of the dedicated ECU, thus making the tampering useless. The only way that an adversary has to analyze a copy of the update is to tamper with either the dedicated ECU or the ECU that needs the update. The automotive industry is aware of such risks, and to contrast such issues, they developed some security strategies such as the *multi-layer security architecture* (NXP, 2018). The internal architecture of a vehicle is classified over different security levels based on the security requirements of the involved application, ranging from low-level security (e.g., infotainment) to high-level security (e.g., the brake system in an autonomous vehicle). Among other things, this strategy ensures that the ECUs not directly connected to the Internet are hard to tamper with.

Despite many high-quality works that have been published during the years (Girgenti et al., 2019; Ambrosin et al., 2015, 2016) presenting the feasibility of ABE on a wide range of devices, to the authors' knowledge, the literature has not tested the impact of the ABE on a real hardware automotive embedded platform. In this chap-

ter, our task is to demonstrate that ABE schemes, in general, are well supported by a platform that is very close to a real ECU mounted on a vehicle. To do this, we selected the CP-ABE by Bethencourt et al. (Bethencourt et al., 2007) since it is the one considered in the previously cited feasibility works. Showing that this scheme has little-to-no impact on the OTA process, we show that ABE can perform well over such a category of devices.

The contribution of this chapter consists in: (i) showing an ABE technique for OTA secure update of software/firmware that can be seamlessly integrated into state-of-the-art solutions; (ii) proving that ABE is compliant with the in-vehicle network organization in modern cars as well as with the computing capabilities of real automotive ECUs; (iii) provide an experimental evaluation of the ABE performances on a real automotive compliant platform, namely the Xilinx ZCU102 board. The rest of the chapter is structured as follows: in Section 8.1 we give some backgrounds and show the related works; in Section 8.2 we explain the setup of our performance evaluation; in Section 8.3 we show and discuss our results; and finally Section 8.4 ends the chapter, answering the question inquired.

8.1 Related Work

Over the Air Frameworks

The “Over the Air" update solution is the future of software and firmware update concerning the ECUs inside a vehicle. To the user, not having to bring the car to the nearest licensed workshop is a great relief, and it also can improve the chance that the update is actually installed. Moreover, reported statistics show that automotive OTA can reduce warranty costs by a factor of 2 (Aptiv, 2020).

There are some state-of-the-art solutions that implement end-to-end encryption for OTA FW/SW update as *vConnect* (Vector, 2020). Their solution is to establish a secure channel through an *encrypted session* between their servers and the vehicle’s gateway. This is dangerous because after the image download is completed, it can be considered “at rest" inside the gateway. The gateway is the most probable ECU to be compromised since it is the only one directly connected to the Internet. In contrast, if a company were to adopt the ABE OTA SW update technique shown in this chapter, the gateway should forward the downloaded encrypted and signed image to the *Update and Configuration Manager* (UCM). The UCM, according to the Autosar Adaptive specification document (AutosarAdaptive, 2019), can also run on a dedicated ECU different from the gateway and, therefore, more protected from external attacks.

In 2016 Karthik et al. (Karthik et al., 2016) released Uptane, a Framework for software and firmware update over the air, created for securing ground vehicles.

Uptane, *optionally*, allows one to encrypt software images (i.e., software updates) using symmetric, asymmetric, or digital envelope techniques. In this chapter, we design a simple framework integrated with ABE to measure its impact on the OTA software update. Since ABE is, by all means, an asymmetric encryption scheme, we show that it is possible to integrate ABE in a real and complex framework, therefore showing that ABE is a viable solution to provide confidentiality for the IP.

In 2018 Asokan et al. (Asokan et al., 2018) proposed ASSURED, a framework for OTA firmware, based on Uptane (Karthik et al., 2016). In their work, they claim that ASSURED reaches five objectives:

1. End-to-End authentication and integrity: the update must be signed by the manufacturer and verified by the device.

2. Update Authorization from Controller: only authorized devices can install the update.

3. Attestation of update installation: the device must provide proof of the update installation.

4. Protection of Code and secret key *on device*: the update must be stored and then installed in secure storage and isolated execution of critical code.

5. Minimal burden for the device.

However, ASSURED does not consider as an adversary an external entity that eavesdrops on the communication to retrieve the update's code or that retrieves it from a tampered gateway. Instead, in our work, in addition to the objectives achieved by ASSURED, we consider such an adversary and protect it using Attribute-Based Encryption.

In 2020 Ghosal et al. (Ghosal et al., 2020) proposed STRIDE, an OTA software update scheme for autonomous vehicles. In their work, the authors provide confidentiality to the software update by using the CP-ABE scheme proposed by Bethencourt et al. (Bethencourt et al., 2007). Furthermore, they provide an extensive performance evaluation by simulation through OMNeT++ (Varga, 2010). However, they do not test the performance of the introduction of ABE on a real automotive platform, as we do in this chapter with the Xilinx ZCU102 evaluation board. This gives us a realistic estimation of the performances. Moreover, the authors do not evaluate the performance of key revocation mechanisms, which cannot be neglected as they are necessary for practical use in a real-world scenario.

Halder et al. recently published a survey (Halder et al., 2020) on secure over the air software updates in connected vehicles. The update's confidentiality is a mandatory requirement in their work, and they investigated and discussed many

schemes. Covered techniques are, for example, OTA based singularly on: symmetric key; hash functions; blockchain; RSA and steganography; HSM; secure update frameworks; and so on. However, in their work, an approach explicitly based on Attribute-Based Encryption has not been considered.

Testing Platforms and Automotive Hardware Background

To the authors' knowledge, the literature has yet to test the impact of ABE schemes on a real hardware automotive embedded platform. The main difference between traditional IoT devices (e.g., smartphones, sensors, Raspberry Pi,...) and automotive embedded platforms is that the latter feature different hardware. As the reader will see at the end of this section, automotive embedded platforms feature, among other things, multi-core processors and real-time processors, hardware that is not available in common IoT devices. Therefore, in this section, we explain the on-board network organization and the computation capability of real automotive platforms so that the proposed ABE technique is integrated into a representative automotive scenario. We reference to *emerging vehicle architectures*, describing how it is designed, to show that previous works cannot be taken into consideration when arguing about the performances of ABE in the automotive domain.

In-vehicle networks are in a transition from legacy domain-based electronic architectures to zonal architectures. Domain-based architectures with many simple and separate ECUs and networks will be used for commodity automotive subsystems (e.g., break or steer control). Instead, a small number of supercomputers are needed for high-performance tasks (e.g., sensor fusion for obstacle detection, navigation, and trajectory planning). Besides classic local interconnect and controller area networks, in emerging automotive platforms, the wireless V2X (vehicle to everything) connectivity is ensured by vehicular versions of WLAN (e.g., 802.11p technology) and of cellular networks (e.g., C-V2X). This wide range of connectivity will highly increase the opportunity of SW OTA distributed dissemination (Halder et al., 2020). However, such a wide range of connectivity solutions to the external world can be a liability since it opens the vehicle to external cyber threats. This is perceived as a severe threat by the automotive companies that tried to aggregate all the connectivity capabilities over a single ECU, called the gateway. However, for critical applications like the OTA SW/FW update, it is recommended to provide a dedicated ECU, different from the gateway, called Update and Configuration Manager (AutosarAdaptive, 2019). This ECU contains the cryptographic quantities needed for the OTA SW/FW update, such as public keys and private keys for signature verification and decryption, respectively. Indeed, the recommended OTA update process inside the vehicle looks like this: i) the gateway downloads the signed and encrypted update and forwards it to the UCM; ii) the UCM verifies the sig-

nature; iii) the UCM decrypts the update; and finally iv) the UCM forwards the decrypted update to the ECU that needs it.

A key component of this new automotive networking architecture is the availability of more powerful ECUs than before. Differently from commodity ECUs, characterized by low-cost microcontrollers, powerful automotive ECU processors typically are equipped with i) interfaces towards Ethernet physical layer and switches, ii) application processors like those of the Cortex-A family with AArch64 64-bit instruction set, iii) Hardware Security Engine (HSE) for secure boot and accelerated security services. Referring to point (iii) in particular, the UCM should be a *"security level IV"* ECU, as specified by the de-facto standard on vehicular security hardware, the *EVITA* project (Evita, 2008). In terms of intra-vehicle connectivity, the S32G chip sustains several network protocols, like Ethernet and CAN. Those characteristics allow the UCM to efficiently perform many cryptographic operations and be connected with every commodity ECUs that need support for the OTA SW/FW update.

