UNIVERSITÀ DEGLI STUDI FIRENZE

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

Security & Privacy in Smart Cities and Industrial IoT

Marco Rasori

Supervisor:
Prof. Gianluca Dini

Coordinator:
Prof. Paolo Frasconi

Evaluation Committee:
Prof. Francesco Buccafurri, *University of Reggio Calabria*
Prof. Ming Li, *University of Arizona*

ING-INF/05 — Cycle XXXII — Years 2016/2019

To my grandpa

Acknowledgments

I would like to thank my supervisor, Prof. Gianluca Dini, for giving me constructive comments and warm encouragement throughout these three years.

I am particularly grateful to Dr. Pericle Perazzo, who greatly helped me in every phase of my Ph.D. studies.

Also, I would like to thank also the members of my supervisory committee, Prof. Enzo Mingozzi, and Prof. Stefano Chessa, who let me improve my research through precious comments and suggestions.

I must express my gratitude to Prof. Shucheng Yu, whose research has been inspiring to me. I thank him for the opportunity of collaborating with him at Stevens Institute of Technology.

I reserve special thanks for Prof. Francesco Buccafurri and Prof. Ming Li, who reviewed my dissertation and helped to improve it through valuable observations.

The administrative staff of the University of Pisa and Florence for their courtesy and help provided to get out of the bureaucratic jungle.

All the people I collaborated with and spent so much good time with during these years. Antonio, Carlo, Francesca, Giovanni, Gloria, Marco, and Niccolò, with whom I shared not only the office and useful suggestions but also thoughts, laughs, and some great hiking adventures. The cybersecurity group and the upstairs Ph.D. students for the many coffee breaks, which spanned from light chitchatting to sharing of opinions about our research topics.

Thanks to my whole family. Thank you for your wisdom, love, and support provided throughout the duration of this experience, for all the one prior to it, and the one that will come. Thank you, mom. Thank you, dad, Andrea, Eli, Edo, and Patrizia. Thank you Giuntoli for your encouragement, for your love, and your patience.

Last but not least, I would like to express my gratitude to all my friends! It would be hard to name some as it would result in forgetting some others; so, if you consider yourself a friend, your acknowledgment is here!

Any omission in this brief acknowledgment does not mean lack of gratitude.

Abstract

The Internet of Things (IoT) technology is now widespread and is indisputably changing our lives. As this technology advances, new security and privacy challenges must be faced. The limitations imposed by resource-constrained devices used in IoT applications play a crucial role and often determine the research directions. This dissertation addresses security problems related to the two main branches of the IoT, namely smart cities and industrial IoT.

IoT applications usually rely on input data coming from either provider's smart devices or by end-user's devices, as in mobile participatory sensing. In the latter case, the infrastructure rests on data reported by participants. This paradigm, however, introduces new security problems related to the trustworthiness of the reported data, which might be inaccurate or even counterfeit. Differently, if the data is sensed by trusted devices, the data trustworthiness is not a concern as far as other security properties, e.g., authentication and integrity, are guaranteed. Depending on the scenario, the sensed data can be directly transmitted to the users or stored on cloud servers. Oftentimes, sensed data includes sensitive or valuable information which is intended to be read only by authorized users. If data is stored on the cloud, the owner loses any control on it, and an attack that leads to data disclosure could represent an important loss of money for the data owner or a serious privacy violation for the users.

In this dissertation we propose novel solutions to the aforementioned problems. False-measurement reports could be tackled by having a set of trusted verifiers which checks the same measurement. If the measurement cannot be directly checked, the verifiers could at least check the participant's position as an indirect proof. To this aim, we propose an effective and secure location verification solution which uses a swarm of few drones equipped only with common radio-frequency transceivers, e.g., WiFi.

Secondly, we propose the use of Attribute-Based Encryption (ABE) in IoT scenarios to protect the data from unauthorized access. We propose ABE-Cities, a secure scheme for smart cities which implements a publish/subscribe-like application in which the data is outsourced to a semi-trusted cloud server. Since ABE encryption might be burdensome for a range of resource-constrained devices, in ABE-Cities, the sensing devices execute only symmetric-key algorithms. Moreover, ABE-Cities leverages the peculiarities of a smart city in order to reduce the complexity of the key revocation operation, which is the most onerous one in ABE systems. In addition, we extend an existing ABE revocation scheme by providing additional security that limits the cloud server capabilities and inhibits it from accessing the data stored on it, when in possession of a revoked key. Finally, we propose fABElous, an ABE scheme for low-bitrate wireless sensor and actuator networks, often used in industrial IoT systems, which aims at minimizing the communication overhead introduced by the adoption of ABE to selectively distribute data through broadcast communications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet of Things (IoT) technology is facilitating seamless advancement in any conceivable aspect of our everyday life. From health care to gaming, from buildings to homes, the IoT-based services and applications developed in the last years are uncountable. As the IoT technology advances, new security and privacy challenges must be faced. In particular, the limitations imposed by the employment of resource-constrained devices play a crucial role and often determine the research directions. A 2017 analysis by GrowthEnabler[1] shows the trend of the global IoT market share until 2020 and reveals that it will be dominated by *smart cities* (26 %) and *industrial IoT* (24 %) sectors. This dissertation addresses security problems related to such domains.

The input source of IoT applications is usually data provided either by smart devices deployed in the environment or by end-user's devices, as in the novel paradigm of mobile participatory sensing. In the latter case, the infrastructure relies on sensed data or events reported by participants. This paradigm, however, introduces new security problems related to the trustworthiness of the reported data, which might be inaccurate or even counterfeit. Differently, if the data is sensed by trusted devices deployed in the environment, the data trustworthiness is not a concern as far as other security properties, e.g., authentication and integrity, are guaranteed. Depending on the scenario, the sensed data can be directly transmitted to the users for a rapid consumption or stored on cloud servers for on-demand usage. Oftentimes, sensed data includes sensitive or valuable information which is intended to be read only by authorized users. Unfortunately, as soon as data lands on the cloud, the owner loses any control on it and has to completely trust the cloud server. In particular, cloud servers are highly exposed to a plethora of attacks, and a data breach could either represent an important loss of money for the data owner or a serious privacy violation for the users.

In this dissertation we propose solutions to the aforementioned problems. In

[1]https://growthenabler.com/flipbook/pdf/IOT%20Report.pdf

order to hinder false-measurements reports, a simple solution would employ a set of trusted verifiers to check the same measure. Otherwise, if the measurement cannot be checked, the verifiers could at least check the participant's position as an indirect proof. Lying about the measurement is still possible, but the lying participant must be physically present and must wait for the verifiers to verify his/her position. To this aim, we propose an effective and secure location verification solution which uses a swarm of few drones equipped only with off-the-shelf hardware, i.e., common radio-frequency transceivers.

When the data is stored on semi-trusted cloud servers and is selectively accessed by authorized users, the data should be stored in encrypted form in order to prevent data disclosure due to attacks to the cloud server. A novel public-key encryption scheme called Attribute-Based Encryption (ABE) is tailored for these scenarios as it enforces an access control mechanism directly on the encrypted data. Although ABE is very attractive, it still has some open challenges to be addressed, for example it lacks an efficient key revocation mechanism. As a contribution in this field, we propose a secure scheme called ABE-Cities which makes use of ABE to implement a generic publish/subscribe-like service optimized for a smart city. ABE-Cities takes into account the peculiarities of a smart city, such as its street network, the geographical distribution of the sensor network, etc., and maps them onto the underlying ABE scheme in order to gain efficiency in key revocation procedures, which are the most critical ones. In addition, we extend an existing ABE revocation scheme (Yu et al., 2010a) in order to improve its security against the cloud server, otherwise capable of accessing the stored data under the assumption that it comes in possession of a revoked key. ABE can also be useful in IoT applications in which data is selectively shared through broadcast communications. To this aim, we propose fABElous, an ABE scheme for low-bitrate wireless sensor and actuator networks, often used in industrial IoT systems, which provides integrity, authentication, confidentiality, and access control on data. The scheme aims at minimizing the impact of the introduction of ABE technology in terms of communication overhead.

1.1 Structure of the Dissertation

This dissertation is divided in a first part about secure location verification and a second part about Attribute-Based Encryption. It follows a brief description of each chapter.

Chapter 2. A Low-Cost UAV-Based Secure Location Verification Method

In this chapter we propose a secure protocol for verifying the location of a device. Differently from the majority of the existing works in literature, we assume the device and the verifying infrastructure (a swarm of Unmanned Aerial Vehicles) to be equipped only with common radio-frequency transceivers, e.g., WiFi. This assumption makes our location verification mechanism less precise than other solutions which use special hardware, e.g., ultra-wideband transceivers. However, the proposed scheme easily fits IoT smart city applications of participatory sensing in which the devices used by the participants are their smartphones. By simulations, we show our scheme's performance in terms of success probability against different types of adversaries. The results presented in this chapter have been published in the 12th International Conference on Availability, Reliability and Security (Rasori et al., 2017).

Chapter 3. A Lightweight and Scalable Attribute-Based Encryption System for Smart Cities

In this chapter we propose ABE-Cities, a scheme for IoT smart city applications in which the data is stored on semi-trusted storage. The scheme makes use of Attribute-Based Encryption (ABE) to secure the data and to allow fine-grained access control on it. ABE-Cities senses data from the city and stores it on a semi-trusted cloud server in an encrypted form. Then, it provides users with keys able to decrypt only data sensed from authorized paths or zones of the city. In ABE-Cities, sensors perform only lightweight symmetric-key encryption, thus we can employ constrained sensor devices such as battery-powered motes. ABE-Cities allows to plan an expiration date for each key, as well as to revoke a given key in an unplanned fashion. We analyze and leverage the peculiarities of a smart city in order to lighten the key revocation mechanism, which is the most burdensome operation in ABE systems. Through simulations on large street networks, we prove that ABE-Cities scales well with the number of users and the number of streets. The results presented have been published in Computer Communication Journal (Rasori et al., 2020).

Chapter 4. Improving KP-ABE Revocation Mechanism

In this chapter we build on top of Yu et al.'s (2010a) proxy re-encryption scheme and propose a Key-Policy Attribute-Based Encryption (KP-ABE) scheme that limits the semi-trusted cloud capabilities. In Yu et al.'s scheme, the key revocation is accomplished by the cloud server and consists of an update (re-keying) of all the decryption keys in the system but the one to be revoked. However, if the cloud server happens to be in possession of a revoked key, it could “undo” the revocation and access the data stored on it. In our scheme, we split the key revocation task on both the cloud server and legitimate users, so a user’s decryption key is re-keyed through a double update. Likewise, in our scheme the update (re-encryption) of old ciphertexts stored on the cloud server is performed by both the cloud server and the users. We show in our construction that this is practical for the user and verifiable for the cloud server.

Chapter 5. fABElous: An Attribute-Based Scheme for Industrial Internet of Things

In this chapter we use the dual scheme of KP-ABE, i.e., Ciphertext-Policy Attribute-Based Encryption (CP-ABE), and propose fABElous, a secure scheme for industrial IoT which provides integrity, authentication, confidentiality, and access control on data. In these scenarios, where low-power and low-bitrate communication protocols are used, the challenge is to provide all the aforementioned security properties while keeping the communication overhead as low as possible. We describe how our system provides them, and we show how it outperforms a system which integrates CP-ABE naively. The results of this chapter have been published in the 5th IEEE International Conference on Smart Computing (La Manna et al., 2019).

Chapter 2

A Low-Cost UAV-Based Secure Location Verification Method

No man is wise enough by himself.

Plautus

Secure location verification is a process by which an infrastructure composed by one or more *verifiers* attempts to verify that a *prover* is actually placed where it claims to be. The problem of secure location verification has been widely studied, and many different solutions have been proposed (Zeng et al., 2013). However, existing solutions make use of special hardware and/or many fixed verifiers, which entail high deployment costs and make them scarcely attractive. A promising and low-cost approach involves the use of *Unmanned Aerial Vehicles* (*UAVs*) as verifiers. In the last decade, UAVs employment has known a prosperous growth, especially in commercial and civil fields. UAVs have been used in swarms to accomplish disparate tasks (Costa et al., 2012; Waharte et al., 2009). Recent studies addressed the problem of secure location verification by means of UAVs (Perazzo et al., 2015; Yokoyama et al., 2014). However, they use special hardware, e.g., ultra-wideband (UWB) or stereo cameras, which makes them expensive and therefore hard to realize.

In this work, we present a novel low-cost secure location verification approach based on a swarm of few UAVs equipped with common radio-frequency (RF) transceivers (e.g., WiFi). This perfectly fits, for instance, a crowd sensing application in which a set of participants provided with smartphones share their positions to estimate the crowd density in some area. In this application, the UAVs can carry out random spot checks on participants in order to assure that their generated positions are genuine.

In our work, we suppose that a device claims its position and is reached by the swarm of UAVs, which places in formation around the device. Then, through an RF communication, a location verification protocol starts, and the device is asked to broadcast a message. By measuring the strengths of the received signals by the UAVs, the system can establish if the device claimed its actual position.

The work has several contributions. We first analyze a simple attack in which a malicious prover claims a false position. Then we introduce a stronger adversary that is capable of tracking UAVs' positions and adjusting its transmission power. By experimental evaluations, we investigate the impact of swarm cardinality and formation on the ability of detecting the presence of an adversary. The results of these tests show that a swarm of only three UAVs detects more than 99 % of the attacks starting from a falsification distance of 20 m against a basic adversary, and 35 m against a stronger adversary.

The chapter is organized as follows: in Section 2.1 we compare our approach with related work. In Section 2.2 we introduce system and adversary models and describe the proposed location verification protocol. In Section 2.3 we evaluate the success probability of the adversary.

2.1 Related Work

The use of UAVs as mobile infrastructure represents a low-cost solution to the secure location verification problem. Yokoyama et al. (2014) use UAVs in a study related to secure positioning. They describe a method based on image processing to estimate the distances between the verifiers and the prover. This implies UAVs need to be equipped with high-resolution stereo cameras. In contrast, our approach is cheaper because it takes advantage of basic onboard components of a UAV and does not require additional hardware. Moreover, a visual technique entails a direct line of sight between every verifier and the prover. This constraint is not mandatory in our model.

Perazzo et al. (2015, 2016a) approach the location verification problem using one UAV. They base their solution on UWB transceivers. However, at the present time, these transceivers are neither widespread nor cheap.

Baker and Martinovic (2016) describe an approach for secure location verification that employs a mobile verifier and at least one fixed verifier to determine prover's position by means of time difference of arrival (TDoA). This solution necessitates strict time synchronization between the base stations to achieve accuracy, and this requires technical effort and raises implementation costs. Our protocol is not subject to any rigid synchronization constraints, and thus it can be realized in a cheaper way.

Rasmussen et al. (2008) describe an approach in which the verifiers are covert base stations, and another approach that employs a mobile base station. They both rely on time differences between RF and ultrasonic (US) signals sent by the prover in order to estimate the distances between verifiers and prover. Such a solution depends on US communication modules that are not off-the-shelf components in commercial UAVs. Our solution is more general and cheaper.

Other studies (Vora and Nesterenko, 2004; Čapkun and Hubaux, 2006; Perazzo et al., 2016b) deal with secure location verification by relying on infrastructures composed of many fixed verifiers whose locations are known. A fixed infrastructure is a huge constraint in terms of deployment and maintenance costs. In contrast, UAVs used as mobile verifiers are cheaper and more versatile since a few of them can cover wide areas. Moreover, Čapkun and Hubaux (2006) and Perazzo et al. (2016b) make use of UWB transceivers that are currently not widespread. In contrast, our solution presumes the use of common wireless communication hardware (e.g., WiFi), which is often installed off the shelf on UAVs and mobile devices.

2.2 System and Adversary Models

In our system, a swarm of UAVs forms the verification infrastructure where each UAV plays the role of verifier. UAVs are mobile stations, so they can autonomously reach given positions to start the location verification protocol. The *prover* is a device, for example a smartphone, claiming to be at a certain position (*claimed position*, p_C) that must be verified. We use n to indicate the *swarm cardinality*, i.e., the number of UAVs in the swarm.

We require UAVs to be equipped with a GPS module and an RF communication module (e.g., a WiFi transceiver). We assume that UAVs can determine their own positions by means of GPS. The RF module is used for communication with the prover and the other UAVs. Moreover, we suppose that one UAV, the *leader*, shares a secret K (*shared secret*) with the prover. The secrets can be distributed to the provers in a secure manner through a generic key distribution scheme (Du et al., 2004). The way in which this is deployed goes beyond the scope of this work. We also assume the UAVs can communicate securely among one another. The UAVs move toward the claimed position and take positions within a *communication range* R from it, according to a given *swarm formation*. Then, the following location verification protocol, represented also in Figure 2.1, takes place:

$M1$:	leader $\rightarrow$ prover:	N	
$M2$:	prover $\not\rightarrow$ *:	$\text{Sign}_K(N)$	
$M3$:	$u_i \rightarrow$ leader:	$\text{Sign}_K(N), P_{Rx,i}$	$\forall i$,

where the symbol $\not\rightarrow$ * represents a broadcast message.

The leader starts the protocol by transmitting a nonce N to the prover, while the other UAVs remain passive. Then, the prover creates and broadcasts the message $M2$, which includes the nonce signed with the shared secret K. This is needed to authenticate the prover, and it is necessary to avoid that a malicious entity impersonates the actual prover. Additionally, we assume that the prover transmits $M2$

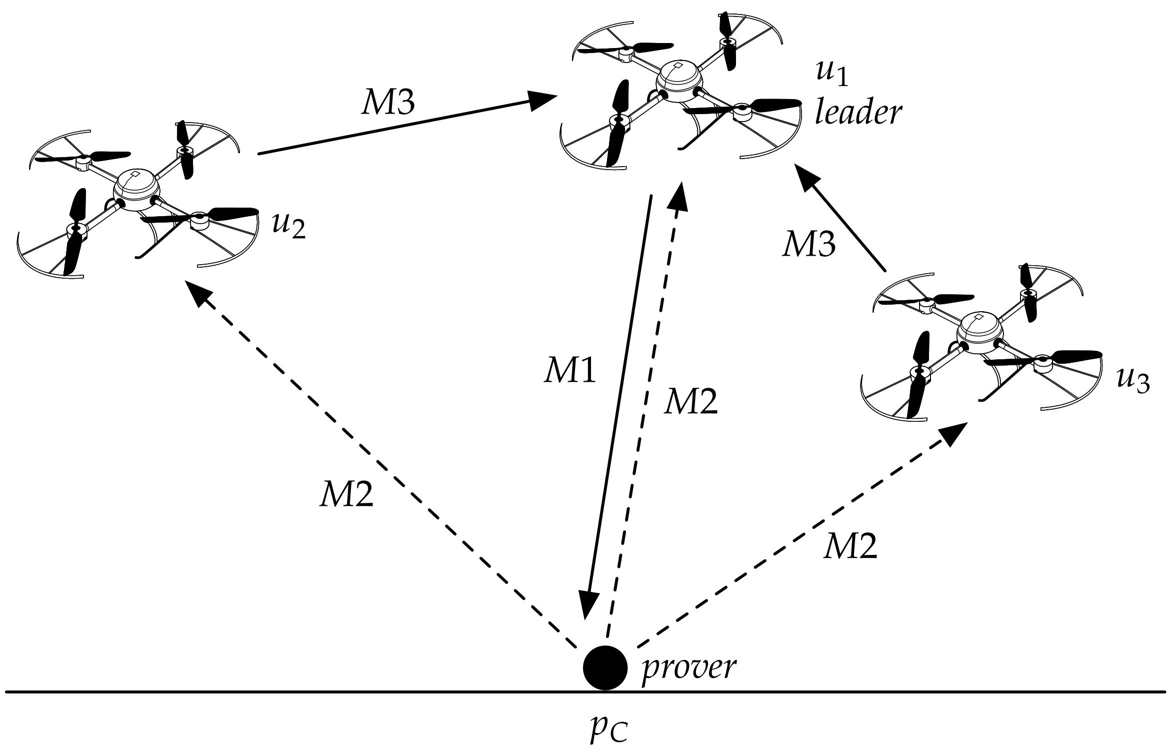


Figure 2.1: Location verification protocol. p_C is the *prover*'s claimed position; solid arrows indicate unicast messages, whereas dashed arrows indicate broadcast message.

using a *nominal transmission power* P_{Tx}, which is known by the leader. Each UAV u_i is supposed to receive $M2$ with power $P_{Rx,i}$ (*received power*), and send the message $M3$ to the leader, including the signature and the received power. As soon as the leader receives all the messages from the other UAVs, it verifies the signature of all the received messages by using the shared secret K. Then, according to a path loss model, the leader computes the power that u_i should have received (*expected power*, $P_{Exp,i}$). The leader detects an attack if at least one absolute difference between the expected and the received powers is greater than a predetermined *consistency threshold* ΔP_{Rx}:

```
if $(\forall i \; |P_{Exp,i} - P_{Rx,i}| \leq \Delta P_{Rx})$
    then  no attack
    else  attack detected.
```

In the following section, we will show how the system can fix a value for the consistency threshold.

Path Loss Model and Consistency Threshold Computation

The received power that each UAV experiences can be modeled through the standard log-distance path loss model (Andersen et al., 1995), which follows the equation:

$$P_{Rx} = P_{Tx} - PL_0 - 10\gamma \log_{10}\left(\frac{d}{d_0}\right) - X_g, \tag{2.1}$$

where P_{Tx} is the transmission power in decibel-milliwatt, PL_0 is the path loss in decibel (dB) at the reference distance d_0, γ is the *path loss exponent*, and X_g, in dB, represents the *shadowing effect term* and is modeled as a normal random variable with zero mean and standard deviation σ_g. We assume to know the parameters of the log-distance path loss model, namely the reference distance d_0, the path loss at the reference distance PL_0, the path loss exponent γ, and the standard deviation σ_g of the Gaussian random variable X_g.

By Eq. 2.1, the leader can estimate the expected power at u_i by setting $d = d_{C,i}$:

$$P_{Exp,i} = P_{Tx} - PL_0 - 10\gamma \log_{10}\left(\frac{d_{C,i}}{d_0}\right), \tag{2.2}$$

where $d_{C,i}$ is the distance between the u_i's position (which is known by the leader) and the prover's claimed position. Obviously, the shadowing effect term is not included in (2.2) because its value cannot be predicted by the system. Therefore, in case of an honest prover, the received power and the expected power differ only in the shadowing effect term.

We fix the consistency threshold in order to obtain a given probability of false positives, i.e., honest provers considered adversaries. The *false positive probability* (fp) coincides with the probability of warning an attack in the honest scenario:

$$fp = 1 - \prod_{i=1}^{n} \Pr\left(\left|P_{Exp,i} - P_{Rx,i}\right| \leq \Delta P_{Rx}\right), \tag{2.3}$$

where the product represents the probability that there are no inconsistencies between the expected and the received power at every UAV. Since the expected and the received powers differ only in the shadowing effect term $X_{g,i}$ (cfr. Eqq. 2.1 and 2.2), and since $X_{g,i}$ are Gaussian and identically distributed random variables with standard deviation σ_g, then Eq. 2.3 becomes:

$$fp = 1 - \left[2\,\Phi\left(\frac{\Delta P_{Rx}}{\sigma_g}\right) - 1\right]^n, \tag{2.4}$$

where $\Phi\left(\cdot\right)$ is the normal cumulative distribution function. It follows that:

$$\Delta P_{Rx} = \sigma_g \Phi^{-1}\left(\frac{1 + \sqrt[n]{1 - fp}}{2}\right). \tag{2.5}$$

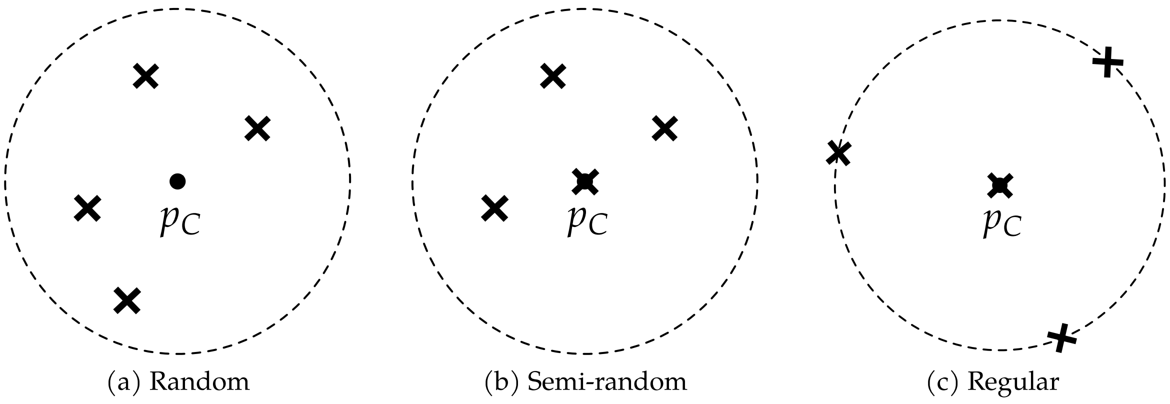


Figure 2.2: Example of the swarm formations tested. Crosses represent the UAVs while the black dot is the claimed position. The dashed line represent the communication range R from p_C.

Adversary Model

An attack occurs when a prover lies on its position. We assume the adversary claims to be at a distance d_f (*falsification distance*) from its *actual position* p_A, in a random direction. In other words, the falsification distance is the distance between the claimed position and the actual position. Such attack is known as the *false reported location attack* (Zeng et al., 2013), and we will refer to this malicious prover as the *blind* adversary.

The second kind of attack that we consider extends the first one. The *observer* adversary is also able to estimate the position of every UAV. Using this information, he can adapt the transmission power of $M2$ in order to cheat the verification system. For each u_i, the observer adversary computes the transmission power $P_{TxA,i}$ for which the expected and the received power at u_i coincide ($P_{Exp,i} = P_{Rx,i}$), so obtaining:

$$P_{TxA,i} = P_{Tx} + 10\gamma \log_{10}\left(\frac{d_{A,i}}{d_{C,i}}\right), \quad \forall i, \tag{2.6}$$

where $d_{A,i}$ and $d_{C,i}$ are the distances from u_i to the actual position and to the claimed position, respectively. We assume the adversary as a single entity which transmits from a unique position, hence no collusion attacks are considered. We further assume that the adversary does not have directional antennas by which he could send multiple copies of $M2$ with different transmission powers. Therefore, the adversary chooses the value P_{TxA}, that is the mean value of all the $P_{TxA,i}$, as transmission power.

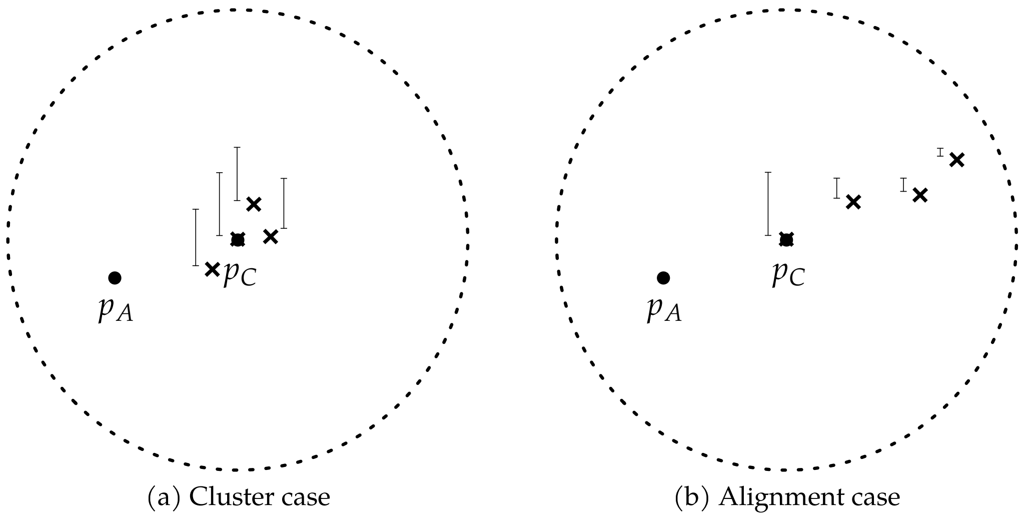


Figure 2.3: Critical cases for the semi-random formation. Crosses are the UAVs, and bars represent the expected powers $P_{Exp,i}$.

Swarm Formations

The way the swarm places in the proximity of the claimed position plays an important role for the attacks detection. Our model considers UAVs placement in the 3D space. Initially, we assume a fixed altitude h for all the UAVs in the swarm, and their positions within R from the claimed position, according to a uniform random distribution. From now on, we will refer to this formation as the *random* formation (Figure 2.2a). It is the simplest formation and also makes UAVs' positions less predictable for the blind adversary. However, if all the UAVs happen to be far from both the claimed and the actual position, a blind adversary has high success probability. Indeed, all the expected and the received powers would be low, and thus similar. Of course, the same problem arises with the observer adversary too.

To solve this problem, we put one UAV in plumb line above p_C at altitude h_{pl} lower than h, while the others are randomly disposed as in the random formation. This change should improve system security because of the presence of a UAV near the claimed position, whose expected power should be high. We will refer to this formation as the *semi-random* formation (Figure 2.2b). However, this solution might be still vulnerable against the observer adversary in those cases in which UAVs are placed in a limited area near p_C, forming a *cluster* (Figure 2.3a). Indeed, all the expected powers are similar, and therefore the observer adversary can cheat the verification system by properly increasing the transmission power. Moreover, another critical case may happen when the UAVs are aligned and opposite to p_A as shown in Figure 2.3b. In this situation, a high transmission power by the observer adversary could result in received powers comparable with the expected ones. This means that in such a scenario the observer adversary has high success probability.

To solve these problems, we then analyze a third formation: one UAV is still placed in plumb line above the claimed position at altitude h_{pl}, and the others are evenly distributed on a circumference centered on p_C at altitude h; the radius is computed so that the distance between p_C and these UAVs is equal to the communication range R. By doing so, we avoid the weaknesses of the semi-random formation shown in Figures 2.3a and 2.3b. From now on, we will refer to this formation as the *regular* formation (Figure 2.2c).

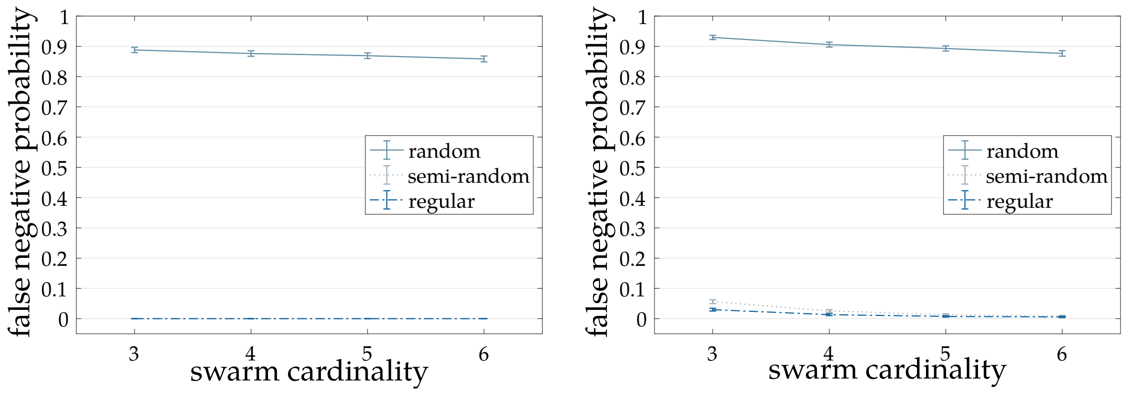
2.3 Experimental Evaluation

Our goal is to achieve a low *false negative probability* (fn), i.e., the percentage of adversaries considered honest provers, while keeping a low false positives probability. While fp was fixed a priori and set to 1 percent $(fp = 0.01)$, fn was obtained through simulations of different scenarios.

The parameters of the log-distance path loss model mainly depend both on the environment and the obstacles that the signal encounters along its path. Yanmaz et al. (2011) studied WiFi channel for UAV-to-ground link. Accordingly to their work, we set the path loss exponent γ to 2.6. The swarm cardinality was set from a minimum of 3 to a maximum of 6 to test whether acceptable outcomes could be obtained even with fewer UAVs. Moreover, we assumed $\sigma_g = 3\,\text{dB}$, the reference distance $d_0 = 1\,\text{m}$, and a communication range R of 100 m. Every formation was basically placed according to Section 2.2. In the random formation, UAVs were disposed with a fixed altitude, i.e., $h = 25\,\text{m}$. In the semi-random and regular formations, the altitude for the plumb-line UAV was set to $h_{pl} = 5\,\text{m}$.

We deployed our simulator in Matlab. In order to obtain statistically sound results, 5000 independent trials with different seeds were run for each scenario. Within each trial, we firstly simulated the presence of an adversary by generating a claimed position and an actual position. Then, we placed the UAVs around p_C, according to the formation to be tested. At that point, the location verification protocol was simulated; the received powers by the various UAVs were simulated as well following Eq. 2.1, and a random value for the shadowing effect was generated for every value of $P_{Rx,i}$.

In our first set of simulations we set the falsification distance d_f to 30 m, and we tested swarm formations security in terms of false negative probability, varying the swarm cardinality. Figure 2.4a shows the formations' performances against the blind adversary. We observe that the random formation was the weakest one in detecting attacks. Indeed, fn was about 85 % with the highest swarm cardinality tested. On the other hand, both semi-random and regular formations did not miss a single detection within every scenario tested.



(a) Blind adversary (b) Observer adversary

Figure 2.4: False negative probability and 95 % confidence intervals for different number of UAVs. $d_f = 30\,\mathrm{m}$.

Figure 2.4b shows the false negative probability with regards to the observer adversary. Obviously, security lowers for all the formations if compared to the case of the blind adversary. However, semi-random and regular formations still exhibit excellent results. Their performance is clearly much better than the one provided by the random formation; specifically, with a swarm of 3 UAVs, the random formation did not detect 93 % of the attacks whereas the semi-random and the regular formations did not detect 5.6 % and 3 % of the attacks, respectively. In this case, but also with higher swarm cardinalities, the regular formation outperforms the others and results as the best choice for the attacks detection. With the semi-random formation, cluster and alignment cases are more likely when the swarm cardinality is low. Therefore, as n increases, the performance of the semi-random formation improves to the point of performing almost equivalently to the regular formation. With a swarm of 6 UAVs, fn for the semi-random and the regular formations are 0.85 % and 0.6 %, respectively.

Figure 2.5 shows the results obtained varying d_f from 0 to 50 m. The swarm cardinality was set to 3. Figure 2.5a confirms that the random formation is not suitable to make the system secure since fn is over 60 % when $d_f = 50\,\mathrm{m}$. The other two formations perform equivalently against the blind adversary. Specifically, they both achieved a false negative probability of about 1 % starting from a falsification distance of 20 m.