The reader should be aware that such resourceful ECUs are not future developments, but they are already being used. The same concept of integrating multicore Cortex-A processors in automotive platforms is also followed by supercomputer platforms like the Renesas H3 heterogeneous System-on-Chip. It integrates 4 Cortex-A72 and 4 Cortex-A53 cores, supervised by a dual lock-step Cortex-R7 real-time microcontroller and a rich set of networking interfaces. To this aim, the European Processor Initiative (Kovač et al., 2020) is developing a High-Performance heterogeneous processor which integrates multiple ARM cores with AArch 64-bit architecture with SVE (Scalable Vector Extension) plus co-processor tiles for embedded FPGA, massively parallel processor array (MPPA), and RISC-V based stencil and neurostream accelerators (STX). This chapter provides performance evaluation on a *real automotive compliant board* to assess the easy integration of ABE as an additional feature. To this aim, the ZCU102 from Xilinx has been selected as a versatile prototyping platform, representative of both automotive powerful ECUs processors and scaled versions of heterogeneous supercomputers. The ZCU102 hosts a Zynq UltraScale+ MPSoC chip with quad Arm Cortex®-A53 cores with Arm Neon™ technology plus dual-core Cortex-R5F real-time processors, and a Mali™-400 MP2 graphics processing unit. The FPGA resources of the ZCU102 allow for further accelerators integration. ZCU102 also provides a rich set of connectivity interfaces. From a software development point of view, the ZCU102 sustains a Linux-like OS (PetaLinux) and an integrated design environment (VITIS) to develop both HW and SW parts. Moreover, the Xilinx Zynq UltraScale+ MPSoC platform is not only a rapid prototyping tool; but it can also be used as a *product*, being recently adopted by Continental for its 4D automotive radar (Xilinx, 2020).

8.2 Methods

To evaluate the impact that the introduction of ABE has on a vehicle's performance, we designed the following experiments. As ABE scheme for the experiments, we choose the CP-ABE by Bethencourt et al. (Bethencourt et al., 2007). Our scenario is composed of many vehicles, a manufacturer, and an honest-but-curious cloud server. The manufacturer possesses the CP-ABE master key, the CP-ABE encryption key, a pair of RSA keys, and it knows each vehicle's RSA public key. The manufacturer is in charge of generating all the cryptographic keys needed in the system. We assume that each vehicle has an ECU dedicated to the OTA update called *Update and Configuration Manager* (UCM), as specified in the Autosar specification document (AutosarAdaptive, 2019). This ECU is not connected directly to the Internet, though it is connected to the gateway and each ECU that supports the OTA update functionality. Each vehicle possesses: i) a CP-ABE decryption key, which describes the vehicle's components and characteristics; ii) a pair of RSA keys; and iii) the manufacturer's RSA public key. These vehicle-related cryptographic keys are installed in the ECU that implements the UCM (AutosarAdaptive, 2019) by the OEM at the time of its construction. The manufacturer produces the software update, encrypts it with CP-ABE, signs it –along with a version number– using RSA, and stores the signed and encrypted update on the cloud server. The cloud server sends the signed and encrypted update to any vehicle that requests it. Upon receiving the software update, the gateway forwards the message to the UCM. The UCM first verifies the manufacturer signature, then decrypts the CP-ABE ciphertext. Finally, the UCM forwards the software update to the intended ECU, which installs it as soon as the user gives his/her consent. The use case and the interactions are depicted in figure 8.1.

Furthermore, if one or many decryption keys are compromised, the manufacturer also provides new keys to the non-compromised vehicles. To do so, the manufacturer generates a new CP-ABE decryption key for each non-compromised vehicle, encrypt said key using the vehicle's RSA public key, and signs the ciphertext using its RSA private key. Then, the manufacturer stores the encrypted and signed decryption key (from now on, the key update) in the cloud server. If a new decryption key has been released for a vehicle, when such a vehicle requests a software update, the cloud server also sends to it the key update. In this case, the UCM first verifies the signature on the key update, then retrieves the new CP-ABE decryption key using RSA decryption. Finally, the UCM verifies the signature on the software update and decrypts it using the new CP-ABE decryption key. Such interactions are depicted in figure 8.2. The reader may argue that this revocation mechanism is inefficient. However, our example is a worst-case scenario: if the impact of such a naive mechanism is limited on our test platform, this means that more advanced and efficient revocation mechanisms will be well supported too.

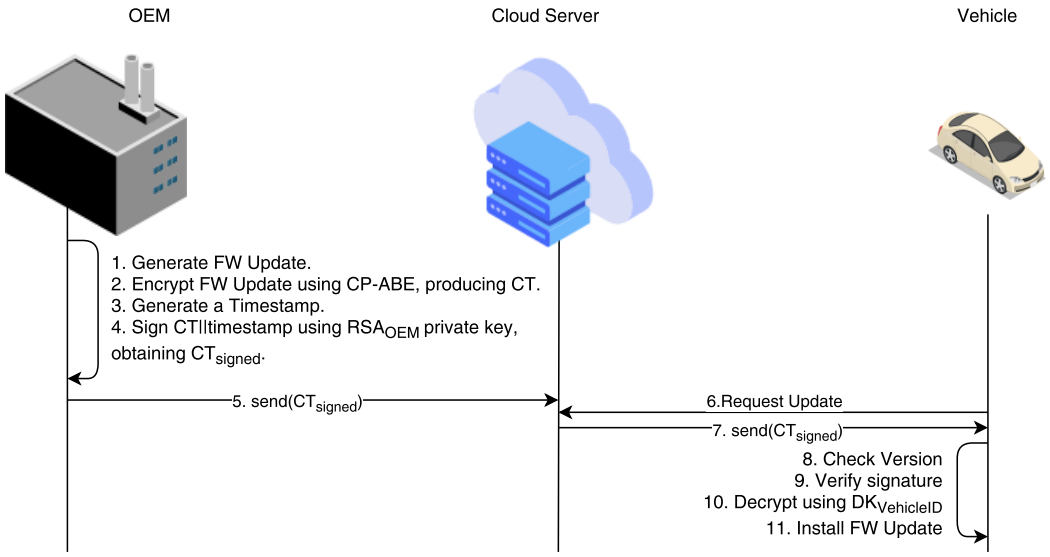


Figure 8.1. Use-case scenario of firmware over-the-air update using CP-ABE.

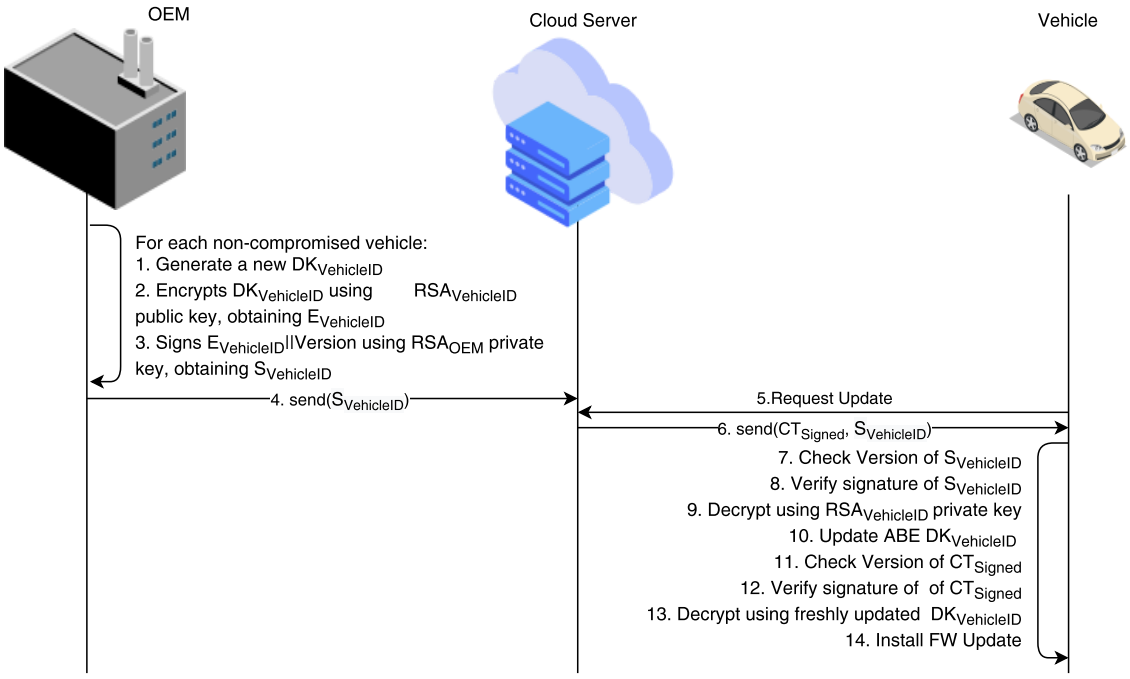


Figure 8.2. CP-ABE decryption key distribution in case of key compromise.

Attacker Model

With reference to figure 8.2, which represents the most complex scenario treated in this chapter, we now define the attacker model: its capabilities, its motivation, its objectives, and how it would like to achieve them.

We assume that the OEM, the UCM, and all the vehicle's ECUs (except the gateway) are trusted, whereas the Cloud Server and the gateway are untrusted. We think this is a reasonable assumption since the gateway inside a vehicle is the only ECU directly connected to the Internet, and therefore it is more exposed to external attacks than the other ECUs.