Figure 2.5b shows the results obtained with an observer adversary. It is noticeable how the regular formation outperforms the others; specifically, starting from $d_f = 20\,\mathrm{m}$ the regular formation detects more attacks than the semi-random one. Indeed, the latter is vulnerable to the critical cases of Figures 2.3a and 2.3b, and this leads to a performance lowering. In contrast, the regular formation is more robust and avoids such critical cases. After the analysis varying the falsification distance,

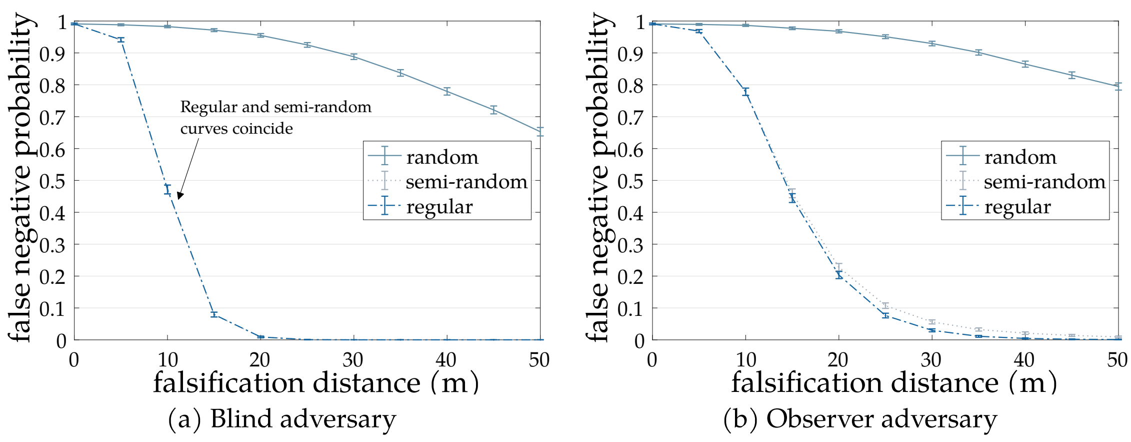


Figure 2.5: False negative probability and 95 % confidence intervals as function of d_f, ranging from 0 to 50 m. $n = 3$.

we can assert that the regular formation is the most secure among the ones we tested, achieving a false negative probability lower than 1 % starting from a falsification distance of 35 m with a swarm of 3 UAVs.

Chapter 3

A Lightweight and Scalable Attribute-Based Encryption System for Smart Cities

Simplify, then add lightness.

Colin Chapman

A *smart city* is an urban environment that offers digital services to improve the quality of life of the citizens. Such services span across all sectors of society including health, logistics, and mobility, and they often capitalize on the Internet of Things (IoT) as enabling technology. Smart devices underlying these services produce large amounts of data which can be outsourced to a *cloud server*. This is usually preferred to an in-house solution because it reduces costs while providing high availability. In many cases, sensed data includes sensitive or valuable information which is intended to be read only by authorized users. Unfortunately, as soon as data lands on the cloud, we lose any control on it and we have to completely trust the cloud server. In particular, cloud servers are highly exposed to a plethora of attacks: from "classic" injection (Halfond et al., 2006) to recent hardware-based attacks like Spectre (Kocher et al., 2018) and Meltdown (Lipp et al., 2018). A data breach on a smart city system could either represent an important loss of money for the data owner or a serious privacy violation for the citizens[1]. Moreover, cloud service providers may have some incentive to release stored information to others (Di Vimercati et al., 2007; Coppolino et al., 2017).

A possible solution to this problem is to store data in an encrypted form on the cloud server. This can be done by means of *Attribute-Based Encryption* (ABE) (Sahai and Waters, 2005; Goyal et al., 2006; Bethencourt et al., 2007; Ostrovsky et al.,

[1]https://securityintelligence.com/ponemon-cost-of-a-data-breach-2018/

2007), which also ensures a fine-grained access control on data. In ABE everyone can encrypt because only public parameters are used for encryption. Decryption is instead performed by means of a *decryption key*, which is specific to each user and generated by a *Trusted Third Party* (TTP). Decryption keys and encrypted data "embed" *attributes* and *access policies*, which are basically Boolean formulas defined over the attributes. A given decryption key is able to decrypt a given piece of data only if the embedded access policy is satisfied by means of the embedded attributes.

In this chapter we present ABE-Cities, an encryption system for smart city applications employing Attribute-Based Encryption. ABE-Cities allows fine-grained access control on the encrypted data stored in the cloud server, and it is secure against traffic eavesdropping, sensors compromise, and data leakage from the cloud server. ABE-Cities senses data from the city and stores it on the cloud server in an encrypted form. Then, it provides users with keys able to decrypt only data sensed from authorized paths or zones of the city. In ABE-Cities, sensors perform only lightweight symmetric-key encryption, thus we can employ resource constrained sensor devices such as battery-powered motes. This makes ABE-Cities suitable for a broad set of IoT smart city applications. ABE-Cities allows us to plan an expiration date for each key, as well as to revoke a given key in an unplanned fashion. We prove that ABE-Cities scales well with the number of users and the city size by simulating it with 3000 users on the Houston street network (12 000+ streets), and with 30 000 users on the Beijing street network (30 000+ streets). In addition to the "vanilla" ABE-Cities scheme, we propose an "advanced" one that leverages the possible presence of IoT gateways to reduce the TTP computational load. This advanced version is based on Merkle trees, and it takes advantage of nodes acting as sensor gateways to outsource a significant amount of computation from the TTP. We finally validate the advantages introduced by the advanced scheme by testing the encryption performance on Houston and Beijing street networks. We remark that with respect to generic IoT ABE systems (e.g., Yu et al. (2011)), ABE-Cities is particularly suitable for smart city applications. This is because ABE-Cities uses a street network representation that makes key revocations efficient, and it avoids broadcast communications which may be infeasible in sensor networks distributed over large urban areas.

This chapter is organized as follows. Section 3.1 compares with relevant related work. Section 3.2 introduces some background on ABE and other techniques that we use in our system. Section 3.3 describes the vanilla ABE-Cities scheme, and the adversary model. Section 3.4 introduces different attribute representations of the street network, resulting in different system performance. Section 3.5 compares quantitatively the different attribute representations by simulations, and it evaluates the scalability of ABE-Cities on small and large street networks. Section 3.6 describes the ABE-Cities advanced scheme. Section 3.7 validates the advanced scheme by testing its encryption performance on simulated large street networks.

3.1 Related Work

ABE has been used to protect the confidentiality in IoT (Yao et al., 2015; Yu et al., 2011; Odelu et al., 2017; Huang et al., 2018; Ambrosin et al., 2016; Wang et al., 2014; Oualha and Nguyen, 2016; La Manna et al., 2019; Girgenti et al., 2019), digital health (Lounis et al., 2012; Hu et al., 2013; Ibraimi et al., 2009; Akinyele et al., 2011), military battlefields (Roy and Chuah, 2009), and online social networks (Baden et al., 2009).

Yu et al. (2010a) first used ABE techniques for outsourcing sensitive data to a semi-trusted cloud storage, while enforcing fine-grained access control at the same time. Their scheme is well applicable in digital health applications, where patients own privacy-sensitive records and can employ full computational capabilities. However, it is less suitable for IoT applications, in which data is produced by constrained devices, which usually lack the necessary computational power and energy to perform ABE encryption. In our system, IoT devices perform only lightweight symmetric-key encryption, thus we can employ constrained sensor devices and battery-powered motes. ABE-Cities is thus suitable for a broader set of IoT applications.

Wang et al. (2010) proposed a hierarchical ABE scheme that allows the TTP to delegate part of the responsibility to other authorities, which can independently make decisions on the semantics of their attributes. Hierarchical ABE allows also for proxy re-encryption, but it forces a fixed structure for the access policies, so it limits the flexibility of the access control system. In this work, we focused on a non-hierarchical ABE scheme with a single TTP, leaving possible hierarchical extensions for future work.

Yu et al. (2011) proposed an ABE scheme to encrypt data sensed by a wireless sensor network (WSN). The authors used broadcast encryption to revoke keys in an efficient manner. Although their system is suitable for a large set of application scenarios involving locally distributed WSNs, it could be unpractical for geographically distributed ones, like those employed in a smart city. This is because an actual broadcast channel could not be realizable in these scenarios. In our system, we use a key revocation method whose efficiency does not depend on the presence of a broadcast channel, but rather on a convenient attribute representation of the smart city. Moreover, the authors of Yu et al. (2011) perform ABE encryption directly on the sensors, which could not have the necessary computational power and energy to do it. In our system the IoT devices perform only lightweight symmetric-key encryption, thus we can employ constrained sensor devices and battery-powered motes.

Yao et al. (2015) proposed an ABE scheme based on elliptic curves instead of pairings. This makes ABE operations more lightweight and thus more suitable for IoT constrained devices. In our scheme this feature is not needed since the IoT devices perform only lightweight symmetric-key encryption. This permits us to use the pairing-based ABE scheme of Goyal et al. (2006), which allows us to employ advanced features unavailable in the Yao et al.'s scheme, like proxy re-encryption (Yu et al., 2010a).

Recently, Odelu et al. (2017) proposed an ABE scheme for IoT applications, which guarantees $\mathcal{O}(1)$ encryption and decryption times. Such a scheme forces the access policies to use only AND operators, so it limits the access control flexibility. On the contrary, our system is based on Goyal et al.'s scheme, which offers a far greater degree of expressiveness.

3.2 Preliminaries

Key-Policy Attribute-Based Encryption

Attribute-Based Encryption comes in two flavours: *Key-Policy ABE* (KP-ABE) and *Ciphertext-Policy ABE* (CP-ABE). In this section we describe the former scheme, while in Section 5.2 we describe CP-ABE in detail.

The KP-ABE paradigm was first introduced by Goyal, Pandey, Sahai, and Waters (Goyal et al., 2006). In the following, we will refer to this scheme as the GPSW scheme. In KP-ABE an encrypting party labels each piece of encrypted data (*KP-ABE ciphertext*) with a set of attributes which describes it (*encryption attributes*, γ). Decryption keys embed an access policy ($\mathcal{T}$), which is basically a Boolean formula defined over some attributes (*access policy attributes*, λ). An access policy can be represented as a tree in which leaves are access policy attributes, and non-leaf nodes are Boolean operators. A given decryption key is able to decrypt a given KP-ABE ciphertext only if the access policy can be satisfied by using the attributes labeling the ciphertext.

Figure 3.1 shows an example of a KP-ABE ciphertext labeled with some encryption attributes, and a decryption key embedding an access policy.

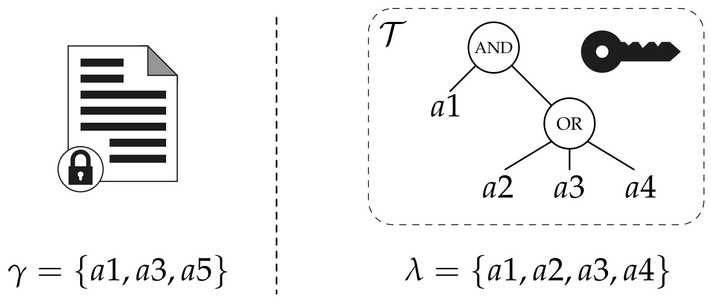


Figure 3.1: Example of KP-ABE ciphertext and decryption key. The access policy can be read as "at least one attribute between $a2$, $a3$, and $a4$ must be present among the attributes labeling the ciphertext, and $a1$ must be present as well".

In the example, when evaluating the access policy against the given KP-ABE ciphertext, the decryption key uses the attributes labeling the ciphertext. In particular, the access policy can be satisfied by using the attributes $a1$ and $a3$, so the decryption key is capable of decrypting the KP-ABE ciphertext.

The set of all the attributes used in a given KP-ABE scheme is called *universe of the attributes* $(\mathcal{U})$. Without losing in generality, in the following we will indicate an attribute in the universe either with its unique nameor with a unique natural number, which is more convenient in formulas. Therefore, the attribute sets $\mathcal{U}$, γ, and λ are subsets of $\mathbb{N}$. Who encrypts data is called a *data producer*, whereas who holds a decryption key and decrypts data is called a *data consumer*.

In this work we build on GPSW scheme. In order to ease the reading, we abstract away the mathematical insight of such a scheme. The interested reader can refer to Goyal et al. (2006) for such details. We model GPSW scheme with the following black-box primitives.

$(MK, PK) = \text{KP.Setup}(\mathcal{U})$ This primitive initializes the scheme. It takes as input the universe of the attributes $\mathcal{U}$ and generates a random *master key* $MK = (y, t_{i \in \mathcal{U}})$, which must be kept secret by the TTP, and an associated set of *public parameters* $PK = (Y, T_{i \in \mathcal{U}})$, which must be distributed to the data producers. Each attribute i in the universe is associated with a t_i and a T_i. The KP.Setup primitive is executed by the TTP.

$CT = \text{KP.Encrypt}(M, \gamma, Y, T_{i \in \gamma})$ This primitive encrypts a message M with the encryption attributes γ. It takes as input Y and $T_{i \in \gamma}$, which are public parameters, and it produces the KP-ABE ciphertext $CT = (\gamma, \hat{ct}, ct_{i \in \gamma})$. Each encryption attribute $i \in \gamma$ is associated to a *KP-ABE ciphertext component* ct_i. The KP.Encrypt primitive is executed by a data producer.

$DK = \text{KP.KeyGen}(MK, \mathcal{T})$ This primitive generates a decryption key $DK = (\mathcal{T}, \lambda, dk_{i \in \lambda})$, which is provided to a data consumer. It takes as input the master key MK and an access policy $\mathcal{T}$, with access policy attributes λ. Each access policy attribute $i \in \lambda$ is associated with a *decryption key component* dk_i. The KP.KeyGen primitive is executed by the TTP.

$M = \text{KP.Decrypt}(CT, DK)$ This primitive retrieves the message M from the KP-ABE ciphertext CT if the decryption key DK is *authorized* to decrypt, i.e., if the access policy $\mathcal{T}$ embedded in the decryption key is satisfied by means of the encryption attributes γ labeling the KP-ABE ciphertext. It produces $\perp$ otherwise. The KP.Decrypt primitive is executed by a data consumer.

Proxy Re-Encryption

Proxy Re-Encryption (PRE) is a technique which allows an entity, given an encrypted message, to produce the same message encrypted with a different key, without accessing the message itself. By using this technique, one can re-encrypt old ciphertexts in order to prevent a revoked key to decrypt them and outsource this burdensome operation to a cloud server without compromising data confidentiality.

In this work we build on Yu et al.'s PRE scheme (Yu et al., 2010a). In the following, we will refer to this scheme as the YWRL scheme. The YWRL scheme implements the revocation of a decryption key by means of a system-wide update of a subset μ of the access policy attributes ($\mu \subseteq \lambda$) to a new *version*. The subset μ is called *updatee set* and contains the minimal set of attributes without which the decryption key cannot ever be satisfied. In the example of Figure 3.1, the updatee set is $\mu = \{a_1\}$. As a consequence, after the system-wide update, the revoked decryption key will not be able to decrypt any KP-ABE ciphertext anymore. Updating an attribute i to a new version means that all the cryptographic quantities in the system related to that attribute are changed. Specifically, for each attribute $i \in \mu$, (1) the t_i of the master key, (2) the T_i of the public parameters, (3) the ct_i of all the KP-ABE ciphertexts having i as encryption attribute, and (4) the dk_i of all the decryption keys (except the revoked one) having i as access policy attribute are updated to new quantities. We indicate such new quantities respectively with t_i', T_i', ct_i', and dk_i'. The t_i''s and T_i''s are computed by the TTP. On the other hand, the computation of the ct_i''s and the dk_i''s is outsourced to the cloud server. To do this, for each attribute $i \in \mu$ the TTP computes a *re-encryption key* (rk_i) and gives it to the cloud server, which applies it to each ct_i to obtain a ct_i', and to each dk_i to obtain a dk_i'.

Note that the computation of all the dk_i''s cannot be completely outsourced to the cloud server, otherwise it would know all the decryption keys, and thus a data breach would compromise the entire system. To avoid this, the cloud server is provided with all the decryption key components except for those relative to a special attribute (*dummy attribute*) which has no meaning and is never updated to a new version. The dummy attribute is ANDed at the root of the access policy of every decryption key, and it is included as an encryption attribute in all the KP-ABE ciphertexts. In this way, a decryption key cannot decrypt anything without the component relative to the dummy attribute, which is known only by the data consumer holding the whole decryption key. A data breach on the cloud server would leak only incomplete decryption keys, which are useless.

Proxy re-encryption is performed in a lazy fashion (*lazy re-encryption*), meaning that the cloud server does not perform any re-encryption operation at the moment of the key revocation, but only afterwards when needed. At the time of a revocation, the TTP produces a re-encryption key for each attribute in the updatee set and adds it to a *re-encryption key list* (RKL) stored in the cloud server. RKL_i is a table that keeps track of all the re-encryption keys relative to the attribute i, one for each version update of the attribute $(rk_i, rk_i', rk_i'', \dots)$. The cloud server updates components only when a data consumer asks the cloud server for a KP-ABE ciphertext. If either a KP-ABE ciphertext component or a consumer's decryption key component is outdated of more than one version, then the cloud server updates it by using more than one re-encryption key from the list. Finally, the cloud server provides the data consumer with the KP-ABE ciphertext (s)he asked for and, possibly, with the updated decryption key components.

In order to ease the reading, we abstract away the mathematical insight of the YWRL scheme. The interested reader can refer to Yu et al. (2010a) for such details. We model YWRL scheme with the following black-box primitives.

$(t_i', T_i', rk_i) = \text{PRE.UpdateAtt}(i, MK)$ This primitive updates the attribute i to a new version. It takes as input the master key MK, and it produces the new quantities t_i', T_i', and the re-encryption key rk_i. This primitive is executed by the TTP.

$ct_i' = \text{PRE.UpdateCT}(i, ct_i, rk_i)$ This primitive updates a KP-ABE ciphertext component ct_i to a new version ct_i' by means of the re-encryption key rk_i. This primitive is executed by the cloud server.

$dk_i' = \text{PRE.UpdateDK}(i, dk_i, rk_i)$ This primitive updates a decryption key component dk_i to a new version dk_i' by means of the re-encryption key rk_i. This primitive is executed by the cloud server.

Segment Trees

Segment trees (De Berg et al., 2000) are data structures useful in ABE schemes to implement efficient point-in-interval access policies (Attrapadung et al., 2016). A segment tree represents a set of intervals in such a way that it is efficient to query which intervals contain a given number (or *point*). Although segment trees can in principle represent any point and interval in $\mathbb{R}$, in this work we focus on segment trees capable of representing discrete points and discrete intervals included in a limited number range $[1, \rho]$, with $\rho \in \mathbb{N}$. In the following, we will generically use the term "segment tree" to intend such a specific type of segment tree.

A segment tree over the range $[1, \rho]$ is a binary tree in which each point in $[1, \rho]$ is relative to a leaf. For example, Figure 3.2 shows a segment tree over the range $[1, 7]$.

The leaf l_1 is relative to the number 1, the leaf l_2 to the number 2, and so on. A generic point v in the range $[1, \rho]$ is represented by a *point representation set* RS_v,

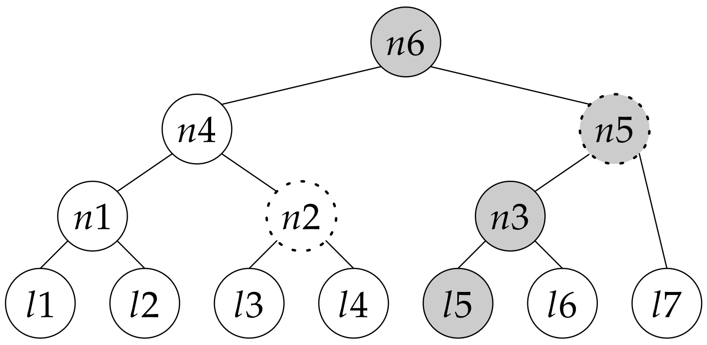


Figure 3.2: Example of a segment tree. Gray nodes form the point representation set of 5, dashed-border nodes form the interval representation set of $[3,7]$.

which is formed by the nodes on the path from the root to the leaf relative to the point. For example, in the segment tree of Figure 3.2, the point $\nu = 5$ is represented by the point representation set $RS_\nu = \{n_6, n_5, n_3, l_5\}$. It can be shown that every point representation set is $\mathcal{O}(\log \rho)$ nodes. On the other hand, a generic interval ι included in the range is represented by an *interval representation set* RS_ι, which is the minimum set of nodes whose descendant leaves form the interval. For example, in the segment tree of Figure 3.2, the interval $\iota = [3,7]$ is represented by the interval representation set $RS_\iota = \{n_2, n_5\}$. It can be shown that every interval representation set is $\mathcal{O}(\log \rho)$ nodes. By construction, point ν belongs to interval ι iff RS_ν and RS_ι have a non-empty intersection, i.e., $\nu \in \iota \Leftrightarrow RS_\nu \cap RS_\iota \neq \varnothing$. For example, in the segment tree of Figure 3.2, point $\nu = 5$ belongs to the interval $\iota = [3,7]$ because $RS_\nu \cap RS_\iota = \{n_5\} \neq \varnothing$.

Segment trees are useful in ABE schemes to implement efficient point-in-interval access policies (Attrapadung et al., 2016), i.e., access policies which are satisfied iff a given point ν belongs to a given interval ι. According to the terminology in the work of Attrapadung et al. (2016), we consider only "KP-ABE Type 1 construction", which means that the ciphertext embeds the point, and the access policy embeds the interval. The point-in-interval access policy is thus implemented in the following way. Each node of the segment tree is represented by an attribute. The KP-ABE ciphertext is labeled with the attributes representing the nodes of the point representation set RS_ν. The access policy is an OR operator between the nodes representing the interval representation set RS_ι. By construction, the access policy is satisfied iff RS_ν and RS_ι have non-empty intersection, that is, iff ν is in ι.

3.3 ABE-Cities

In ABE-Cities, a city is represented by a *street network*, i.e., a graph in which the edges represent *road segments*. Each street is composed of one or more road segments, and

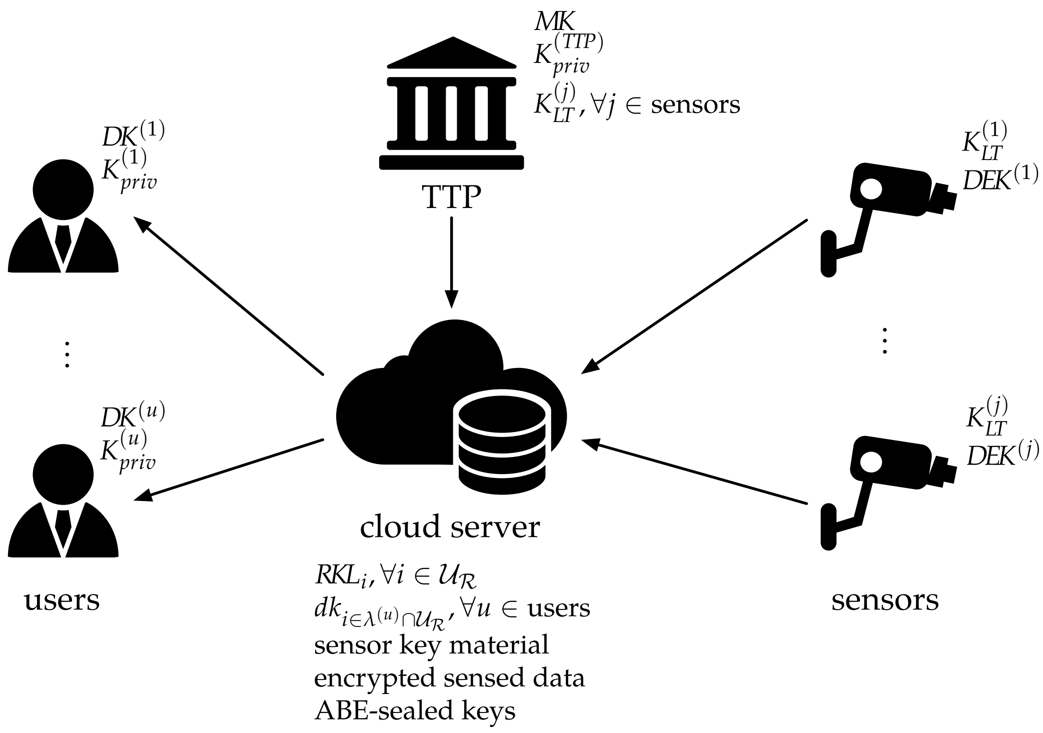


Figure 3.3: ABE-Cities architecture.

it has a unique identifier. A *sensor* is a constrained device placed on a road segment that acquires data (*sensed data*) relative to such a road segment. The sensed data is stored in an encrypted form in the cloud server, where it is made read-only accessible to the *users*. Users represent the data consumers in our system. Each user is authorized to access data sensed from some paths or zones of the city within a specific time period. For example, supposing the sensors to be IP cameras for traffic monitoring, the user can be authorized to access videos showing the route (s)he usually travels from home to work and back. In this way (s)he can detect traffic congestion in advance and possibly choose different routes. To do this, IP cameras could store in the cloud server multiple short video files in an encrypted form, in such a way to implement an encrypted streaming. The user may also monitor multiple routes in order to identify the least congested one day by day. Moreover, some high privileged user could monitor all the city, for example for providing a transportation management service.

Figure 3.3 shows the architecture of ABE-Cities. In the following, we will give a general and intuitive description of the system, and in Section 3.3 we will describe the system procedures in detail. The TTP holds the master key and a public/private key pair $(K_{pub}^{(TTP)}, K_{priv}^{(TTP)})$ through which it can perform digital signature algorithms. The notation $\text{Sign}(\cdot)$ represents the TTP's signature algorithm. The TTP shares with each sensor j a *long-term key* $(K_{LT}^{(j)})$, which is a symmetric key preloaded in the sensors. Each sensor holds also a *data encryption key* ($DEK^{(j)}$), which is a symmetric key used to encrypt the sensed data. For each piece of sensed data, the sensor computes a new data encryption key as a one-way hash of the old one, and the old one is securely destroyed. Each user u holds a decryption key $DK^{(u)}$, and a public/private key pair ($K^{(u)}_{pub}, K^{(u)}_{priv}$) through which (s)he can receive confidential messages by asymmetric encryption. The cloud server maintains a database of re-encryption key lists and a database of decryption key components that allows for proxy re-encryption. In addition it stores *encrypted sensed data*, *sensor key material*, and *ABE-sealed keys*. The encrypted sensed data is the sensed data produced by the sensors and encrypted with the data encryption key. The sensor key material is a set of cryptographic quantities needed by the sensors to encrypt sensed data. The ABE-sealed keys are the data encryption keys encrypted with ABE. Both the sensor key material and the ABE-sealed keys are produced by the TTP. To decrypt a piece of encrypted sensed data, users must first decrypt the corresponding ABE-sealed key thus retrieving the data encryption key, and then decrypt the sensed data with it.

The universe of the attributes $\mathcal{U}$ is logically partitioned into two subsets: $\mathcal{U}_\mathcal{R}$ and $\mathcal{U}_\mathcal{X}$. The $\mathcal{U}_\mathcal{R}$ subset includes the attributes that represent road segments. Such attributes allow us to restrict users to access data sensed only from some paths or zones of the city, specified as a set of road segments (*authorized road segments*). $\mathcal{U}_\mathcal{X}$ includes the attributes that represent time. Such attributes allow us to restrict users to access data sensed only within a specific time period (*validity period*). The maximum resolution with which ABE-Cities can represent validity periods is a system parameter that we call *time unit*. Setting a too short time unit (for example, one second) could be infeasible because the TTP should produce a large set of ABE-sealed keys within a time unit (see Section 3.3). A reasonable choice for the time unit is one day.

Each ABE-sealed key relative to the sensor j ($ASK^{(j)}$) is labeled with the encryption attributes $\gamma^{(j)}$, which are logically partitioned into two subsets: $\gamma^{(j)}_\mathcal{R}$ and $\gamma^{(j)}_\mathcal{X}$. The attributes in $\gamma^{(j)}_\mathcal{R}$ identify the road segment in which the sensor is deployed, while the attributes in $\gamma^{(j)}_\mathcal{X}$ identify the time unit during which data has been sensed (*data production time unit*). Similarly, the access policy $\mathcal{T}$ embedded in each decryption key is logically partitioned into two subtrees: $\mathcal{T}_\mathcal{R}$ and $\mathcal{T}_\mathcal{X}$ (Figure 3.4).

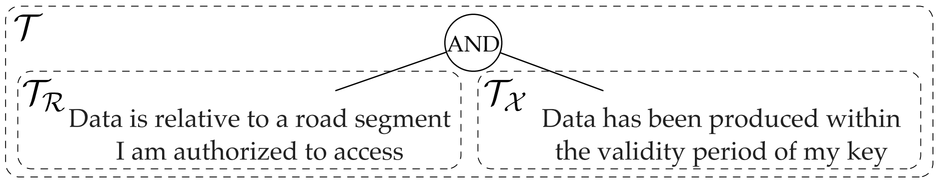

$\mathcal{T}$
AND
$\mathcal{T}_\mathcal{R}$
Data is relative to a road segment I am authorized to access
$\mathcal{T}_\mathcal{X}$
Data has been produced within the validity period of my key

Figure 3.4: Example of access policy for a user's decryption key.

The two subtrees are operands of an AND operator ($\mathcal{T} = \mathcal{T}_\mathcal{R} \wedge \mathcal{T}_\mathcal{X}$), which is the root of the access policy. $\mathcal{T}_{\mathcal{R}}$ identifies the authorized road segments of the decryption key, while $\mathcal{T}_{\mathcal{X}}$ identifies its validity period. In order to decrypt an ABE-sealed key, both the subtrees of the user's decryption key must be satisfied. In the traffic monitoring example, the user can be authorized to access only videos produced after his/her subscription to the service starts, and before it expires. The maximum resolution with which the validity periods can be represented is the time unit.

To implement proxy re-encryption, we use the whole attribute set $\mathcal{U}_{\mathcal{X}}$ in place of the dummy attribute used in YWRL scheme. This allows us to save a component for each decryption key and a KP-ABE ciphertext component for each ABE-sealed key. The attribute set $\mathcal{U}_{\mathcal{X}}$ is a good replacer for the dummy attribute because: (1) at least one attribute in $\mathcal{U}_{\mathcal{X}}$ is ANDed at the root of the access policy of all the decryption keys; (2) at least one attribute in $\mathcal{U}_{\mathcal{X}}$ is an encryption attribute of all the ABE-sealed keys; and (3) the attributes in $\mathcal{U}_{\mathcal{X}}$ are never updated to a new version (see Section 3.3). The cloud server is provided with all the decryption key components except those relative to the attributes in $\mathcal{U}_{\mathcal{X}}$.

System Procedures

In the following, we will implicitly consider the KP-ABE ciphertext components ct_i, the decryption key components dk_i, and the re-encryption keys rk_i as always accompanied by the information about their version.

Setup. This procedure initializes the system. The TTP decides the universe of the attributes $\mathcal{U}$ basing on the street network of the city and on the maximum system lifetime (see Section 3.4). Then, it executes the KP.Setup primitive which takes the universe of the attributes and produces the master key and the public parameters. The TTP keeps the master key secret and initializes an empty re-encryption key list for each attribute in $\mathcal{U}_{\mathcal{R}}$ in the cloud server.

Key Distribution. This procedure provides a user u with a decryption key $DK^{(u)}$. It is executed when a new user joins the system, as well as when an old user needs to renew his/her decryption key because it has expired or has been compromised. The TTP first defines the access policy for the new user u. Which access policy assigning to each user strictly depends on the application. In the traffic monitoring example, users can be authorized to view videos from short or long paths, depending on whether they pay low or high dues. Once the TTP has defined the user's access policy $\mathcal{T}^{(u)}$, it creates the user's decryption key by executing:

$$DK^{(u)} = \text{KP.KeyGen}(MK, \mathcal{T}^{(u)}).$$

The TTP encrypts the message $(DK^{(u)}, ts, \text{Sign}(DK^{(u)}, ts))$, where ts is a timestamp, with the user's public key $K_{pub}^{(u)}$ and gives it to the user either directly or through the cloud server. The user verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the user accepts $DK^{(u)}$ as his/her decryption key. In the meanwhile, the TTP sends the message $(u, dk_{i\in\lambda^{(u)}\cap\mathcal{U}_\mathcal{R}}, ts, \mathrm{Sign}(u, dk_{i\in\lambda^{(u)}\cap\mathcal{U}_\mathcal{R}}, ts))$ to the cloud server, where $\lambda^{(u)}$ are the access policy attributes of the user. The cloud server verifies the TTP signature to be valid, and the timestamp to be fresh. If everything is correct, the cloud server stores the decryption key components $dk_{i\in\lambda^{(u)}\cap\mathcal{U}_\mathcal{R}}$.

Seal. This procedure is executed at the beginning of every time unit. For each sensor j, the TTP randomly generates a data encryption key $DEK^{(j)}$ and encrypts it by executing:

$$ASK^{(j)} = \mathrm{KP.Encrypt}(DEK^{(j)}, \gamma^{(j)}, Y, T_{i\in\gamma^{(j)}}),$$

thus obtaining the ABE-sealed key for the sensor j for the current time unit. The TTP sends the ABE-sealed key to the cloud server, which stores it. Also, for each sensor j the TTP encrypts the message $(DEK^{(j)}, ts, \mathrm{MAC}_{K_{LT}^{(j)}}(DEK^{(j)}, ts))$ with the long-term key $K_{LT}^{(j)}$, where ts is a timestamp and $\mathrm{MAC}_{K_{LT}^{(j)}}(\cdot)$ denotes a message authentication code keyed by $K_{LT}^{(j)}$. Such an encrypted message constitutes the sensor key material for the sensor j for the current time unit. The TTP sends the sensor key material for each sensor to the cloud server, which stores it. Each sensor j retrieves its sensor key material from the cloud server and verifies the message authentication code to be valid and the timestamp to be fresh. If everything is correct, the sensor accepts $DEK^{(j)}$ as its current data encryption key and initializes a *data encryption counter* $(c^{(j)})$ to zero. The sensor uses the data encryption counter to keep track of how many times it renews the data encryption key in the current time unit.

Note that the seal procedure is potentially a long operation for the TTP, since it requires the TTP to execute an KP.Encrypt primitive for each sensor. For this reason, setting a too short time unit (e.g., one second) could be infeasible. A reasonable choice for the time unit is one day.

Data Production. This procedure is executed every time a sensor j senses a new piece of data. First, the sensor encrypts the sensed data $SD^{(j)}$ with its current data encryption key $DEK^{(j)}$. The result, together with the current data encryption counter $c^{(j)}$, constitutes an encrypted sensed data $ESD^{(j)}$. The sensor sends the encrypted sensed data to the cloud server, which stores it. Then, the sensor computes a new data encryption key as a one-way hash of the old one: $DEK^{(j)} \leftarrow \mathrm{H}(DEK^{(j)})$. Finally, the sensor securely destroys the old data encryption key and increments its data encryption counter. This key renewal method prevents an adversary who compromises a sensor from decrypting old sensed data.

Note that sensors perform only lightweight symmetric-key encryption, thus we can employ constrained sensor devices such as battery-powered motes.

Data Consumption. This procedure is executed each time a user u wants to access a piece of data sensed by the sensor j. The user first sends a data request to the cloud server specifying which sensed data (s)he wants to access. On receiving the data request, the cloud server checks whether some of the stored decryption key components are out of date by checking their versions. For each out-of-date component dk_i, the cloud server updates it by executing:

$$dk_i' = \mathrm{PRE.UpdateDK}(i, dk_i, rk_i),$$

and it provides dk_i' to the user. If the decryption key component was outdated of more than one version, then the cloud server will execute the PRE.UpdateDK primitive multiple times, each with different re-encryption keys from the re-encryption key list RKL_i. After that, the cloud server checks whether some of the KP-ABE ciphertext components of the ABE-sealed key $ASK^{(j)}$ relative to the sensed data requested by the user are out of date. For each out-of-date component ct_i, the cloud server updates it by executing:

$$ct_i' = \mathrm{PRE.UpdateCT}(i, ct_i, rk_i),$$

and it replaces ct_i with the updated version. If the KP-ABE ciphertext component was outdated of more than one version, then the cloud server will execute the primitive PRE.UpdateCT multiple times, each with different re-encryption keys from the re-encryption key list RKL_i. Finally, the cloud server provides the user with the encrypted sensed data $ESD^{(j)}$, and the ABE-sealed key $ASK^{(j)}$ relative to the time unit during which the requested data was sensed. On receiving this, the user retrieves the data encryption key by executing:

$$DEK^{(j)} = \mathrm{H}^{c^{(j)}}\left(\mathrm{KP.Decrypt}(ASK^{(j)}, DK^{(u)})\right),$$

where $\mathrm{H}^n(\cdot)$ denotes the one-way hash applied n times. The user finally uses the data encryption key $DEK^{(j)}$ to decrypt the sensed data. Of course, if the user is not authorized to access the data, then the KP.Decrypt primitive will return $\perp$, so s(he) will not able to retrieve the data encryption key.