We now analyze the considered threats: a passive attacker and an active attacker. A passive attacker can intercept every message sent over the Internet, both between the OEM and the Cloud Server and between the Cloud Server and the vehicle. The objectives of a passive attacker are two: p_i) to capture and decrypt an ABE ciphertext, obtaining an update; and p_ii) to capture and decrypt an RSA ciphertext to retrieve an ABE decryption key. Objective (p_i) and (p_ii), however, cannot be achieved since it would mean that the attacker can break the schemes in (Bethencourt et al., 2007) and (Rivest et al., 1978) respectively.

Instead, the active attacker can gain access to and/or control of the Cloud Server and/or the gateway. This can be done by leveraging one of the many vulnerabilities that have been discovered over the years (Kocher et al., 2020, 2019; Lipp et al., 2018). For example, an active adversary can install a spyware on the gateway (or on the cloud server) so that each and every piece of information managed and manipulated by it is forwarded to the attacker.

The objectives of an active attacker are: a_i) to force a vehicle to install a malicious SW update; a_ii) to decrypt an ABE ciphertext; and a_iii) to acquire a decryption key and a private RSA key from a vehicle.

Objective (a_i) cannot be achieved without forging a valid signature applied to the SW update by the OEM, therefore breaking the RSA signature scheme. The attacker can pursue objective (a_ii) by gaining control of the cloud server or the gateway. However, the attacker cannot achieve such an objective in both cases since neither the Cloud Server nor the gateway possesses any decryption key. If in some way, the attacker retrieves a vehicle's CP-ABE decryption key and the RSA private key — achieving (a_iii) — it will be capable of decrypting any ciphertext that such ABE key complies with, and it will be capable of retrieving and decrypting the key update made for said vehicle. This attack is effective until the OEM performs a revocation for the key. When the OEM does this, it removes the associated RSA public key from its database of public keys, and it stops generating key updates for the compromised vehicle.

8.3 Performance Evaluation

In this section, we briefly explain how we recreated the scenario, showing the software and hardware we used.

Experimental Setup

We designed a client-server application that reflects the interaction between the cloud server and the vehicles. We used C as the programming language and OpenSSL, libswabe, the CP-ABE toolkit (Bethencourt et al., 2011), GMP, and the *Pairing Based Cryptography* (PBC) as libraries. The objective is to measure the time passed from the moment an update is requested to the moment it is installed. Ultimately, we show that CP-ABE has little-to-no impact on the performance while providing a fundamental feature. We investigated three different scenarios: i) NO CP-ABE; ii) only CP-ABE encryption; iii) CP-ABE encryption + key update. In the first scenario, when the vehicle requests an update, the cloud sends the update in the clear along with the associated version, all signed by the OEM. This scenario will be our reference for the CP-ABE performance evaluation. In the second scenario, the interaction between the cloud server and the vehicle is depicted in figure 8.1. The cloud server stores the SW update encrypted with CP-ABE, along with the update's version, all signed by the OEM. In the third scenario, the interaction between the cloud server and the vehicle is similar to that in the second scenario. However, every once in a while, in addition to the SW update encrypted with CP-ABE, the cloud server will send to the vehicle also a new CP-ABE decryption key, as depicted in figure 8.2. For scenarios 2 and 3, we used a single policy to encrypt the software update, and we used two different attribute sets to represent two different vehicles (Vehicle1 and Vehicle2) that can satisfy the policy. The policy and attribute sets are depicted in figure 8.3. The policy reads as: "A vehicle can access the data if and only if it has the *ECU_MODEL_2247* **OR** it is both a *CAR_MODEL_21* **AND** it has the *ECU_MODEL_2248*". The attribute set of Vehicle1 is composed of four different attributes and it reads as follow: "Vehicle1 is a *CAR_MODEL_23* and it has *ECU_MODEL_2247*, *ECU_MODEL_2256*, and *ECU_MODEL_2268*"; the attribute set of Vehicle2 is composed of three different attributes and it reads as follow: "Vehicle2 is a *CAR_MODEL_21* and it has *ECU_MODEL_2246*, and *ECU_MODEL_2248*". Vehicle1 is able to decrypt the ciphertext because it has the *ECU_MODEL_2247* attribute, while Vehicle2 is able to decrypt the ciphertext because it has both the attributes *CAR_MODEL_21* and *ECU_MODEL_2248*.

We run the client (which simulates the vehicle) on a Xilinx ZCU102 evaluation board equipped with a Zynq UltraScale+ MPSoC chip which features, as already discussed before, a quad Arm Cortex®-A53 cores with Arm Neon™ technology plus dual-core Cortex-R5F real-time processors, a Mali™-400 MP2 graphics processing

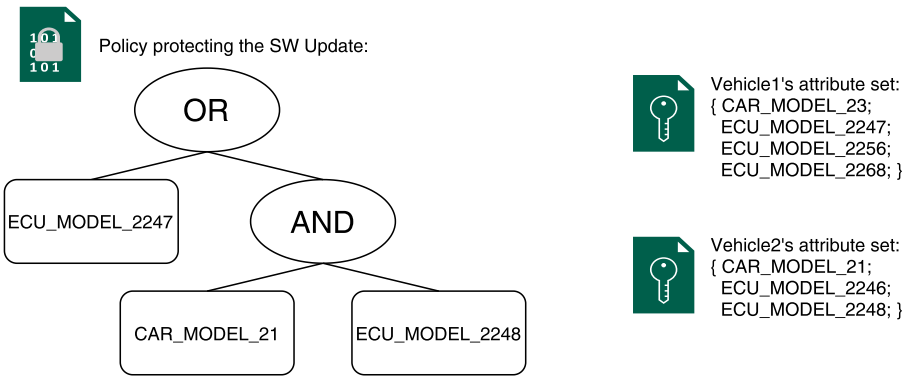


Figure 8.3. The policy and the two attribute sets used for the experiments.

unit, and four SLFP+ interfaces for Ethernet, six 16.3Gb/s GTH transceivers and 64 user-defined differential I/O signals, 600 system logic cells, 32Mb of memory, 2500 DSP slices.

For each scenario, we evaluated the performances of the ZCU102 board over 5,000 iterations, with a confidence interval of 95%. For the Elliptic Curve Cryptography operations, we used Type A internals of the PBC library with a group order of 160 bits, element size of 512 bits, and an embedding degree $k = 2$, which gives an 80-bit security level. To choose the revocation rate, we based our analysis on the frequency of updates in Tesla vehicles (Tesla, 2020b). From their websites, we can see that, from January 2020 to November 2020, 122 updates have been released, meaning that — on average — more than 11 updates are released each month. Therefore, we evaluated four different revocation rates, roughly from weekly to monthly: once every two updates, once every three updates, once every six updates, and once every twelve updates.

Results

We show in figure 8.4 the results of our experiment. The results of scenario 1's evaluation show us that the download time and the verification of the RSA signature take about 256 milliseconds. The introduction of only CP-ABE decryption in scenario two increases by 200 - 230 ms the time elapsed from the request of the update to the starting of the installation. This increase in time is due to CP-ABE decryption and the CP-ABE ciphertext overhead download. From the graph, we can see that Vehicle2 spends on average about 24 ms more than Vehicle1. Indeed, in order to decrypt the CP-ABE ciphertext, in Vehicle2 the attribute used to decrypt the policies are 2 (i.e., *CAR_MODEL_21* and *ECU_MODEL_2248*), whereas Vehicle1 only has 1 (i.e., *ECU_MODEL_2247*). Finally, in scenario 3, we see that the additional seldom retrieval of a new decryption key costs on average 90-105 ms, in case of a revocation

frequency of once every six updates, which translates to a revocation every 15 days. Moreover, figure 8.5 shows that for a wide range of revocation frequencies, the impact of CP-ABE decryption and key update is limited. Even at the higher measured frequency -once every two updates, or once every five days- the download, key update, and the decryption processes are all performed in just under 675 – 710 ms. If the revocation frequency drops to once every 12 updates (roughly once a month), the entire process takes between 518 – 535 ms.