Key Revocation. Whenever a user u's decryption key must be revoked, for example when his/her key is compromised, the system executes the key revocation procedure to make the decryption key unable to decrypt any ABE-sealed key.

At first, the TTP determines the updatee set $\mu^{(u)} \subseteq \lambda^{(u)}$, i.e., a set of attributes without which the access policy will never be satisfied. The TTP must update such attributes in order to revoke the decryption key. In our system, the updatee set $\mu^{(u)}$ is formed by the attributes $\lambda^{(u)} \cap \mathcal{U}_{\mathcal{R}}$. This is because we use the whole attribute set $\mathcal{U}_{\mathcal{X}}$ in place of the dummy attribute used in YWRL scheme. For each attribute $i \in \mu^{(u)}$, the TTP updates the related quantities by executing:

$$(t_i', T_i', rk_i) = \text{PRE.UpdateAtt}(i, MK),$$

and it replaces t_i and T_i with the updated versions. Then, the TTP sends the message $(u, rk_{i,\forall i \in \mu^{(u)}}, ts, \text{Sign}(u, rk_{i,\forall i \in \mu^{(u)}}, ts))$ to the cloud server, where ts is a timestamp. The cloud server verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the cloud server adds each re-encryption key rk_i to the proper RKL_i and discards all the decryption key components $dk_{i \in \lambda^{(u)} \cap \mathcal{U}_{\mathcal{R}}}$ related to the revoked user u. Finally, the TTP executes a seal procedure restricted to those sensors whose at least one KP-ABE ciphertext attribute is in $\mu^{(u)}$, i.e., for those sensors which produce sensed data that the revoked key was authorized to decrypt. We refer to this set of sensors as *affected sensors*, meaning that they are "affected" by a key revocation. The decryption key is now revoked and is not anymore capable of decrypting any ABE-sealed key.

Note that the revocation of a decryption key is a burdensome operation for ABE systems, and it usually causes side effects. Indeed, the decryption keys of all the users sharing at least one attribute in $\mu^{(u)}$ must be updated in order to decrypt new ABE-sealed keys. We refer to this set of users as *affected users*, meaning that they are "affected" by a revocation of the key of another user. A decryption key update is carried out by the cloud server, as described in the data consumption procedure. Another side effect of the revocation of a decryption key is the necessity of generating new ABE-sealed keys with the newest version of the attributes in order to make the decryption key ineffective to decrypt any ABE-sealed key. The generation of the new ABE-sealed keys is carried out by the TTP by means of a seal procedure for the affected sensors, as described earlier.

Adversary Model(s) and Security Analysis

We distinguish three adversaries differing on their capabilities: (i) someone capable of eavesdropping and injecting traffic between the sensors, the cloud server, and the TTP; (ii) someone capable of compromising one or more sensors; and (iii) someone capable of carrying out a data exfiltration by exploiting some cloud server vulnerability. All these adversaries aim at accessing sensed data without authorization.

The adversary capable of eavesdropping and injecting traffic could either try to steal decryption keys or inject malicious sensor key material to make the sensors use a compromised data encryption key. However, none of these tactics is viable. Stealing decryption keys is infeasible since the TTP sends them to new users in encrypted form. Injecting malicious sensor key material to a sensor is infeasible too since it is authenticated with the long-term key shared between the sensor and the TTP.

The adversary capable of compromising a sensor can compromise the long-term key and the current data encryption key. However, he can only read sensed data produced by the sensor *after* the compromise, but not that produced before. Indeed, each sensor renews the data encryption key at each new piece of produced data, and the old data encryption key is securely destroyed. The new data encryption key is computed as a one-way hash of the old one, thus it is infeasible to recover the old data encryption keys.

The adversary capable of performing data exfiltration can steal all the encrypted sensed data, the ABE-sealed keys, the sensor key material, and the re-encryption keys. Nevertheless, he cannot access sensed data because it is encrypted. If he owns a decryption key, then he can access only the sensed data that such a decryption key is authorized to decrypt. If the decryption key has been revoked, the adversary can update it by means of the re-encryption keys. Then, he can use such an updated decryption key to decrypt sensed data, even if it has been already re-encrypted by the cloud server. In other words, he can "rollback" the key revocation. As a mitigation measure, the cloud server may store the re-encryption keys in a protected memory space, accessible only by high-privileged processes. On the other hand, sensor key material, encrypted sensed data, and ABE-sealed keys can be accessible by low-privileged processes. As a consequence of this, the process that communicates with sensors and users could run with low privileges, whereas the process that communicates with the TTP should run with high privileges. Supposing that the adversary exploits the former process, which is the more exposed one, he cannot access the re-encryption keys unless he performs privilege escalation. Another mitigation measure is to securely delete old re-encryption keys. Before deleting a re-encryption key, the cloud server must update all the ciphertexts and the decryption key components that use the related attribute. Of course, this can be done without involving the users.

Note that we are interested only in adversaries that aim at illegally accessing sensed data. Other kinds of adversaries, like someone that injects bogus sensed data into the system or someone that performs a denial of service, fall outside the scope of this work. In case that the system must defend also against such adversaries, additional defense techniques must be implemented. An adversary that injects bogus sensed data can be counteracted by data authentication mechanisms, for example by accompanying sensed data with Message Authentication Codes (MACs). An adversary that performs a denial of service can be counteracted by IDS/IPS techniques.

3.4 Universe of the Attributes and Access Policies

In this section we introduce different attribute representations of the street network, resulting in different system performance especially in terms of affected users and *key size*, i.e., the space needed to store a decryption key.

Road Segments Representation: Universe Subset $\mathcal{U}_{\mathcal{R}}$

Basic Representation

A basic and naive representation consists of mapping one road segment onto one attribute. In this case, the $\gamma_{\mathcal{R}}^{(j)}$ set of each ABE-sealed key is formed by just a single attribute, whereas the $\mathcal{T}_{\mathcal{R}}$ subtree of a decryption key is an OR between many attributes, each one representing a road segment the user is authorized to monitor. We will refer to this implementation as the *basic representation*.

Segment-Tree Representation

The basic representation is very simple to implement, but it has numerous drawbacks. First of all, the number of access policy attributes of a decryption key, and thus the key size, grows linearly with the number of authorized road segments. Moreover, the revocation of a decryption key affects every user whose decryption key is authorized to access *any* road segment that the revoked key was authorized to access. For example, the revocation of a key authorized to access an entire street will affect all the users authorized to access any subset of road segments of that street. Indeed, by construction, in the basic representation, each road segment is mapped onto one attribute. This causes many affected users for each key revocation.

Segment trees can help us to reduce the affected users. In the following, we introduce a representation based on segment trees which aims at limiting the number of users affected by a key revocation, thus making the key revocation more efficient. Let us consider a generic street with ρ road segments, and let us denote the road segments of that street with the identifiers from 1 to ρ. We build a segment tree in which the leaves are the road segment identifiers. Each node of the segment tree is represented by an attribute. The $\gamma_{\mathcal{R}}^{(j)}$ set of each ABE-sealed key is formed by the point representation set of the sensor j's road segment, and it contains $\mathcal{O}(\log \rho)$ attributes. We use a point-in-interval access policy to authorize the user to access a subset of consecutive road segments of the street. The complete $\mathcal{T}_{\mathcal{R}}$ subtree of the access policy will be an OR between multiple point-in-interval access policies, one for each street the user is authorized to access. Figure 3.5 shows a graphic example of this. The key is authorized to access consecutive road segments belonging to two streets, namely *STREET10* and *STREET56*. The subtree $s10\_l_5 \vee s10\_n_2$ is a point-in-interval access policy which allows the user to access a set of consecutive

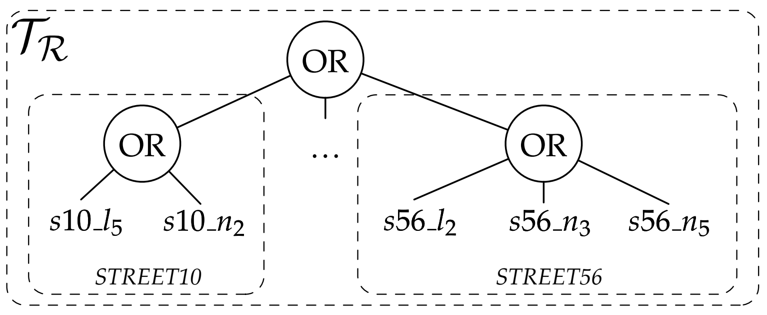


Figure 3.5: Example of $\mathcal{T}_{\mathcal{R}}$ subtree of the key access policy with segment-tree representation.

road segments of the street *STREET10*. We will refer to this implementation as the *segment-tree representation*.

Note that, with respect to the basic representation, the TTP is required of a little more effort to produce the ABE-sealed keys since the number of the KP-ABE ciphertext attributes for each street is $\mathcal{O}(\log \rho)$ instead of 1, and the complexity of the KP.Encrypt primitive depends on such a number. However, the segment-tree representation sensibly reduces the number of affected users. Indeed, a user authorized to access a road segment in common with the revoked key is not automatically affected, as it happens in the basic representation. Moreover, the segment-tree representation reduces the key size, because the number of access policy attributes is always less than or equal to the number of authorized road segments.

Attribute-Pool Representation

We introduce a third representation which extends the segment-tree representation to further lower the number of users affected by a key revocation. We replace each attribute i in the universe subset $\mathcal{U}_{\mathcal{R}}$ of the segment-tree representation with a pool of ε attributes $\{i_\omega\}, \forall \omega \in [1, \varepsilon]$ (*replicas*). Each replica i_ω in the pool has the same meaning.

The $\gamma_{\mathcal{R}}^{(j)}$ set of an ABE-sealed key is composed of ε point representation sets and includes $\mathcal{O}(\varepsilon \log \rho)$ attributes, and the size of a decryption key is the same as in the segment-tree representation. Indeed, the structure of the $\mathcal{T}_{\mathcal{R}}$ subtree is equivalent to the one of the segment-tree representation, but each attribute in the subtree is one of the ε replicas. We will refer to this implementation as the *attribute-pool representation*.

To implement this representation, for each replica in $\mathcal{U}_{\mathcal{R}}$ the TTP maintains the number nk_{i_ω} of non-revoked keys using that replica. nk_{i_ω} is incremented when the TTP issues a new decryption key having i_ω as access policy attribute, and it is decremented when a similar key is revoked. When the TTP executes a key distribution procedure, for each attribute which forms the $\mathcal{T}_{\mathcal{R}}$ subtree, it chooses the *least frequently used replica* of the attribute, i.e., $i_{\widetilde{\omega}}$ such that $\widetilde{\omega} = \arg\min_\omega (nk_{i_\omega})$. This is to minimize the number of affected users in case the newly generated key is revoked.

This representation further reduces the number of affected users due to a key revocation with respect to the segment-tree representation. Indeed, two users authorized to access the very same set of road segments may have no attributes in common in their decryption keys if they have different replicas of each attribute. In such a case, if one of them is revoked, the other will not be affected.

Time Representation: Universe Subset $\mathcal{U}_\chi$

In ABE-Cities, users can be authorized to access only data sensed within a specific validity period. A "not-before" and a "not-after" times are embedded in each decryption key, in a similar way that X.509 certificates do. Both times are represented with the maximum resolution of the time unit. Let the *maximum system lifetime* be the maximum time of operation of the system. We represent the time units from 1 to χ, where $\chi = \lceil \text{maximum system lifetime / time unit} \rceil$, by means of a segment tree. A basic representation is possible and consists in mapping each time unit onto one attribute. However, with this representation the number of access policy attributes of a decryption key grows linearly with the number of time units in the validity period. This would produce possible huge-sized keys. To avoid this we adopted a segment-tree representation also for time. The $\gamma_\chi^{(j)}$ set of each ABE-sealed key is formed by the point representation set of the data production time unit, which is $\mathcal{O}(\log \chi)$ attributes. On the other hand, the validity period of a decryption key is implemented through a point-in-interval access policy, which forms the $\mathcal{T}_\chi$ subtree and is, again, $\mathcal{O}(\log \chi)$ attributes. The segment-tree representation reduces the key size, because the number of access policy attributes is always less than or equal to the number of time units in the validity period. Note that using an attribute-pool representation would not improve the key revocation performance in this case, since these attributes are never updated.

3.5 Experimental Evaluation

We tested our scheme by simulations, using a street network representing the city of Pisa[2] obtained from OpenStreetMap. We assumed a maximum system lifetime of 100 years and the time unit equal to one day. Then, we considered a dataset of 300 users with randomly chosen 365-day overlapping validity periods. In our simulations, every user is authorized to access a *route* composed of consecutive road segments. A route is characterized by a *route length* L, which is the line-of-sight distance between its source point and its destination point. We chose a source point at random within the map and a destination point at random at a distance L from the

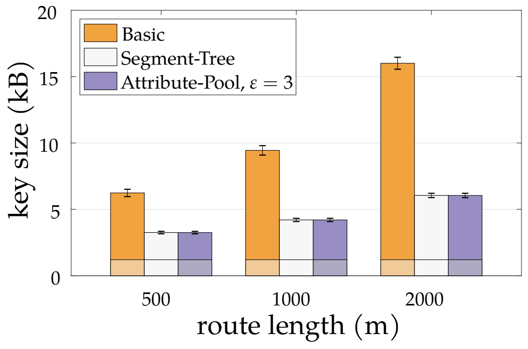
[2]Lat: [43.7052, 43.7264], Lon: [10.3867, 10.4266].

source point. Within each scenario we tested, we fixed a value for the route length which is the same for all the users. We generated a decryption key for each user, by using an implementation of GPSW scheme written in the C language (Zheng, 2011), which realizes the four ABE primitives described in Section 3.2 with 80-bit of security strength.

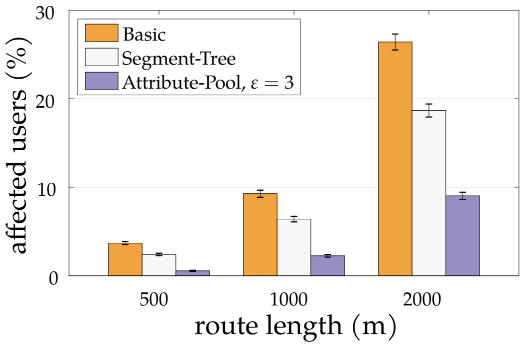
In Figure 3.6a we show a comparison between the average key sizes for the three road segments representations, with respect to the route length. The lower part of the bars shows the portion of the key size concerning the $\mathcal{T}_{\mathcal{X}}$ subtree. With all the three representations, a user's device, e.g., a smartphone, can easily store a decryption key of a few kilobytes. The cloud server stores only the decryption key components of the $\mathcal{T}_{\mathcal{R}}$ for all the users, and the total size for our dataset of 300 keys is about 1.4 MB using either the segment tree or the attribute-pool representation with a route length of 2000 m. The key size of the segment tree and the attribute-pool representations are about a third compared to the basic representation. Nevertheless, the keys are fairly small for all the three representations (< 16 kB in the worst case), so the key size should be affordable for both the users and the cloud server.

By using the same dataset, we revoked each decryption key in turn and measured how many users were affected. In Figure 3.6b we show the average percentage of affected users by a single key revocation. As expected, the segment tree and the attribute-pool representations show less affected users than the basic representation. This holds for all the route lengths tested. Figure 3.6c shows the affected users with respect to the number of replicas in the attribute-pool representation.
It is evident from the figure that the affected users can be reduced even further by increasing the parameter ε. The segment-tree representation and the attribute-pool representation are capable of reducing the number of affected users, and this lightens the load for the cloud server due to a key revocation, which is also distributed over time thanks to the laziness of proxy re-encryption.

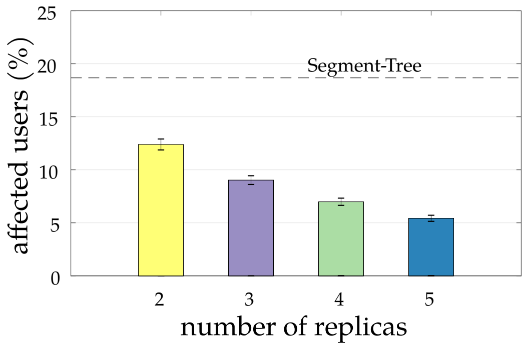
In our system, the load on the TTP is mainly determined by the complexity of the KP.Encrypt primitive, which grows linearly with the KP-ABE ciphertext attributes. The TTP executes such primitive once for each sensor at the beginning of the seal procedure. We experimentally evaluated the computational load and the required bandwidth on the TTP for the seal procedure. We used the same street network representing the city of Pisa and a maximum system lifetime of 100 years. We assumed 100 sensors deployed randomly on the street network. We used the attribute-pool representation with $\varepsilon = 5$, which represents the worst considered case in terms of KP-ABE ciphertext attributes, and thus the worst case for the computational load and the required bandwidth on the TTP. We define the *key sealing time* as the time needed to generate all the ABE-sealed keys at the beginning of the seal procedure. We observed a key sealing time of 5.4 seconds, and the total size of the ABE-sealed keys was about 550 kB. Note that the seal procedure is executed within the key revo-



(a) Average key size wrt route length.