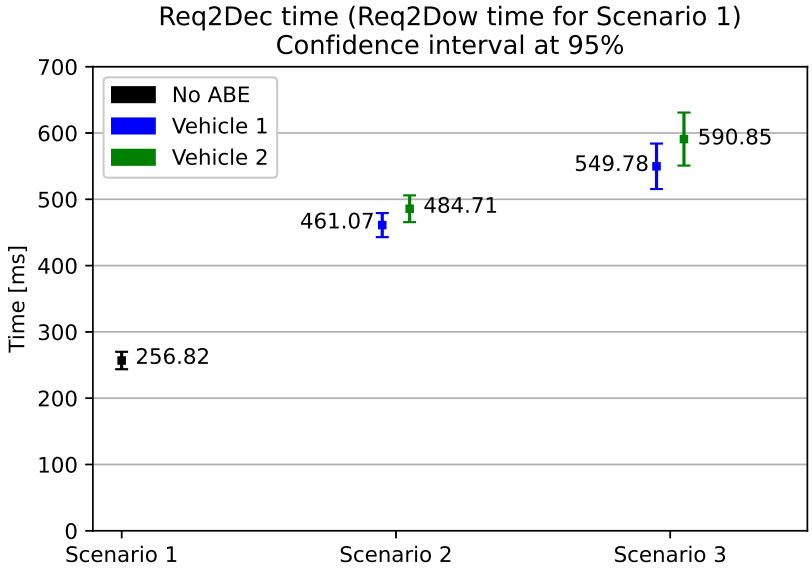


Figure 8.4. Elapsed time from the update request to the moment just before the installation. The considered revocation frequency in Scenario 3 is once every six updates.

Analyzing the time spent in scenarios 2 and 3 compared to the time spent in scenario 1, it seems that CP-ABE has a non-negligible impact on the OTA SW update process. However, when we compare these results to the time spent on installing the SW, we can see that the time increase due to CP-ABE is negligible. Indeed, we also investigated the size of a software update in the automotive scenario. We found out that, typically, an update's size for a Tesla "Model 3" is about 100MB (Tesla, 2020a). To replicate the installation process, we chose to install installation packages of different sizes on the ZCU102. Namely, we measured the installation time for programs with sizes of ~6.9 KiB, ~2.7 MiB, ~ 5.9 MiB, as shown in figure 8.6. We did not perform tests of greater SW size for two reasons: i) we had difficulties finding programs with the size around 100MB; and ii) even with such small sizes, the installation times are already orders of magnitude greater than decryption and download time. Figure 8.6 shows that the software of size ~6.9 KiB, ~2.7 MiB, ~ 5.9 MiB took, on average, 2100 ms, 12388 ms, and 22148 ms, respectively.

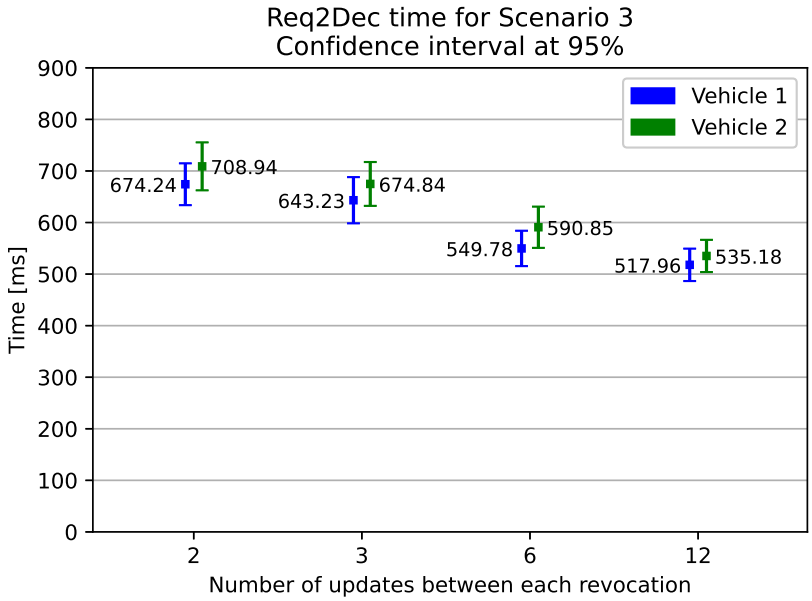


Figure 8.5. Elapsed time from the update request to the moment just before the installation in scenario 3, varying the revocation frequency.

In figure 8.7 we can see how much time it takes for each scenario from the update request to the end of the update installation. We had to use the logarithmic scale to see the CP-ABE impact since the three scenarios are practically equivalent when considering the installation time. Indeed, the average time spent on the 5.9 MiB SW update is 22148 ms with a 95% confidence interval of ±247 ms. This means that the SW installation time is two orders of magnitude greater than all the previous times computed in the three presented scenarios. Therefore, also considering that we measured the installation times on SW images that are way smaller than the ones deployed in reality, we can conclude that the time impact of CP-ABE is negligible.

Furthermore, we also considered the impact of CP-ABE in terms of message size. Figure 8.8 shows the size of the single components of the update message that the cloud sends to the vehicle. The main three fields of such a message are: i) the symmetrically encrypted SW update (5.9 MiB); ii) the CP-ABE ciphertext containing the symmetric key; iii) the RSA signature of the OEM. Compared to the RSA signature, the CP-ABE ciphertext amounts to over three times the size. However, compared to the actual software update size, the impact of the CP-ABE ciphertext is so negligible that, in order to show them both in the same graph, we have to use a logarithmic scale.

Considering that the OTA SW update operation is not a time-critical task, and considering that, in any case, the dominant time cost is the SW installation, we think that the adoption of CP-ABE also in real-life applications will be an excellent addi-

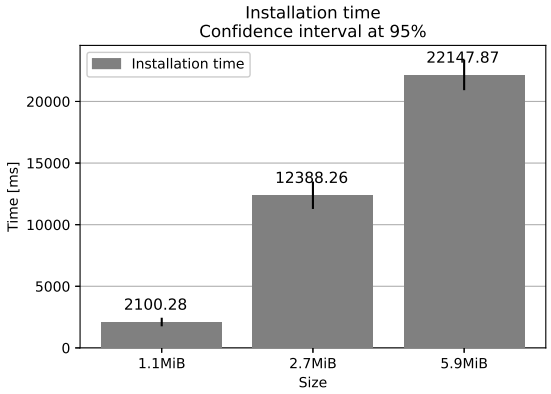


Figure 8.6. A comparison of the installation times of various SW's size.

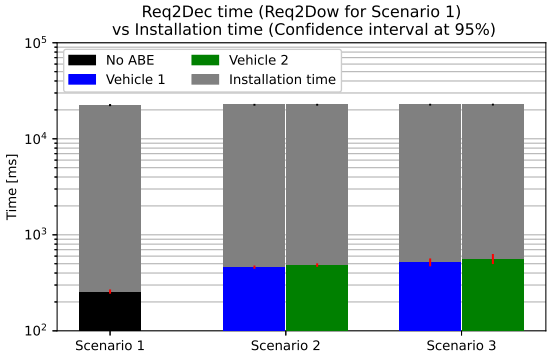


Figure 8.7. A comparison of the total time taken from the update request to the end of SW installation. The considered SW size is 5.9 MiB, and the considered revocation frequency is once every 6 updates.

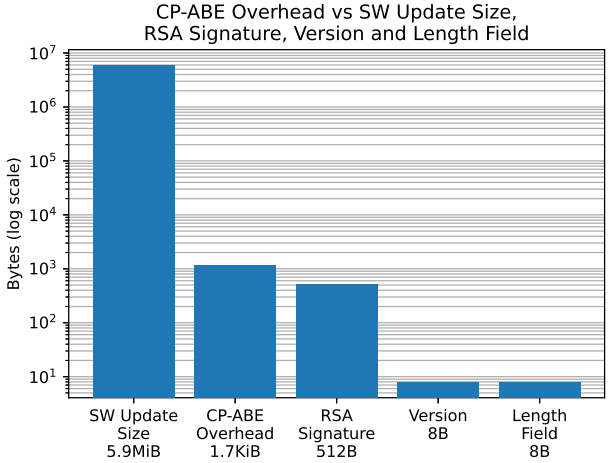


Figure 8.8. The size of each field inside the update message that the cloud sends to the vehicle.

tion to the security of our vehicles.

8.4 Answer

Is it Feasible to Leverage CP-ABE in the Automotive Environment?

Yes, and it is also convenient! With a negligible increment of the computational cost, time required, and bandwidth overhead, we showed how to provide confidentiality to the update destined to the vehicles.

Chapter 9

Conclusions

In this Ph.D. dissertation, we studied and applied the technique known as Attribute-Based Encryption in different scenarios.

At first, we presented a comparison between CP-ABE and sticky policy approaches in a smart home environment. The analysis, conducted through a prototypical implementation of the two solutions, revealed the potentialities of CP-ABE in guaranteeing a better secure-aware and efficient data flow management than approaches based on sticky policies. Concerning robustness towards different possible attacks, CP-ABE appears to be more resilient. The CP-ABE approach suits both small-scale scenarios and wide-area low-density scenarios.

Secondly, we showed fABElous, an ABE solution suitable for Industrial IoT applications that minimizes the communication overhead introduced by ABE encryption. We described its architecture, system procedures and provided a use case example. We reduced the communication overhead by 49% than using ABE techniques naively.

We also described SEA-BREW (Scalable and Efficient ABE with Broadcast REvocation for Wireless networks), an ABE revocable scheme suitable for low-bitrate Wireless Sensor and Actuator Networks (WSANs) in IoT applications. SEA-BREW is highly scalable in the number and size of messages necessary to manage decryption keys. SEA-BREW can revoke or renew multiple decryption keys by sending a single broadcast message over a WSAN. Intuitively, such a message allows all the nodes to update their keys locally. In SEA-BREW, things and users can exchange encrypted data via the cloud and directly if they belong to the same WSAN. This makes the scheme suitable for both remote cloud-based communications and local delay-bounded ones. The scheme also provides a mechanism of proxy re-encryption (Yu et al., 2010a,b; Zu et al., 2014) by which old data can be re-encrypted by the cloud to make a revoked key unusable. We formally proved that our scheme is adaptively IND-CPA secure also in case of an untrusted cloud server that colludes with a set of users, under the generic bilinear group model. We finally showed by simulations

that the computational overhead is constant on the cloud server, with respect to the complexity of the access control policies.

Then, we carried out a performance evaluation of ABE in constrained IoT devices. Specifically, we implemented two representative ABE schemes and tested their performance on ESP32 and RE-Mote, two popular IoT rapid prototyping platforms. Our performance evaluation showed that classical ABE schemes significantly impact the lifetime of battery-powered devices, especially when a high number of attributes (i.e., 20-50) is used in ciphertexts. However, if we assume to employ fewer attributes (up to 10) and leverage hardware elliptic-curve cryptographic acceleration, which is present on some platforms (e.g., RE-Mote), ABE can indeed be adopted on devices with very limited memory and computing power. We also obtained a significant yet tolerable battery lifetime reduction. In addition, we presented a novel benchmark method that allows us to evaluate the average decryption performance, i.e., time and energy consumption, instead of the worst-case performance, typically used by the literature. We exploited this method to complete our evaluation by estimating the average decryption time and energy on RE-Mote. We showed that the current literature significantly overestimates the processing time and energy by always considering the worst-case decryption.

Approaching the end, we surveyed ABE schemes and solutions suitable for IoT applications. We analyzed various schemes under six performance indicators, and we identified and described the strategies that the state-of-the-art schemes adopt to improve such performance indicators. Moreover, we assessed the efficiency of some prominent ABE schemes by thorough simulations.

Finally, we changed the use-case and showed that the Attribute-Based Encryption technique improves security for over-the-air update functionalities in an automotive scenario. Particularly, ABE provides confidentiality to the software/firmware update done Over The Air and also for data at rest. Furthermore, we tested a naive key revocation mechanism, another missing feature in state-of-the-art systems. We showed that ABE can seamlessly be integrated into the existing solutions regarding the OTA update, and more broadly, it complies with automotive standards in terms of architecture and documentations. Furthermore, the overhead of the ABE integration in terms of computation time and storage is negligible w.r.t. the other tasks involved in an OTA software update, like the installation. These results show that security can be enhanced at a minimum cost.

Appendix A

Publications

This appendix contains a list of publications in which the candidate contributed as an author. The conrtibution given for each article is specified according to the Contributor Roles Taxonomy (CRediT)[1], which is reported in Table A.1 for convenience.

Journal papers

1. S. Sicari, A. Rizzardi, G. Dini, P. Perazzo, **M. La Manna**, A. Coen-Porisini, "Attribute-based encryption and sticky policies for data access control in a smart home scenario: A comparison on networked smart object middleware", *International Journal of Information Security*, pages 695-713, 2021. **Candidate's contributions**: Conceptualization, Formal Analysis, Methodology, Software, Visualization, Writing - original draft, Writing - review & editing.

2. **M. La Manna**, L. Treccozzi, P. Perazzo, S. Saponara, G. Dini, "Performance evaluation of attribute-based encryption in automotive embedded platform for secure software over-the-air update", *MDPI Sensors*, page 515, 2021. **Candidate's contributions**: Conceptualization, Formal Analysis, Methodology, Supervision, Visualization, Writing - original draft, Writing - review & editing.

3. P. Perazzo, F. Righetti, **M. La Manna**, C. Vallati, "Performance evaluation of attribute-based encryption on constrained IoT devices", *Computer Communications*, pages 151-163, 2021. **Candidate's contributions**: Conceptualization, Formal Analysis, Investigations, Methodology, Software, Writing - original draft, Writing - review & editing.

4. **M. La Manna**, P. Perazzo, G. Dini, "SEA-BREW: A scalable Attribute-Based Encryption revocable scheme for low-bitrate IoT wireless networks", *Journal*

[1] www.casrai.org/credit.html

of Information Security and Applications, page 102692, 2021. **Candidate's contributions**: Conceptualization, Formal Analysis, Investigation, Methodology, Software, Visualization, Writing - original draft, Writing - review & editing.

Workshop papers

1. **M. La Manna**, P. Perazzo, M. Rasori, G. Dini, "fABElous: An attribute-based scheme for industrial internet of things", *3rd IEEE International Workshop on Big Data and IoT Security in Smart Computing (IEEE BITS 2019), part of IEEE SMARTCOMP 2019*, 2019. **Candidate's contributions**: Conceptualization, Formal Analysis, Investigations, Methodology, Software, Visualization, Writing - original draft.

2. **M. La Manna**, L. Treccozzi, P. Perazzo, G. Dini, "Assessing the Cost of Quantum Security for Automotive Over-The-Air Updates", *11th Workshop on Management of Cloud and Smart City Systems (MoCS 2021), part of IEEE ISCC 2021*, 2021. **Candidate's contributions**: Conceptualization, Formal Analysis, Methodology, Supervision, Visualization, Writing - original draft, Writing - review & editing.

Accepted Papers Waiting for Publication

1. M. Rasori, P. Perazzo, **M. La Manna**, G. Dini, "A Survey on Attribute-Based Encryption Schemes Suitable for the Internet of Things", *IEEE Internet of Things Journal*. **Candidate's contributions**: Conceptualization, Formal Analysis, Investigations, Methodology, Software, Writing - original draft, Writing - review & editing.

Table A.1
Contributor Roles Taxonomy (CRediT) Table.

Role	Definition
Conceptualization	Ideas; formulation or evolution of overarching research goals and aims.
Data Curation	Management activities to annotate (produce metadata), scrub data and maintain research data (including software code, where it is necessary for interpreting the data itself) for initial use and later re-use.
Formal Analysis	Application of statistical, mathematical, computational, or other formal techniques to analyze or synthesize study data.
Funding Acquisition	Acquisition of the financial support for the project leading to this publication.
Investigations	Conducting a research and investigation process, specifically performing the experiments, or data/evidence collection.
Methodology	Development or design of methodology; creation of models.
Project administration	Management and coordination responsibility for the research activity planning and execution.
Resources	Provision of study materials, reagents, materials, patients, laboratory samples, animals, instrumentation, computing resources, or other analysis tools.
Software	Programming, software development; designing computer programs; implementation of the computer code and supporting algorithms; testing of existing code components.
Supervision	Oversight and leadership responsibility for the research activity planning and execution, including mentorship external to the core team.
Validation	Verification, whether as a part of the activity or separate, of the overall replication/reproducibility of results/experiments and other research outputs.
Visualization	Preparation, creation and/or presentation of the published work, specifically visualization/data presentation.
Writing - original draft	Preparation, creation and/or presentation of the published work, specifically writing the initial draft (including substantive translation).
Writing - review & editing	Preparation, creation and/or presentation of the published work by those from the original research group, specifically critical review, commentary or revision – including pre- or post-publication stages.

Bibliography

Agrawal, S. and Chase, M. (2017). FAME: Fast attribute-based message encryption. Cryptology ePrint Archive, Report 2017/807. `https://eprint.iacr.org/2017/807`.

Akinyele, J. A., Garman, C., and Hohenberger, S. (2015). Automating fast and secure translations from type-i to type-iii pairing schemes. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1370–1381, New York, NY, USA. Association for Computing Machinery.

Al-Dahhan, R. R., Shi, Q., Lee, G. M., and Kifayat, K. (2019). Survey on revocation in ciphertext-policy attribute-based encryption. *Sensors*, 19(7):1695.

Ambrosin, M., Anzanpour, A., Conti, M., Dargahi, T., Moosavi, S. R., Rahmani, A. M., and Liljeberg, P. (2016). On the feasibility of attribute-based encryption on internet of things devices. *IEEE Micro*, 36(6):25–35.

Ambrosin, M., Conti, M., and Dargahi, T. (2015). On the feasibility of attribute-based encryption on smartphone devices. In *Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems*, pages 49–54.

Aptiv (2020). *What Is Over-the-Air (OTA)?* `www.aptiv.com/newsroom/article/what-is-over-the-air-(ota)`.

Arena, A., Perazzo, P., and Dini, G. (2019). Virtual private ledgers: Embedding private distributed ledgers over a public blockchain by cryptography. In *Proceedings of the 23rd International Database Applications & Engineering Symposium*, IDEAS '19, New York, NY, USA. Association for Computing Machinery.

Ashton, K. et al. (2009). That 'internet of things' thing. *RFID journal*, 22(7):97–114.