(b) Average percentage of affected users due to a single key revocation wrt route length.



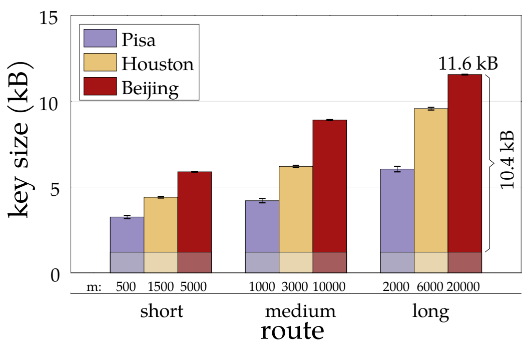
(c) Average percentage of affected users due to a single key revocation wrt number of replicas in the attribute-pool representation. $L = 2000$ m.

Figure 3.6: Performance evaluation. All the plots show 95 %-confidence intervals in error bars.

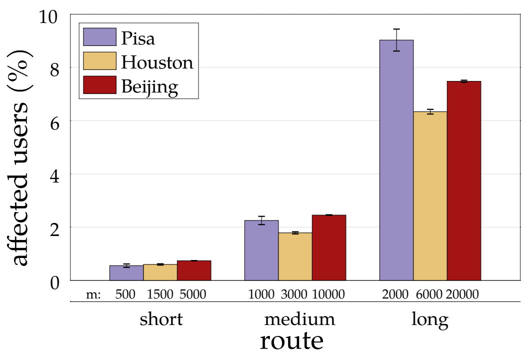
cation procedure too, so the key sealing time represents also the minimum time that the TTP takes to revoke a decryption key. We evaluated the key sealing time on a desktop computer equipped with 16 GB of RAM, an Intel® Core™ i5-6600 CPU, and running Ubuntu 16.04.3 LTS 64-bit operating system. From our tests, the KP-ABE ciphertext attributes γ of the average ABE-sealed key are about 39, of which about 21 those of $\gamma_{\mathcal{R}}$. In other words, the TTP is asked to carry out a task of a few seconds and then send a few kilobytes of data to the cloud server every day. This load should be affordable by the average TTP.

	Pisa	**Houston**	**Beijing**
short route	500 m	1500 m	5000 m
medium route	1000 m	3000 m	10 000 m
long route	2000 m	6000 m	20 000 m

Table 3.1: Route lengths.



(a) Average key size wrt route length.



(b) Average percentage of affected users due to a single key revocation wrt route length.

Figure 3.7: Performance evaluation on large cities.

Experimental Evaluation on Large Cities

In order to prove the scalability of our system, we performed the same simulations on two large cities, namely Houston[3] and Beijing[4]. We illustrate the effects of revocation in terms of affected users, and we report results which show the average key size in these new scenarios as the length of the routes varies. The area of these cities is about 10 and 100 times the area of Pisa, respectively. Accordingly, we considered proportional datasets of users for Pisa (300 users), Houston (3000 users), and Beijing (30 000 users). Moreover, we considered route lengths roughly proportional to the square root of the city area. For each city, we thus defined "short route", "medium route", and "long route" a route whose length is as specified by Table 3.1. Again, we assumed a maximum system lifetime of 100 years, a time unit equal to one day, and 365-day validity periods. As we outlined in Section 3.5, the attribute-pool representation guarantees the best performance with regards to affected users. Therefore, simulations in this section take into consideration only such a representation with $\varepsilon = 3$.

In Figure 3.7a we show the average key size for the three route lengths tested on each city. The lower part of the bars shows the portion of the key size concerning the $\mathcal{T}_{\mathcal{X}}$ subtree. Of course, for each city, the key size increases with the route length

[3]Lat: [29.7171 , 29.7975], Lon: [−95.4145 , −95.3208].
[4]Lat: [39.7705 , 40.0271], Lon: [116.2251 , 116.5581].

since the set of authorized road segments becomes larger. However, the graph in Figure 3.7a reveals that the key size is influenced by distinctive features of each city's street network. In other words, street network properties, such as the average road segment length, the average number of road segments within a street, etc., can impact on the key size. Among these properties, we found the *average road segment length* (Table 3.2) to be the most influential in the outcomes.

Pisa	**Houston**	**Beijing**
22.4 m	34.6 m	66.4 m

Table 3.2: Average road segment length.

In particular, longer road segments produce smaller keys, since the same route length is composed of fewer road segments. For example, with reference to Figure 3.7a, the medium route in Beijing produces a key smaller than the long route in Houston, even if the former is longer than the latter (10 000 m versus 6000 m). The reason is that in Beijing the average road segment is 66.4 m long, whereas in Houston it is 34.6 m long. As a consequence, a route in Beijing is composed of fewer road segments than a route of the same length in Houston, thus resulting in a smaller key. The average road segment length is an indirect measurement of the "density" of the street network since the road segment is basically the distance between two crossroads, so the shorter the distance is, the higher the number of crossroads in the same area. To sum up, small but "dense" cities like Houston results in larger keys with respect to big but "sparse" cities like Beijing.

From Figure 3.7a it can be seen that, even in the worst case considered, i.e., Beijing with routes of 20 000 m, the average key is about 11.6 kB, which should be affordable for the majority of the users. Such a small average key is also advantageous to the cloud server, which must store all the key components associated with the $\mathcal{T}_{\mathcal{R}}$ subtree for all the users. Considering that the average size of the $\mathcal{T}_{\mathcal{R}}$ subtree key components is about 10.4 kB, and that the Beijing scenario counts 30 000 users, the cloud server must dedicate only 311 MB for the decryption key components, which again should be affordable. This corroborates the scalability of ABE-Cities in terms of key size.

In Figure 3.7b we show the average percentage of affected users due to a revocation. As expected, this percentage grows with the route length in all the three cities. The reason is that longer routes are more likely to go through the same roads, which in turn makes collisions between users more probable. With many user collisions, a revocation of a user causes a high number of affected users. Interestingly, the small city of Pisa scales particularly bad in terms of affected users as the route length increases. The reason is attributable to the medieval origins of Pisa, whose street network tends to canalize the traffic on a few main roads, instead of spreading it on the whole city. This makes collisions between users extremely probable as the route grows in length. Nonetheless, for each city and route length tested, the average percentage of affected users never exceeds 10 %. The worst scenario in terms of absolute number of users affected by a revocation is Beijing with long routes, which gives $7.48\,\% \cdot 30\,000 = 2244$ affected users. This number of affected users should be bearable by the average cloud server, also thanks to the laziness of the mechanism. Indeed, considering that the cloud server updates the key components only upon a data request by the affected user, its computational load is largely spread over time. On the contrary, the computational load of the TTP is sensibly affected by the size of the city. We will thoroughly examine this aspect in Section 3.7.

3.6 Advanced ABE-Cities

In the ABE-Cities scheme described in Section 3.3, the key sealing time grows roughly linearly with the sensors deployed in the city. This is because the TTP executes the KP.Encrypt primitive once for each sensor. Depending on the city size, the number of sensors in the city can be large, and this might overload the TTP. Moreover, the TTP may be a single point of failure of the whole system. If the TTP misses or delays the execution of a seal procedure for some reason (e.g., a technical failure), the sensors cannot upload data to the cloud server any longer. Indeed, they cannot execute the data production procedure since they lack the new sensor key material. In this section we introduce a more advanced scheme which reduces the computational load of the TTP and mitigates the effects of TTP downtime, resulting in a briefer seal procedure.

The advanced ABE-Cities scheme leverages full-resource *gateways*, often deployed in IoT systems, through which the sensors access the Internet. Gateways can execute the KP.Encrypt primitive on behalf of the TTP and in parallel with it, thus reducing the execution time of the seal procedure. This raises the problem of distributing authentic copies of many public parameters to a potentially large set of gateways, which we address by using Merkle trees.

A *Merkle tree* is a binary tree used to efficiently verify the integrity of large data sets or portions of them. Assume that a large data set is partitioned into *n data blocks*. Each data block corresponds to a leaf in the Merkle tree. The label of each leaf (*leaf label*) is a one-way hash of the corresponding data block. The label of each non-leaf node (*non-leaf label*) is a one-way hash of the concatenation of its child nodes' labels. The root label is called *top hash* (t_H). The integrity of the top hash implies, of course, the integrity of all the data blocks. To verify the integrity of a subset of data blocks, it is sufficient to retrieve the labels of the minimal set of nodes covering the other data blocks (*covering hash set*, $\overline{\eta}$), and to have a proof of integrity of the top hash, which can be for example a digital signature on it. In the average case, the covering hash set is only $\mathcal{O}(\log n)$, thus the amount of information required to verify the integrity is little compared to the complete set of data blocks. Figure 3.8 shows an example of a Merkle tree.

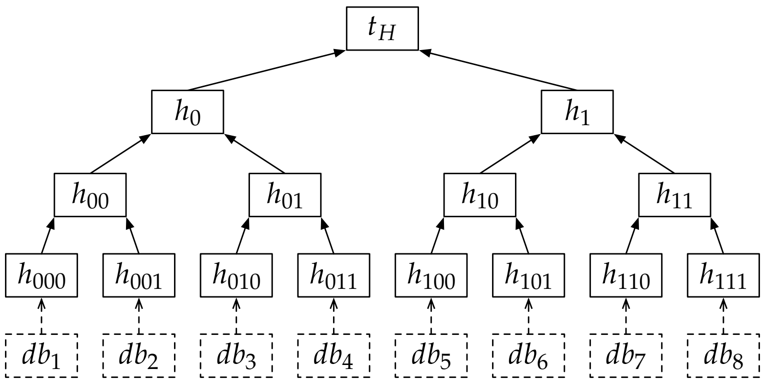


Figure 3.8: Example of Merkle tree with eight data blocks.

With reference to such an example, if we want to verify the integrity of db_1, db_2 and db_4, we just need to retrieve such data blocks, the set $\overline{\eta} = \{h_{010}, h_1\}$, and the proof of integrity of t_H.

The architecture of the advanced ABE-Cities scheme is shown in Figure 3.9. We assume that a sensor can access the cloud server either through a gateway (*networked sensor*), for example through a Wi-Fi connection, or directly (*direct sensor*), for example through a Narrowband IoT connection. A sensor j shares a long-term key $K_{LT}^{(j)}$ either with its gateway, in the case is a networked sensor, or the TTP, in the case is a direct sensor. Since gateways execute the KP.Encrypt primitive, they need to receive authentic copies of the public parameters from the TTP. The TTP uses a Merkle tree to transmit different subsets of public parameters to different gateways in an authenticated manner. This permits the gateways to verify the public parameters authenticity by receiving only a few data. Each gateway must receive the public parameters corresponding to the union of the attributes $\gamma_{\mathcal{R}}^{(j)}$ identifying the road segments of its networked sensors. We call such a set $\sigma_{\mathcal{R}}^{(g)} = \bigcup \gamma_{\mathcal{R}}^{(j)}$ for all the networked sensors j connected to the gateway g. Also, each gateway must receive all the public parameters in $\mathcal{U}_{\mathcal{X}}$.

System Procedures

With respect to the procedures of the vanilla ABE-Cities scheme explained in Section 3.3, key distribution, data production, and data consumption procedures remain unchanged. In the following, we describe the new version of setup, seal, and key revocation procedures, and we introduce a new one, namely the gateway join procedure.

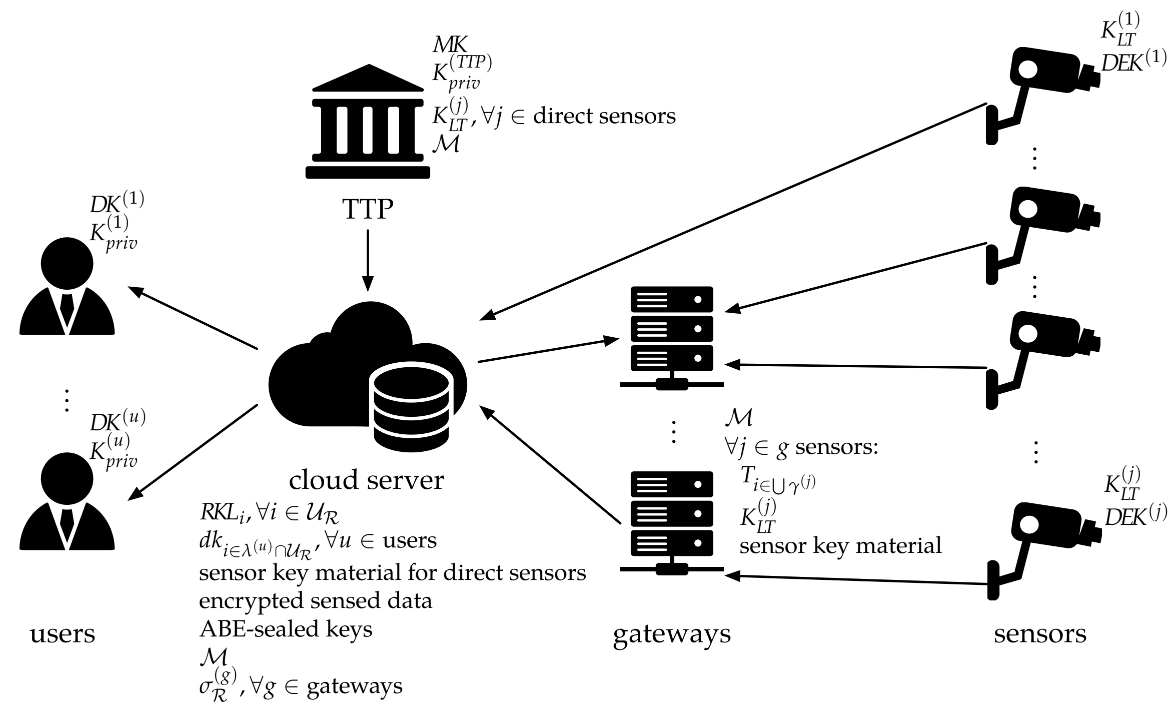

MK
$K_{priv}^{(TTP)}$
$K_{LT}^{(j)}, \forall j \in$ direct sensors
$\mathcal{M}$
TTP
$DK^{(1)}$
$K_{priv}^{(1)}$
$DK^{(u)}$
$K_{priv}^{(u)}$
users
cloud server
$RKL_i, \forall i \in \mathcal{U}_{\mathcal{R}}$
$dk_{i \in \lambda^{(u)} \cap \mathcal{U}_{\mathcal{R}}}, \forall u \in$ users
sensor key material for direct sensors
encrypted sensed data
ABE-sealed keys
$\mathcal{M}$
$\sigma_{\mathcal{R}}^{(g)}, \forall g \in$ gateways
$\mathcal{M}$
$\forall j \in g$ sensors:
$T_{i \in \bigcup \gamma^{(j)}}$
$K_{LT}^{(j)}$
sensor key material
gateways
$K_{LT}^{(1)}$
$DEK^{(1)}$
$K_{LT}^{(j)}$
$DEK^{(j)}$
sensors

Figure 3.9: Advanced ABE-Cities architecture.

Setup. This procedure initializes the system. After having executed the setup procedure of Section 3.3, the TTP creates a Merkle tree $\mathcal{M}$ where data blocks are the public parameters $T_{i \in \mathcal{U}_{\mathcal{R}}}$ and signs the top hash t_H. Then, the TTP sends the message $(T_{i \in \mathcal{U}}, ts, \text{Sign}(t_H || T_{i \in \mathcal{U}_{\mathcal{X}}} || ts))$ to the cloud server, where ts is a timestamp. The cloud server re-constructs the Merkle tree with the received public parameters and verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the cloud server accepts $T_{i \in \mathcal{U}}$ as authentic public parameters and stores the $\mathcal{M}$ data structure.

Gateway Join. This procedure is executed every time a new gateway g joins the system. We assume the gateway knows the public key of the TTP, through which it can verify TTP signatures. The gateway communicates to the TTP which road segments its networked sensors are placed onto. The TTP determines the attribute set $\sigma_{\mathcal{R}}^{(g)}$ and sends the message $(\sigma_{\mathcal{R}}^{(g)}, ts, \text{Sign}(t_H || T_{i \in \mathcal{U}_{\mathcal{X}}} || T_{i \in \sigma_{\mathcal{R}}^{(g)}} || ts))$ to the cloud server. Then, the cloud server sends the message $(\sigma_{\mathcal{R}}^{(g)}, T_{i \in \sigma_{\mathcal{R}}^{(g)}}, \overline{\eta}^{(g)}, T_{i \in \mathcal{U}_{\mathcal{X}}}, ts,$ $\text{Sign}(t_H || T_{i \in \mathcal{U}_{\mathcal{X}}} || T_{i \in \sigma_{\mathcal{R}}^{(g)}} || ts))$ to the gateway, where $\overline{\eta}^{(g)}$ is the covering hash set of the public parameters $T_{i \in \sigma_{\mathcal{R}}^{(g)}}$ and ts is a timestamp. The gateway re-constructs the Merkle tree $\mathcal{M}$ and verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the gateway accepts $T_{i \in \sigma_{\mathcal{R}}^{(g)}}$ and $T_{i \in \mathcal{U}_{\mathcal{X}}}$ as authentic public parameters and will use them to generate ABE-sealed keys during the seal procedure.