Asokan, N., Nyman, T., Rattanavipanon, N., Sadeghi, A.-R., and Tsudik, G. (2018). Assured: Architecture for secure software update of realistic embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2290–2300.

Attrapadung, N., Hanaoka, G., Ogawa, K., Ohtake, G., Watanabe, H., and Yamada, S. (2016). Attribute-based encryption for range attributes. In *International Conference on Security and Cryptography for Networks*, pages 42–61. Springer.

Attrapadung, N. and Imai, H. (2009). Attribute-based encryption supporting direct/indirect revocation modes. In Parker, M. G., editor, *Cryptography and Coding*, pages 278–300, Berlin, Heidelberg. Springer Berlin Heidelberg.

Attrapadung, N., Libert, B., and de Panafieu, E. (2011). Expressive key-policy attribute-based encryption with constant-size ciphertexts. In Catalano, D., Fazio, N., Gennaro, R., and Nicolosi, A., editors, *Public Key Cryptography – PKC 2011*, pages 90–108, Berlin, Heidelberg. Springer Berlin Heidelberg.

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15):2787–2805.

AutosarAdaptive (2019). *Specification of Update and Configuration Management*.

Baccelli, E., Cragie, R., Der Stok, P., and Brandt, A. (2016). Applicability Statement: The Use of the Routing Protocol for Low-Power and Lossy Networks (RPL) Protocol Suite in Home Automation and Building Control. RFC 7733, RFC Editor.

Bagci, I., Raza, S., Chung, T., Roedig, U., and Voigt, T. (2013). Combined secure storage and communication for the Internet of Things. In *2013 IEEE International Conference on Sensing, Communications and Networking, SECON 2013*, pages 523–631, New Orleans, LA, United States.

Barker, S., Mishra, A., Irwin, D., Cecchet, E., Shenoy, P., and Albrecht, J. (2012). Smart*: An open data set and tools for enabling research in sustainable homes. *SustKDD, August*, 111:112.

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

Bethencourt, J., Sahai, A., and Waters, B. (2011). The cpabe toolkit.

Boldyreva, A., Goyal, V., and Kumar, V. (2008). Identity-based encryption with efficient revocation. In Ning, P., Syverson, P. F., and Jha, S., editors, *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 417–426. ACM.

Boneh, D. and Franklin, M. (2001). Identity-based encryption from the Weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer.

Boneh, D., Gentry, C., and Waters, B. (2005). Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Annual International Cryptology Conference*, pages 258–275. Springer.

Boswarthick, D., Elloumi, O., and Hersent, O. (2012). *M2M Communications: A Systems Approach*. Wiley Publishing, 1st edition.

Chen, F., Talanis, T., German, R., and Dressler, F. (2009). Real-time enabled IEEE 802.15.4 sensor networks in industrial automation. In *Industrial Embedded Systems, 2009. SIES'09. IEEE International Symposium on*, pages 136–139. IEEE.

Chen, J., Gay, R., and Wee, H. (2015). Improved dual system ABE in prime-order groups via predicate encodings. Cryptology ePrint Archive, Report 2015/409. `https://eprint.iacr.org/2015/409`.

Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., and Vercauteren, F. (2005). *Handbook of elliptic and hyperelliptic curve cryptography*. Chapman and Hall/CRC.

Conzon, D., Bolognesi, T., Brizzi, P., Lotito, A., Tomasi, R., and Spirito, M. (2012). The VIRTUS middleware: An XMPP based architecture for secure IoT communications. In *2012 21st International Conference on Computer Communications and Networks, ICCCN 2012*, pages 1–6, Munich, Germany.

CSIS (2021). *Significant Cyber Incidents since 2006*. `https://www.csis.org/programs/strategic-technologies-program/significant-cyber-incidents`.

Cui, H., Deng, R. H., Li, Y., and Qin, B. (2016). Server-aided revocable attribute-based encryption. In *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II*, volume 9879 of *Lecture Notes in Computer Science*, pages 570–587. Springer.

Dai, W., Doröz, Y., Polyakov, Y., Rohloff, K., Sajjadpour, H., Savas, E., and Sunar, B. (2018). Implementation and evaluation of a lattice-based key-policy ABE scheme. *IEEE Trans. Inf. Forensics Secur.*, 13(5):1169–1184.

Di Vimercati, S. D. C., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P. (2007). Over-encryption: management of access control evolution on outsourced data. In *Proceedings of the 33rd international conference on Very large data bases*, pages 123–134. VLDB endowment.

Doshi, N. and Jinwala, D. C. (2014). Fully secure ciphertext policy attribute-based encryption with constant length ciphertext and faster decryption. *Security and Communication Networks*, 7(11):1988–2002.

Edemacu, K., Park, H. K., Jang, B., and Kim, J. W. (2019). Privacy provision in collaborative ehealth with attribute-based encryption: Survey, challenges and future directions. *IEEE Access*, 7:89614–89636.

Emura, K., Miyaji, A., Nomura, A., Omote, K., and Soshi, M. (2009). A ciphertext-policy attribute-based encryption scheme with constant ciphertext length. In *International Conference on Information Security Practice and Experience*, pages 13–23. Springer.

Espressif (2020). Espressif ESP32 datasheet. `https://bit.ly/2qW8yj1`. Accessed: 2020-01-15.

Evita (2008). *Vehicular Security Hardware*. `https://www.evita-project.org/Publications/Wolf08.pdf`.

Farrell, S. (2018). Low-power wide area network (LPWAN) overview. RFC 8376.

Fujisaki, E. and Okamoto, T. (1999). Secure integration of asymmetric and symmetric encryption schemes. In Wiener, M., editor, *Advances in Cryptology — CRYPTO' 99*, pages 537–554, Berlin, Heidelberg. Springer Berlin Heidelberg.

Galbraith, S. D., Paterson, K. G., and Smart, N. P. (2008). Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121.

Ge, A., Zhang, R., Chen, C., Ma, C., and Zhang, Z. (2012). Threshold ciphertext policy attribute-based encryption with constant size ciphertexts. In *Australasian Conference on Information Security and Privacy*, pages 336–349. Springer.

Georgiou, O. and Raza, U. (2017). Low power wide area network analysis: Can LoRa scale? *IEEE Wireless Communications Letters*, 6(2):162–165.

Ghosal, A., Halder, S., and Conti, M. (2020). Stride: Scalable and secure over-the-air software update scheme for autonomous vehicles. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.

Gilchrist, A. (2016). *Industry 4.0: the industrial Internet of Things*. Apress.

Girgenti, B., Perazzo, P., Vallati, C., Righetti, F., Dini, G., and Anastasi, G. (2019). On the feasibility of attribute-based encryption on constrained iot devices for smart systems. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 225–232. IEEE.

Gòmez-Goiri, A., Orduna, P., Diego, J., and de Ipina, D. L. (2014). Otsopack: Lightweight semantic framework for interoperable ambient intelligence applications. *Computers in Human Behavior*, 30:460–467.

Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006a). Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. ACM.

Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006b). Attribute-based encryption for fine-grained access control of encrypted data. Cryptology ePrint Archive, Report 2006/309. `https://eprint.iacr.org/2006/309`.

Halder, S., Ghosal, A., and Conti, M. (2020). Secure over-the-air software updates in connected vehicles: A survey. *Computer Networks*, page 107343.

Hankerson, D., Menezes, A. J., and Vanstone, S. (2006). *Guide to elliptic curve cryptography*. Springer Science & Business Media.

Hernández-Ramos, J. L., Pérez, S., Hennebert, C., Bernabé, J. B., Denis, B., Macabies, A., and Skarmeta, A. F. (2018). Protecting personal data in iot platform scenarios through encryption-based selective disclosure. *Computer Communications*, 130:20–37.

Herranz, J. (2017). Attribute-based encryption implies identity-based encryption. *IET information security*, 11(6):332–337.

Herranz, J. (2020). Attacking pairing-free attribute-based encryption schemes. *IEEE Access*, 8:222226–222232.

Herranz, J., Laguillaumie, F., and Ràfols, C. (2010). Constant size ciphertexts in threshold attribute-based encryption. In *International Workshop on Public Key Cryptography*, pages 19–34. Springer.

Hohenberger, S. and Waters, B. (2014). Online/offline attribute-based encryption. Cryptology ePrint Archive, Report 2014/021. `https://eprint.iacr.org/2014/021`.

Hu, V. C., Kuhn, D. R., Ferraiolo, D. F., and Voas, J. (2015). Attribute-based access control. *Computer*, 48(2):85–88.

Huang, Q., Wang, L., and Yang, Y. (2018). DECENT: Secure and fine-grained data access control with policy updating for constrained IoT devices. *World Wide Web*, 21(1):151–167.

Hunkeler, U., Truong, H. L., and Stanford-Clark, A. (2008). MQTT-S — a publish/-subscribe protocol for wireless sensor networks. In *3rd International Conference on Communication Systems Software and Middleware*.