Seal. This procedure is executed at the beginning of every time unit. First, The TTP executes the seal procedure of Section 3.3 only for the direct sensors. As for networked sensors, each gateway g randomly generates a data encryption key $DEK^{(j)}$ for each networked sensor j and encrypts it by executing:

$$ASK^{(j)} = \text{KP.Encrypt}(DEK^{(j)}, \gamma^{(j)}, Y, T_{i \in \gamma^{(j)}}),$$

thus obtaining the ABE-sealed key for the networked sensor j for current time unit. Also, the gateway g encrypts with the long-term key $K_{LT}^{(j)}$ the message $(DEK^{(j)}, ts,$ $\text{MAC}_{K_{LT}^{(j)}}(DEK^{(j)}, ts))$, where ts is a timestamp and $\text{MAC}_{K_{LT}^{(j)}}(\cdot)$ denotes a message authentication code keyed by $K_{LT}^{(j)}$. Such an encrypted message constitutes the sensor key material for the networked sensor j for the current time unit. The gateway g forwards the sensor key material to its networked sensors. Finally, the gateway securely destroys the data encryption keys and the sensor key materials. This prevents an adversary who compromises the gateway from decrypting sensed data. Each networked sensor verifies the message authentication code to be valid and the timestamp to be fresh. If everything is correct, the networked sensor accepts $DEK^{(j)}$ as its current data encryption key and initializes the data encryption counter $(c^{(j)})$ to zero.

Key Revocation. This procedure is executed every time a user u's decryption key must be revoked. First, the key revocation procedure described in Section 3.3 is executed, except for the final seal procedure for the affected sensors. The TTP generates then a new Merkle tree $\mathcal{M}'$ where data blocks are the latest version of the public parameters $T_{i \in \mathcal{U}_\mathcal{R}}$. Then, the TTP sends the message $(T_{i \in \mu^{(u)}}, ts, \text{Sign}(t'_H || ts))$ to the cloud server, where $\mu^{(u)}$ is the set of attributes updated during the revocation procedure of the vanilla scheme and ts is a timestamp. The cloud server re-constructs the Merkle tree with the latest version of the public parameters and verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the cloud server stores the $\mathcal{M}'$ data structure. Then, the cloud server identifies all the *affected gateways*, which are those gateways whose $\sigma_\mathcal{R}^{(g)}$ set includes at least one attribute of the revoked key. These gateways are "affected" by the key revocation because the public parameters they use to produce ABE-sealed keys must be updated. The cloud server sends the message $(T_{i \in \sigma_\mathcal{R}^{(g)} \cap \mu^{(u)}}, \overline{\eta}^{(g)\prime}, ts, \text{Sign}(t'_H || ts))$ to each affected gateway. Each affected gateway g re-constructs the Merkle tree and verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the gateway accepts $T_{i \in \sigma_\mathcal{R}^{(g)} \cap \mu^{(u)}}$ as authentic public parameters. Finally, a seal procedure is executed for all the affected sensors. Note that the Merkle tree allows the cloud server to send to each affected gateway only the public parameters it needs, and the affected gateway to accept the parameters by verifying only one signature.

Gateway Compromise

The advanced scheme inherits the security properties of the vanilla scheme with respect to the threats examined in Section 3.3. However, since gateways are now part of the system, we have to take into account the possibility of their compromise.

An adversary capable of compromising a gateway can compromise the long-term keys of all its networked sensors. However, the gateway securely destroys the data encryption keys and the sensor key materials during the seal procedure. Hence, the adversary will be able to decrypt only sensed data produced *after* the next execution of the seal procedure.

3.7 Advanced ABE-Cities Evaluation

In the ABE-Cities scheme described in Section 3.3, the key sealing time grows roughly linearly with the sensors deployed in the city. Depending on the city size, the number of sensors in the city can be large, and this might overload the TTP. The advanced ABE-Cities scheme presented in Section 3.6 mitigates this problem by outsourcing part of the computational load to the gateways. In this section we validate this by means of experiments.

We used the same street networks representing the cities of Houston and Beijing, and, for all the simulations performed, we assumed a fixed portion (10 %) of the road segments to be covered by a sensor. The sensors were then partitioned in networked and direct. For each simulation we changed the percentage of networked sensors from 0 to 100 % in steps of 10 %. The road segments covered by sensors and which of these sensors are networked were randomly selected for each simulation. We partitioned each city in 100×100 m tiles, and we assumed to have a gateway at the center of each tile which is connected to all the networked sensors of the tile, if any. This implies gateways to have about 70 m of transmission range, which is realistic for the WiFi protocol. We used the attribute-pool representation with $\varepsilon = 5$ to represent the street network, a maximum system lifetime of 100 years, and a time unit equal to one day. This scenario fits a participatory sensing application in which sensors are owned by different companies or even private citizens, for example a traffic monitoring application with IP cameras.

Since the TTP and the gateways produce their ABE-sealed keys in parallel, the key sealing time will be the maximum between the time required to the TTP (*TTP key sealing time*) and that required to the gateways (*gateway key sealing time*). The gateway key sealing time will be in turn dominated by the most loaded gateway, which is the last one to complete the generation of the ABE-sealed keys. To measure the key sealing time for a city, we generated the ABE-sealed keys by using an implementation of GPSW scheme written in the C language (Zheng, 2011). In particular, we generated an ABE-sealed key for each direct sensor using a desktop computer which emulates the TTP, and an ABE-sealed key for each networked sensor of the most loaded gateway using a single-board computer which emulates the gateway. The desktop computer that emulates the TTP is equipped with 16 GB of RAM, an Intel® Core™ i5-6600 CPU, and running Ubuntu 16.04.3 LTS 64-bit operating system. The single-board computer that emulates the gateway is a Raspberry Pi 3 Model B+, equipped with off-the-shelf hardware (1 GB SRAM and ARM Cortex-A53 1.4 GHz) running Raspbian Jessie.

Figs. 3.10a and 3.10b show the TTP key sealing time and the gateway key sealing time for Houston and Beijing, respectively.

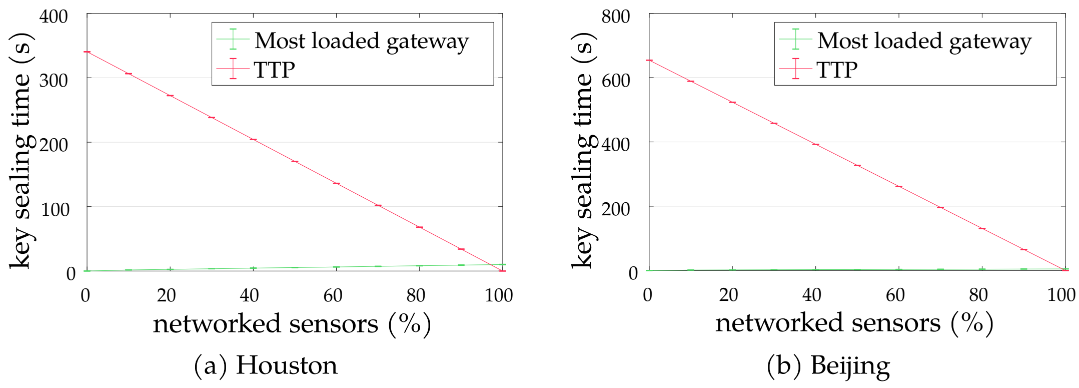
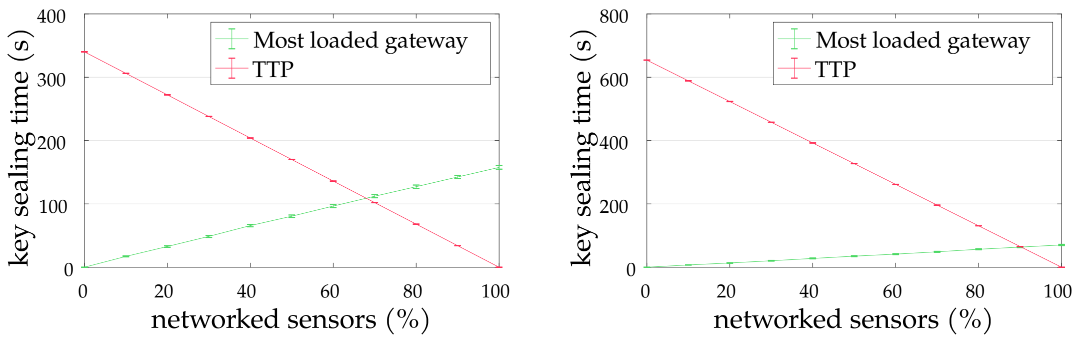


Figure 3.10: TTP key sealing time and gateway key sealing time wrt the percentage of networked sensors with 100 × 100 m-tile gateways. All the plots show 95 %-confidence intervals in error bars.

Remember that the key sealing time of the whole system will be the maximum between the two. From the graphs we can easily notice that, as the percentage of networked sensors increases, the TTP's load decreases whereas the gateways' load increases, both in a linear fashion. With 0 % of networked sensors, all the load is on the TTP, and we obtain the same performance of the vanilla scheme described in Section 3.3. The key sealing time of the vanilla scheme is 5 minutes and 40 seconds in Houston, and 10 minutes and 54 seconds in Beijing. With a greater percentage of networked sensors, the key sealing time drops noticeably. This shows that the advanced scheme mitigates the scalability issues of the non-advanced scheme in terms of TTP computational load. The point of intersection of the two curves represents the best case which minimizes the key sealing time. After that, the gateways are overloaded, and they become the bottleneck of the system. The best-case key sealing time is about 9.9 seconds in Houston with 97 % of networked sensors, and 4.9 seconds in Beijing with 99 % of networked sensors. Beijing is larger than Houston but reaches a better performance because it has more gateways, so the computational load is outsourced to a greater extent.

Note that the above scenario requires to have many gateways in the smart city, namely about 8100 in Houston and about 81 000 in Beijing. This is realistic for a participatory sensing application which involves private citizens' WiFi gateways. On the other hand, in case that private gateways could not be involved, deploying such a big number of gateways would be unfeasible. In this case, adopting a long-range and low-bitrate communication technology like LoRa is a more scalable solution in terms of deployment cost. However, fewer gateways means that the advanced scheme unburdens the TTP to a lesser extent. To quantify the key sealing time in such a scenario, we run again the simulations on the street networks representing the cities of Houston and Beijing. This time, we partitioned each city in 1×1 km tiles, and we assumed to have a gateway at the center of each tile which is connected to all the networked sensors of the tile, if any. This implies gateways to have about 700 m of transmission range, which is realistic for the LoRa protocol. This scenario fits applications in which the infrastructure must be cheap and the sensors produce little traffic, for example an air-quality monitoring application in which sensors produce few bytes per hour.

Figs. 3.11a and 3.11b show the TTP key sealing time and the gateway key sealing time for Houston and Beijing, respectively.



(a) Houston (b) Beijing

Figure 3.11: TTP key sealing time and gateway key sealing time wrt the percentage of networked sensors with 1×1 km-tile gateways.

As it can be seen from the graphs, the gateway key sealing time is sensibly higher than the one in the previous WiFi scenario. The reason is that each gateway handles many more networked sensors. As a consequence, the best-case key sealing times are worse too: about 109 seconds in Houston with 68 % of networked sensors, and about 64 seconds in Beijing with 90 % of networked sensors. Nevertheless, the key sealing time decreases noticeably when passing from 0 % of networked sensors (which represents the vanilla ABE-Cities performance) to a greater percentage. This shows that the advanced ABE-Cities scheme scales better than the vanilla one even in the LoRa scenario, and the seal procedure is still briefer.

To sum up, the decentralized nature of the advanced ABE-Cities scheme permits parallelization of the seal procedure and thus its completion in a considerably shorter time period, as outlined by the experimental results. Moreover, it unburdens the TTP of the entire effort required by the seal procedure by outsourcing a significant amount of computation to the gateways, avoids the TTP to be a single point of failure, and provides scalability.

Chapter 4

Improving KP-ABE Revocation Mechanism

It doesn't matter how long it takes if the end result is a good theorem.

John Tate

Since the first introduction by Goyal et al. (GPSW scheme), plenty of ABE schemes have been proposed. Such schemes mainly aim at reducing the computational cost of the original encryption/decryption primitives either by outsourcing part of them to trusted parties (Li et al., 2012; Touati et al., 2014; Touati and Challal, 2016; Zuo et al., 2018), or by redefining them, e.g., by avoiding using bilinear pairings (Yao et al., 2015; Ding et al., 2018) or by using new constructions (Waters, 2011). Other schemes (Yu et al., 2010a,b; Hur and Noh, 2010; Jahid et al., 2011; Xu and Martin, 2012; Fischer et al., 2019) attempt to add a key revocation mechanism, which is intrinsically not existent in ABE.

In ABE systems, a Trusted Third Party (TTP) generates *public parameters* for encryption and *decryption keys* for decryption. An encrypting entity is called (*data*) *producer*, while a decrypting entity is named (*data*) *consumer*. In practical applications, data producers store the encrypted data on some server, while data consumers retrieve ciphertexts from it. Often, the storage service is provided by a third party server, e.g., a *cloud server*, which is historically considered to be *honest-but-curious*, meaning that it honestly carries out the storage, communication, and computational tasks, but it is also interested in accessing the content of the encrypted data stored on it. Some other works, consider the cloud server breachable, meaning that an adversary can exfiltrate data stored on the cloud server.

As every public-key cryptosystem, an ABE system needs a key revocation mechanism for it to be used in practice. A decryption key must be revoked whenever it has been compromised, i.e., when someone other than the original data consumer is in possession of it. Revoking a key means making it unable to decrypt new ciphertexts anymore. In many ABE schemes, this is achieved by a system-wide update of the attributes present in the key to revoke for every entity in the system except for the revoked one. So, every decryption key using an attribute used also by the revoked key must get an update. This process is called *re-keying*, and the cloud server can be either partially or completely entrusted with it so that data confidentiality is still guaranteed. In other words, the cloud server is provided with some cryptographic quantities which let it update the decryption keys, but it does not know the decryption keys themselves, through which it could otherwise access the content of the encrypted data stored on it. However, a compromised key could be made publicly available by an adversary, but also, the fact that a key has been made public might be the reason why it should be revoked. In both cases, since the cloud is able to potentially update every key through re-keying, it could update the revoked key, thus *undoing the revocation*, and access the content of the encrypted data. Another similar threat comes from an adversary able to breach the cloud server and obtain the re-keying material present on it. Also in this case the revocation can be undone. With these types of attack, the system cannot guarantee data confidentiality any longer.

In this chapter, we propose a secure key revocation mechanism for KP-ABE systems able to ensure data confidentiality even if the cloud server comes into possession of a revoked key. In previous works (Yu et al., 2010a,b; Hur and Noh, 2010; Jahid et al., 2011; Xu and Martin, 2012; Fischer et al., 2019), this type of adversary is not taken into account, but the honest-but-curious paradigm should consider it since revoked keys might be made public anytime since their compromise.

The rest of this chapter is organized as follows. Section 4.1 introduces some background on KP-ABE and other related techniques that we use in our scheme. Section 4.2 describes our model, assumptions, and construction. Section 4.4 analyzes the security and the performances of our scheme.

4.1 Background

KP-ABE: GPSW scheme

In section 3.2 we modeled the primitives defined in GPSW scheme as black boxes since ABE-Cities simply uses them as a means of achieving the properties of confidentiality and access control. In this section we give a mathematical description of such primitives. Before describing them, we introduce some notation. Let $\mathbb{G}_1$ be a group whose group operation is efficiently computable. Let p be its prime order, and let g be a generator of $\mathbb{G}_1$. Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a bilinear map which is efficiently computable and has the properties of bilinearity and non-degeneracy. Let the access policy $\mathcal{T}$ be defined as a tree. Each node x of the tree is described by threshold gate of type k_x-of-n_x, where k_x is its threshold value, and n_x is the number of child nodes of x. All the leaf nodes represent attributes and are described by a threshold value $k_x = 1$.

$(MK_{KP}, PK_{KP}) = \text{KP.Setup}(\mathcal{U})$ Defined the universe of the attributes $\mathcal{U} = \{1, \ldots, n\}$, choose $y, t_1, \ldots, t_n$ uniformly at random in $\mathbb{Z}_p$. Then compute $T_1 = g^{t_1}, \ldots, T_n = g^{t_n}$ and $Y = e(g, g)^y$. Output the public parameters as $PK_{KP} = (Y, T_1, \ldots, T_n)$ and the master key as $MK_{KP} = (y, t_1, \ldots, t_n)$.

$CT = \text{KP.Encrypt}(M, \gamma, PK_{KP})$ To encrypt the message $M \in \mathbb{G}_2$ under the set of attributes γ (*encryption attributes*), choose a number s uniformly at random in $\mathbb{Z}_p$. Then compute $\hat{ct} = MY^s$, and for each attribute $i \in \gamma$ compute the *ciphertext component* $ct_i = T_i^s$. Output the *KP-ABE ciphertext* as $CT = (\hat{ct}, \{ct_i\}_{i \in \gamma})$.

$DK = \text{KP.KeyGen}(MK_{KP}, \mathcal{T})$ Let $\mathcal{T}$ be an access policy defined over a set of attributes λ (*access policy attributes*). The algorithm defines a polynomial q_x of degree $deg_x = k_x - 1$ for each node x, starting from the root node and proceeding in a top-down manner. For the root node ρ, set $q_\rho(0) = y$ and other deg_ρ random points to define it completely. Then, associate each child node x with a unique number index(x). Proceed for any other node x, setting $q_x(0) = q_{\text{parent(x)}}(\text{index}(x))$ and other deg_x random points to define the polynomial completely. The polynomials for the leaf nodes are defined only by $q_x(0)$. For each leaf node, define the *decryption key component* $dk_i = g^{\frac{q_x(0)}{t_i}}$, where the node x identifies the attribute i. Output the *decryption key* as $DK = \{dk_i\}_{i \in \lambda}$.

$M = \text{KP.Decrypt}(CT, DK)$ The algorithm proceeds in a bottom-up manner. For each attribute $i \in \gamma \cap \lambda$ compute $e(dk_i, ct_i) = e(g, g)^{q_x(0) \cdot s}$. Then, by polynomial interpolation at the exponent, compute $e(g, g)^{q_z(0) \cdot s}$, where z is a parent node. Proceeding in a bottom-up fashion, if the polynomial interpolation is feasible at each level of the access policy up to the root ρ, we obtain $e(g, g)^{q_\rho(0) \cdot s} = e(g, g)^{y \cdot s} = Y^s$. Output the message $M = \hat{ct}/Y^s$ if the access policy is satisfied, $\perp$ otherwise.