Hur, J. (2013). Improving security and efficiency in attribute-based data sharing. *IEEE transactions on knowledge and data engineering*, 25(10):2271–2282.

Hur, J. and Noh, D. K. (2011). Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE Trans. Parallel Distributed Syst.*, 22(7):1214–1221.

Jahid, S., Mittal, P., and Borisov, N. (2011). EASiER: Encryption-based access control in social networks with efficient revocation. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 411–415. ACM.

Jiang, Y., Susilo, W., Mu, Y., and Guo, F. (2018). Ciphertext-policy attribute-based encryption supporting access policy update and its extension with preserved attributes. *International Journal of Information Security*, 17(5):533–548.

Joux, A. (2000). A one round protocol for tripartite diffie-hellman. In Bosma, W., editor, *Algorithmic Number Theory, 4th International Symposium, ANTS-IV, Leiden, The Netherlands, July 2-7, 2000, Proceedings*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer.

Karjoth, G., Schunter, M., and Waidner, M. (2002). Privacy-enabled services for enterprises. In *Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop on*, pages 483–487. IEEE.

Karthik, T., Brown, A., Awwad, S., McCoy, D., Bielawski, R., Mott, C., Lauzon, S., Weimerskirch, A., and Cappos, J. (2016). Uptane: Securing software updates for automobiles. In *International Conference on Embedded Security in Car*, pages 1–11.

Kerry, C. F. (2013). FIPS PUB 186-4 digital signature standard (DSS).

Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., et al. (2019). Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19. IEEE.

Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. (2020). Spectre attacks: exploiting speculative execution. *Commun. ACM*, 63(7):93–101.

Koster, M., Keränen, A., and Jimenez, J. (2019). Publish-subscribe broker for the constrained application protocol (CoAP). Internet-Draft draft-ietf-core-coap-pubsub-06, Internet Engineering Task Force. Work in Progress.

Kovač, M., Notton, P., Hofman, D., and Knezović, J. (2020). How europe is preparing its core solution for exascale machines and a global, sovereign, advanced computing platform. *Mathematical and Computational Applications*, 25(3):46.

Kuehner, H. and Hartenstein, H. (2016). Decentralized secure data sharing with attribute-based encryption: A resource consumption analysis. In *4th ACM International Workshop on Security in Cloud Computing*.

La Manna, M., Perazzo, P., and Dini, G. (2021). SEA-BREW: A scalable attribute-based encryption revocable scheme for low-bitrate iot wireless networks. *Journal of Information Security and Applications*, 58:102692.

La Manna, M., Perazzo, P., Rasori, M., and Dini, G. (2019). fABElous: An attribute-based scheme for industrial internet of things. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 33–38. IEEE.

Latré, B., De Mil, P., Moerman, I., Van Dierdonck, N., Dhoedt, B., and Demeester, P. (2005). Maximum throughput and minimum delay in ieee 802.15. 4. In *International Conference on Mobile Ad-Hoc and Sensor Networks*, pages 866–876. Springer.

Lee, C.-C., Chung, P.-S., and Hwang, M.-S. (2013). A survey on attribute-based encryption schemes of access control in cloud environments. *IJ Network Security*, 15(4):231–240.

Lewko, A., Okamoto, T., Sahai, A., Takashima, K., and Waters, B. (2010). Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 62–91. Springer.

Lewko, A. and Waters, B. (2011). Decentralizing attribute-based encryption. In Paterson, K. G., editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 568–588, Berlin, Heidelberg. Springer Berlin Heidelberg.

Li, J., Ma, C., and Zhang, K. (2019). A novel lattice-based ciphertext-policy attribute-based proxy re-encryption for cloud sharing. In Meng, W. and Furnell, S., editors, *Security and Privacy in Social Networks and Big Data*, pages 32–46, Singapore. Springer Singapore.

Li, J., Yao, W., Han, J., Zhang, Y., and Shen, J. (2017). User collusion avoidance CP-ABE with efficient attribute revocation for cloud storage. *IEEE Systems Journal*, 12(2):1767–1777.

Li, X., Gu, D., Ren, Y., Ding, N., and Yuan, K. (2012). Efficient ciphertext-policy attribute based encryption with hidden policy. In *International Conference on Internet and Distributed Computing Systems*, pages 146–159. Springer.

Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M. (2018). Meltdown. *arXiv preprint arXiv:1801.01207*.

Liu, C. H., Yang, B., and Liu, T. (2013a). Efficient naming, addressing and profile services in Internet-of-Things sensory environments. *Ad Hoc Networks*, 18(0):85–101.

Liu, C.-W., Hsien, W.-F., Yang, C. C., and Hwang, M.-S. (2016). A survey of attribute-based access control with user revocation in cloud data storage. *IJ Network Security*, 18(5):900–916.

Liu, J. K., Yuen, T. H., Zhang, P., and Liang, K. (2018). Time-based direct revocable ciphertext-policy attribute-based encryption with short revocation list. In *International Conference on Applied Cryptography and Network Security*, pages 516–534. Springer.

Liu, Z., Cao, Z., and Wong, D. S. (2010). Efficient generation of linear secret sharing scheme matrices from threshold access trees. *Cryptology ePrint Archive: Listing*.

Liu, Z., Cao, Z., and Wong, D. S. (2013b). White-box traceable ciphertext-policy attribute-based encryption supporting any monotone access structures. *IEEE Transactions on Information Forensics and Security*, 8(1):76–88.

Mainetti, L., Patrono, L., and Vilei, A. (2011). Evolution of wireless sensor networks towards the internet of things: A survey. In *Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on*, pages 1–6. IEEE.

McConnell, S. (2009). Code complete: A practical handbook of software construction 2nd edition. redmond.

Menezes, A., Sarkar, P., and Singh, S. (2016). Challenges with assessing the impact of nfs advances on the security of pairing-based cryptography. In *International Conference on Cryptology in Malaysia*, pages 83–108. Springer.

Ming, Y., Fan, L., Jing-Li, H., and Zhao-Li, W. (2011). An efficient attribute based encryption scheme with revocation for outsourced data sharing control. In *Instrumentation, Measurement, Computer, Communication and Control, 2011 First International Conference on*, pages 516–520. IEEE.

Moffat, S., Hammoudeh, M., and Hegarty, R. (2017). A survey on ciphertext-policy attribute-based encryption (CP-ABE) approaches to data security on mobile devices and its application to iot. In *Proceedings of the International Conference on Future Networks and Distributed Systems*.

Montenegro, G., Kushalnagar, N., Hui, J., and Culler, D. (2007). Transmission of IPv6 packets over IEEE 802.15.4 networks. RFC 4944.

NXP (2018). *Whitepaper NXP*. `https://www.nxp.com/docs/en/white-paper/AUTOGWDEVWPUS.pdf`.

Oberko, P. S. K., Obeng, V.-H. K. S., and Xiong, H. (2021). A survey on multi-authority and decentralized attribute-based encryption. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–19.

Odelu, V. and Das, A. K. (2016). Design of a new CP-ABE with constant-size secret keys for lightweight devices using elliptic curve cryptography. *Security and Communication Networks*, 9(17):4048–4059.

Odelu, V., Das, A. K., Khan, M. K., Choo, K.-K. R., and Jo, M. (2017). Expressive CP-ABE scheme for mobile devices in IoT satisfying constant-size keys and ciphertexts. *IEEE Access*, 5:3273–3283.

Ostrovsky, R., Sahai, A., and Waters, B. (2007). Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 195–203. ACM.

OWASP (2021). *OWASP Project Top 10*. `https://owasp.org/www-project-top-ten/`.

Palattella, M., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L., Boggia, G., and Dohler, M. (2013). Standardized protocol stack for the Internet of (important) Things. *Communications Surveys Tutorials, IEEE*, 15(3):1389–1406.

Pang, L., Yang, J., and Jiang, Z. (2014). A survey of research progress and development tendency of attribute-based encryption. *The Scientific World Journal*, 2014.

Pearson, S. and Mont, M. C. (2011). Sticky policies: An approach for managing privacy across multiple parties. *Computer*, 44(9):60–68.

Phuong, T. V. X., Yang, G., and Susilo, W. (2014). Poster: Efficient ciphertext policy attribute based encryption under decisional linear assumption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1490–1492. ACM.

Phuong, T. V. X., Yang, G., Susilo, W., and Chen, X. (2015). Attribute based broadcast encryption with short ciphertext and decryption key. In Pernul, G., Y A Ryan, P., and Weippl, E., editors, *Computer Security – ESORICS 2015*, pages 252–269, Cham. Springer International Publishing.

Picazo-Sanchez, P., Tapiador, J. E., Peris-Lopez, P., and Suarez-Tangil, G. (2014). Secure publish-subscribe protocols for heterogeneous medical wireless body area networks. *Sensors*, 14(12):22619–22642.

Qiao, Z., Liang, S., Davis, S., and Jiang, H. (2014). Survey of attribute based encryption. In *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 1–6. IEEE.

Qin, B., Zhao, Q., Zheng, D., and Cui, H. (2017). Server-aided revocable attribute-based encryption resilient to decryption key exposure. In *International Conference on Cryptology and Network Security*, pages 504–514. Springer.

Rasori, M., Perazzo, P., and Dini, G. (2018). Abe-cities: an attribute-based encryption system for smart cities. In *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 65–72. IEEE.

Rasori, M., Perazzo, P., and Dini, G. (2020). A lightweight and scalable attribute-based encryption system for smart cities. *Computer Communications*, 149:78 – 89.

Rasori, M., Perazzo, P., Dini, G., and Yu, S. (2021). Indirect revocable kp-abe with revocation undoing resistance. *To appear in IEEE Transactions on Services Computing*.

Reidy, K. M. (2018). Complex cybersecurity vulnerabilities: Lessons learned from Spectre and Meltdown. `http://bit.ly/30prfJ8`. Accessed: 2020-01-16.

Rescorla, E. and Modadugu, N. (2012). Datagram transport layer security version 1.2. *RFC*, 6347:1–32.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Rizzardi, A., Sicari, S., Miorandi, D., and Coen-Porisini, A. (2016). AUPS: An open source AUthenticated Publish/Subscribe system for the Internet of Things. *Information Systems*, 62:29–41.

Rouselakis, Y. and Waters, B. (2013). Practical constructions and new proof methods for large universe attribute-based encryption. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 463–474. ACM.

Sahai, A., Seyalioglu, H., and Waters, B. (2012). Dynamic credentials and ciphertext delegation for attribute-based encryption. Cryptology ePrint Archive, Report 2012/437. `https://eprint.iacr.org/2012/437`.

Sahai, A. and Waters, B. (2005). Fuzzy identity-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 457–473. Springer.

Salman, A., Diehl, W., and Kaps, J. (2017). A light-weight hardware/software co-design for pairing-based cryptography with low power and energy consumption. In *2017 International Conference on Field Programmable Technology (ICFPT)*.

Seo, J. H. and Emura, K. (2013). Revocable identity-based encryption revisited: Security model and construction. In *International Workshop on Public Key Cryptography*, pages 216–234. Springer.

Shamir, A. (1985). Identity-based cryptosystems and signature schemes. In Blakley, G. R. and Chaum, D., editors, *Advances in Cryptology*, pages 47–53, Berlin, Heidelberg. Springer Berlin Heidelberg.

Shelby, Z., Hartke, K., and Bormann, C. (2014). The constrained application protocol (CoAP). Technical report.

Sicari, S., Rizzardi, A., Grieco, L. A., and Coen-Porisini, A. (2015). Security, privacy and trust in internet of things: The road ahead. *Computer networks*, 76:146–164.

Sicari, S., Rizzardi, A., Miorandi, D., Cappiello, C., and Coen-Porisini, A. (2016a). A secure and quality-aware prototypical architecture for the Internet of Things. *Information Systems*, 58:43–55.

Sicari, S., Rizzardi, A., Miorandi, D., Cappiello, C., and Coen-Porisini, A. (2016b). Security policy enforcement for networked smart objects. *Computer Networks*, 108:133–147.

Sicari, S., Rizzardi, A., Miorandi, D., and Coen-Porisini, A. (2017a). Dynamic policies in Internet of Things: Enforcement and synchronization. *IEEE Internet of Things Journal*, 4:2228–2238.

Sicari, S., Rizzardi, A., Miorandi, D., and Coen-Porisini, A. (2017b). Security towards the edge: Sticky policy enforcement for networked smart objects. *Information Systems*, 71:78–89.

Singh, M., Rajan, M., Shivraj, V., and Balamuralidhar, P. (2015). Secure MQTT for internet of things (IoT). In *Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on*, pages 746–751. IEEE.

Tan, S., Yeow, K., and Hwang, S. O. (2019). Enhancement of a lightweight attribute-based encryption scheme for the internet of things. *IEEE Internet Things J.*, 6(4):6384–6395.

Tesla (2020a). *Tesla Average SW Update Size*. `https://forums.tesla.com/discussion/134348/size-of-software-updates`.

Tesla (2020b). *Tesla Update Timeline*. `https://teslascope.com/teslapedia/software/timeline`.

Tosi, J., Taffoni, F., Santacatterina, M., Sannino, R., and Formica, D. (2017). Performance evaluation of bluetooth low energy: A systematic review. *Sensors*, 17(12):2898.

Touati, L. and Challal, Y. (2015). Batch-based CP-ABE with attribute revocation mechanism for the Internet of Things. In *Computing, Networking and Communications (ICNC), 2015 International Conference on*, pages 1044–1049. IEEE.

Touati, L. and Challal, Y. (2016). Collaborative KP-ABE for cloud-based internet of things applications. In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–7. IEEE.

Touati, L., Challal, Y., and Bouabdallah, A. (2014). C-CP-ABE: Cooperative ciphertext policy attribute-based encryption for the internet of things. In *Advanced Networking Distributed Systems and Applications (INDS), 2014 International Conference on*, pages 64–69. IEEE.

Union, E. (2019). *European Processor Initiative*. `www.european-processor-initiative.eu/`.

Varga, A. (2010). Omnet++. In *Modeling and tools for network simulation*, pages 35–59. Springer.

Vector (2020). *Vector Automotive OTA solution*. `www.vector.com/int/en/products/products-a-z/software/vconnect`.

Wang, X., Zhang, J., Schooler, E. M., and Ion, M. (2014). Performance evaluation of attribute-based encryption: Toward data privacy in the IoT. In *Communications (ICC), 2014 IEEE International Conference on*, pages 725–730. IEEE.

Waters, B. (2011). Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *International Workshop on Public Key Cryptography*, pages 53–70. Springer.

Xilinx (2020). *Xilinx and Continental Collaborate to Create Auto Industry's First Production-Ready 4D Imaging Radar for Autonomous Driving*. `www.xilinx.com/news/press/2020`.

Xu, S., Yang, G., and Mu, Y. (2019). Revocable attribute-based encryption with decryption key exposure resistance and ciphertext delegation. *Information Sciences*, 479:116–134.

Xu, Z. and Martin, K. M. (2012). Dynamic user revocation and key refreshing for attribute-based encryption in cloud storage. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 844–849. IEEE.

Yao, X., Chen, Z., and Tian, Y. (2015). A lightweight attribute-based encryption scheme for the internet of things. *Future Generation Computer Systems*, 49:104–112.

Yi, S., Li, C., and Li, Q. (2015). A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, pages 37–42. ACM.

Yu, S., Ren, K., and Lou, W. (2011). FDAC: Toward fine-grained distributed data access control in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 22(4):673–686.

Yu, S., Wang, C., Ren, K., and Lou, W. (2010a). Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Infocom, 2010 proceedings IEEE*, pages 1–9. IEEE.

Yu, S., Wang, C., Ren, K., and Lou, W. (2010b). Attribute based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 261–270. ACM.

Zhang, Y., Chen, X., Li, J., Li, H., and Li, F. (2014a). Attribute-based data sharing with flexible and direct revocation in cloud computing. *KSII Transactions on Internet & Information Systems*, 8(11).

Zhang, Y., Deng, R. H., Xu, S., Sun, J., Li, Q., and Zheng, D. (2020). Attribute-based encryption for cloud computing access control: A survey. *ACM Computing Surveys (CSUR)*, 53(4):1–41.

Zhang, Y., Zheng, D., Chen, X., Li, J., and Li, H. (2014b). Computationally efficient ciphertext-policy attribute-based encryption with constant-size ciphertexts. In *International Conference on Provable Security*, pages 259–273. Springer.

Zhou, Z. and Huang, D. (2010). On efficient ciphertext-policy attribute based encryption and broadcast encryption. Cryptology ePrint Archive, Report 2010/395. `https://eprint.iacr.org/2010/395`.

Zhou, Z., Huang, D., and Wang, Z. (2013). Efficient privacy-preserving ciphertext-policy attribute based-encryption and broadcast encryption. *IEEE Transactions on Computers*, 64(1):126–138.

Zickau, S., Thatmann, D., Butyrtschik, A., Denisow, I., and Küpper, A. (2016). Applied attribute-based encryption schemes. In *19th International ICIN Conference-Innovations in Clouds, Internet and Networks-March*, pages 1–3.

Zolertia (2020a). Zolertia re-mote. `https://github.com/Zolertia/Resources/wiki/RE-Mote`. Accessed: 2020-01-15.

Zolertia (2020b). Zolertia RE-Mote datasheet. `https://bit.ly/2OkilYY`. Accessed: 2020-01-15.

Zolertia S.L. (2017). Zolertia re-mote platform.

Zu, L., Liu, Z., and Li, J. (2014). New ciphertext-policy attribute-based encryption with efficient revocation. In *Computer and Information Technology (CIT), 2014 IEEE International Conference on*, pages 281–287. IEEE.