Proxy Re-Encryption for KP-ABE

In section 3.2 we modeled the primitives defined in YWRL scheme as black boxes since ABE-Cities simply uses them as a means of enabling key revocation. In this section we give a mathematical description of such primitives and redefine some notation.

Yu et al. extended GPSW construction to include a key revocation mechanism to the scheme. This is achieved by adding the features of *proxy re-encryption* and *re-*

keying. Proxy re-encryption allows a third party to re-encrypt KP-ABE ciphertexts without accessing the content itself, thus a honest-but-curious cloud server perfectly fits this role. The re-encryption of KP-ABE ciphertexts stored on the cloud server prevents a revoked key from decrypting them. However, this mechanism alone is not sufficient: after the re-encryption, other legitimate keys might not be able to decrypt those KP-ABE ciphertexts too. Hence, YWRL scheme implements also a re-keying mechanism for such keys, which is operated by the cloud server. For each decryption key, the cloud server is given all the decryption key components but one, the one related to a special attribute (*dummy attribute*, Att_D), which is ANDed at the root of every access policy. The dummy attribute is also present in every KP-ABE ciphertext. Without knowing the decryption key component dk_{Att_D} the cloud server cannot decrypt any KP-ABE ciphertext, but it can perform re-keying.

In YWRL scheme, every attribute i, except for the dummy attribute, is identified by a version number. The version of an attribute i is set to zero at setup time, and it is increased by one every time the attribute is updated. Re-encryption and re-keying are performed by means of *re-encryption keys* issued by the trusted third party. A re-encryption key $rk_{i^{(v)} \leftrightarrow i^{(v+1)}}$ is a cryptographic quantity able to update both a ciphertext component and a decryption key component from the version v to the version $v+1$. All the re-encryption keys for the attribute i are ordered and maintained in a dynamic structure called *re-encryption key list RKL_i*. Fig. 4.1 shows an example of it. When a new re-encryption key is issued, an element containing

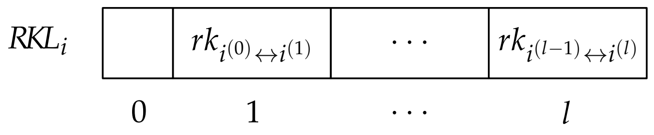

RKL_i
$rk_{i^{(0)} \leftrightarrow i^{(1)}}$
$\cdots$
$rk_{i^{(l-1)} \leftrightarrow i^{(l)}}$
0
1
$\cdots$
l

Figure 4.1: Re-encryption key list for the attribute i.

the latest re-encryption key is appended to the re-encryption key list. Note that the element at index v contains the re-encryption key to update either a ciphertext component or a decryption key component from the version $v-1$ to the version v. To create a new version of the attribute $i^{(v)}$, the TTP updates the component of the master key $t_i^{(v)}$ to $t_i^{(v+1)}$, and the component of the public parameters $T_i^{(v)}$ to $T_i^{(v+1)}$. Next, it issues a re-encryption key $rk_{i^{(v)} \leftrightarrow i^{(v+1)}}$ which can be used to transform either a ciphertext component $ct_i^{(v)}$ to $ct_i^{(v+1)}$ or a decryption key component $dk_i^{(v)}$ to $dk_i^{(v+1)}$. The scheme allows re-encryption and re-keying from *any* old version to the latest version. To simplify the notation, we avoid to indicate the version of the attributes when is not necessary. We will use i to indicate a generic version of the attribute, and i' to indicate the next version of it. The primitives of YWRL scheme have the following syntax:

$(t'_i, T'_i, rk_{i\leftrightarrow i'}) = \text{PRE.UpdateAtt}(i, MK_{KP})$ Choose t'_i uniformly at random in $\mathbb{Z}_p$ and compute $T'_i = g^{t'_i}$. Then compute the *re-encryption key* $rk_{i\leftrightarrow i'} = t'_i/t_i$. Update the component of the master key t_i to t'_i, and the component of the public parameters T_i to T'_i. Output the re-encryption key $rk_{i\leftrightarrow i'}$.

$ct_i^{(l)} = \text{PRE.UpdateCT}(i, ct_i, RKL_i)$ Obtain the current version of the attribute i from ct_i and locate the entry in RKL_i. If the selected entry is not the last one, retrieve all the entries from the next one to the last one. Compute $rk_{i\leftrightarrow i^{(l)}} = rk_{i\leftrightarrow i'} \cdot rk_{i'\leftrightarrow i''} \cdots rk_{i^{(l-1)}\leftrightarrow i^{(l)}} = t_i^{(l)}/t_i$. Next, compute $ct_i^{(l)} = (ct_i)^{rk_{i\leftrightarrow i^{(l)}}} = g^{t_i^{(l)} \cdot s}$. Output the updated ciphertext component $ct_i^{(l)}$.

$dk_i^{(l)} = \text{PRE.UpdateDK}(i, dk_i, RKL_i)$ Obtain the current version of the attribute i from dk_i and locate the entry in RKL_i. If the selected entry is not the last one, retrieve all the entries from the next one to the last one. Compute $rk_{i\leftrightarrow i^{(l)}} = rk_{i\leftrightarrow i'} \cdot rk_{i'\leftrightarrow i''} \cdots rk_{i^{(l-1)}\leftrightarrow i^{(l)}} = t_i^{(l)}/t_i$. Next, compute $dk_i^{(l)} = (dk_i)^{(rk_{i\leftrightarrow i^{(l)}})^{-1}} = g^{\frac{q_x(0)}{t_i^{(l)}}}$. Output the updated decryption key component $dk_i^{(l)}$.

Non-Monotonic Attribute-Based Encryption

Non-Monotonic Attribute-Based Encryption (NM-ABE) is an ABE scheme introduced by Ostrovsky et al. (2007) which supports non-monotonic access structures. This scheme extends GPSW scheme as it allows to include negated attributes in both access policies and attributes sets. In order to ease the reading, we abstract away the mathematical insight of the Ostrovsky et al.'s NM-ABE scheme. The interested reader can refer to Ostrovsky et al. (2007) for such details. We model the NM-ABE scheme with the following black-box primitives.

$(MK_{NM}, PK_{NM}) = \text{NM.Setup}(d)$ On input a fixed number d of attributes per NM-ABE ciphertext, this primitive initializes the scheme and outputs the public parameters PK_{NM} and the master key MK_{NM}.

$E = \text{NM.Encrypt}(M, \Gamma, PK_{NM})$ On input a message M, an encryption attribute set Γ, and the public parameters PK_{NM}, this primitive outputs the NM-ABE ciphertext E.

$NK = \text{NM.KeyGen}(\tilde{\mathbb{A}}, MK_{NM}, PK_{NM})$ On input an access policy $\tilde{\mathbb{A}}$ with access policy attributes Λ, the master key MK_{NM}, and the public parameters PK_{NM}, this primitive outputs an NM-ABE decryption key NK.

$M = \text{NM.Decrypt}(E, NK)$ On input an NM-ABE ciphertext E and an NM-ABE decryption key NK, this primitive outputs the message M if the access policy embedded in the key is satisfied with the attributes on the ciphertext. Otherwise, it outputs $\perp$.

Note that, differently from GPSW scheme, the NM-ABE one does not require the definition of a universe of the attributes when executing the setup primitive, but allows for the so-called "Large-Universe construction". This means that every string can be used as an attribute when executing encryption and key generation primitives.

4.2 Proposed Scheme

Our main idea is to create a robust revocation mechanism by which the honest-but-curious cloud server is not able to undo a key revocation. To this aim we use PRE techniques and distribute the task of keys and ciphertexts update to both the cloud server and the users. In particular, the update of either a decryption key component or a ciphertext component to a new version is accomplished by a double update through two different re-encryption keys, namely *cloud re-encryption key* and *user re-encryption key*. The first update uses the cloud re-encryption key and is executed by the cloud server. The second update uses the user re-encryption key and is executed by a legitimate user. In the following, we describe the system model and the assumptions, and then we present our construction.

Model and Assumptions

System Model

We assume that the system is composed of the following parties: a *data owner*, many *users*, and a *cloud server*. For the sake of simplicity we assume a single entity, i.e., the data owner, to be responsible for system setup, key issuing and data production. Nevertheless, the role of data owner could be played by a trusted third party and many data producers, as in Rasori et al. (2018). In our system the data owner produces the data, encrypts it, and stores it on the cloud server. Users are data consumers, and therefore, they hold decryption keys; they retrieve the data from the cloud server and decrypt it. The data owner can revoke a decryption key at any time. The cloud server is a server owned by a Cloud Service Provider supplying storage and computational capabilities. Basically, the cloud server can be seen as a provider of a PaaS service which runs software on behalf of the data owner. We assume the cloud server to have generous storage space capacity and computational resources. Moreover, we assume the cloud server can ensure high availability, so it can be accessed at any time by the data owner and the users. On the contrary, we do not make the same assumption for the other parties.

Security Model

Similarly to other works, e.g., Yu et al. (2010a); Touati and Challal (2016); Yao et al. (2015), we assume the cloud server to be honest but curious, meaning that it honestly carries out all its designated tasks, i.e., storage, communication, and computational tasks, but it is also interested in accessing the content of the encrypted data stored on it. Also, we assume the data owner to be fully trusted. On the contrary, we assume a user to be interested in accessing unauthorized data. To this aim, users can collude by trying somehow to combine their decryption keys in order to access unauthorized data which each decryption key is unable to decrypt singly. Moreover, we assume that each party owns a traditional public/private key pair, e.g., RSA or ECC keys, and that each public key can be easily retrieved by the other parties. We further assume secure communication channels among all the communicating parties, which can be obtained by using some secure protocol, e.g., TLS. Finally, we assume revoked keys to be publicly available since the revocation of a key is usually enforced when the key has been compromised, i.e., when someone other than the original user is in possession of it.

Definitions and Notation

Every piece of encrypted data, or *file*, that the data owner stores on the cloud server is composed of three parts: a *file identifier FID*, a *KP-ABE ciphertext CT*, and an *encrypted message EM*. The encrypted message *EM* is a piece of data *M* encrypted with a random symmetric key K_S. The KP-ABE ciphertext *CT* is the symmetric key K_S encrypted through KP-ABE encryption and labeled with an encryption attribute set γ. The encryption attribute set of any KP-ABE ciphertext *CT* in the system includes the dummy attribute Att_D.

Every user u is provided with a KP-ABE decryption key $DK^{(u)}$ which embeds an access policy $\mathcal{T}^{(u)}$ defined over a set of attributes $\lambda^{(u)}$. The access policy $\mathcal{T}^{(u)}$ of every decryption key in the system includes the dummy attribute Att_D, which is ANDed at the root with the rest of the policy. Moreover, each user u is associated with a unique identifier $UID^{(u)} \in \{0,1\}^*$ decided by the data owner, which is basically a string. Each identifier is mapped to an NM-ABE attribute by using a collision-resistant hash function $\mathrm{H} : \{0,1\}^* \to \mathbb{Z}_p^*$. Hence, $\mathrm{H}(UID^{(u)})$ is the NM-ABE attribute for the identity $UID^{(u)}$. To simplify the notation we define the attribute $ID^{(u)} = \mathrm{H}(UID^{(u)})$. Every user u is also provided with an NM-ABE decryption key $NK^{(u)}$ that includes the attribute $\overline{ID^{(u)}}$ in the access policy attributes. This key is used only for revocation purposes. By using such key, the user u can decrypt all the NM-ABE ciphertexts *not labeled* with $I\!D^{(u)}$ in their encryption attributes. Each user u maintains a list for each attribute $i \in \lambda^{(u)} \setminus \{Att_D\}$ called $URKL_i$ (*user re-encryption key list for the attribute* i). Each element of this list is a user re-encryption key. Similarly, the cloud server maintains a list for each attribute $i \in \mathcal{U} \setminus \{Att_D\}$ called $CRKL_i$ (*cloud re-encryption key list for the attribute* i). Each element of this list is a cloud re-encryption key. The cloud server maintains also a list for each attribute $i \in \mathcal{U} \setminus \{Att_D\}$ called $EURKL_i$ (*encrypted user re-encryption key list for the attribute* i). Each element of this list is an NM-ABE ciphertext containing a user re-encryption key which is called $eurk_{i \leftrightarrow i'}$ (*encrypted user re-encryption key*). Moreover, the cloud server maintains a list for each attribute $i \in \mathcal{U} \setminus \{Att_D\}$ called PKL_i (*public parameter list for the attribute* i). Each element of this list is a component of the public parameters.

Finally, we define a new primitive called UpdateAtt which allows us to generate cloud and user re-encryption keys.

$(t'_i, T'_i, crk_{i \leftrightarrow i'}, urk_{i \leftrightarrow i'}) = \text{UpdateAtt}(i, MK_{KP})$ Choose α_i, β_i uniformly at random in $\mathbb{Z}_p$ and compute $t'_i = \alpha_i \cdot \beta_i$ and $T'_i = g^{t'_i}$. Then compute the *cloud re-encryption key* $crk_{i \leftrightarrow i'} = \alpha_i / t_i$ and set the *user re-encryption key* $urk_{i \leftrightarrow i'} = \beta_i$. Update the component of the master key t_i to t'_i, and the component of the public parameters T_i to T'_i. Output the cloud re-encryption key $crk_{i \leftrightarrow i'}$ and the user re-encryption key $urk_{i \leftrightarrow i'}$.

Procedures

In the following we describe the procedures of our scheme, namely *Setup*, *User Join*, *Data Production*, *Key Revocation*, and *Data Consumption*.

Setup. This procedure initializes the scheme. The data owner chooses the universe of the attributes $\mathcal{U}$ and executes the KP.Setup$(\mathcal{U})$ primitive which outputs the master key MK_{KP} and the public parameters PK_{KP}. Then, the data owner chooses a number d which specifies the number of attributes per NM-ABE ciphertext and executes the NM.Setup(d) primitive which outputs the master key MK_{NM} and the public parameters PK_{NM}. The data owner keeps the master keys MK_{KP} and MK_{NM} secret. Moreover, it authenticates the public parameters PK_{KP} and stores them on the cloud server. To authenticate these public parameters, the data owner might either sign every single T_i or implement a more efficient solution based on Merkle trees, as proposed in the previous chapter (Section 3.6).

User Join. This procedure provides a new joining user u with a decryption key $DK^{(u)}$ and an NM-ABE decryption key $NK^{(u)}$. First, the data owner defines the access policy $\mathcal{T}^{(u)}$ for the user and executes KP.KeyGen$(MK_{KP}, \mathcal{T}^{(u)})$ to generate the decryption key $DK^{(u)}$. Then, the data owner chooses an identifier $UID^{(u)}$ for the user u and an access policy $\tilde{\mathbb{A}}$ that includes the attribute $\overline{ID^{(u)}}$. Finally, it executes NM.KeyGen$(\tilde{\mathbb{A}}, MK_{NM}, PK_{NM})$ to generate the NM-ABE decryption key $NK^{(u)}$.

The data owner sends the keys $DK^{(u)}$ and $NK^{(u)}$ to the user through a secure channel. Then, it stores all the decryption key components dk_i on the cloud server, except for the one relative to the dummy attribute, i.e., dk_{Att_D}.

Data Production. This procedure is executed when the data owner produces a new piece of data M and wants to make it available to the users. First, the data owner chooses a unique file identifier FID and a random symmetric key $K_S \in \mathbb{G}_2$, and executes Enc(M, K_S) to generate the encrypted message EM. The notation Enc(M, K_S) refers to a symmetric key encryption algorithm, e.g., AES, executed on the message M with the key K_S. Then, the data owner chooses a set of encryption attributes γ and executes KP.Encrypt(K_S, γ, PK_{KP}) to generate the KP-ABE ciphertext CT. Finally, the data owner stores the file (FID, CT, EM) on the cloud server.

Key Revocation. This procedure is started by the data owner to revoke a decryption key $DK^{(r)}$ which has been compromised. The procedure is divided into two phases. The first phase is called *attribute update phase* and prevents $DK^{(r)}$ from decrypting KP-ABE ciphertexts produced after the key revocation. The second phase is called *components update phase* and includes both re-encryption of old KP-ABE ciphertexts and re-keying of legitimate users' decryption keys. While the former phase is immediately executed after the key compromise and promptly comes into force, the latter can be spread over time in a lazy fashion in occasion of data requests by the legitimate users.

> **Attribute Update Phase.** This phase is started by the data owner to revoke a decryption key. Let $DK^{(r)}$ be the decryption key to be revoked and $\lambda^{(r)}$ the access policy attributes of such a key. The data owner selects a minimum set of attributes $\mu^{(r)} \subseteq \lambda^{(r)} \setminus \{Att_D\}$ without which the access policy $\mathcal{T}^{(r)}$ will never be satisfied (*updatee set*). For each $i \in \mu^{(r)}$, the data owner executes UpdateAtt(i, MK_{KP}) to update the component of the master key t_i to t'_i and the component of the public parameters T_i to T'_i. The primitive outputs the cloud re-encryption key $crk_{i\leftrightarrow i'}$ and the user re-encryption key $urk_{i\leftrightarrow i'}$. Then, the data owner composes the message $\left(\{crk_{i\leftrightarrow i'}, T'_i\}_{i\in\mu^{(r)}}\right)$, signs it and sends it to the cloud server. Upon receiving the message from the data owner, the cloud server verifies the signature on it and stores each $crk_{i\leftrightarrow i'}$ as last element of the related $CRKL_i$. Moreover, the cloud server stores each T'_i as last element of the related PKL_i. Then, the data owner chooses $d-1$ attributes at random such that none of them is associated to a user identifier and adds the attribute $ID^{(r)}$ to

create the set of encryption attributes Γ. Next, for each $i \in \mu^{(r)}$, it encrypts the user re-encryption key through NM-ABE encryption. Specifically, it executes NM.Encrypt$(urk_{i \leftrightarrow i'}, \Gamma, PK_{NM})$ to generate an encrypted user re-encryption key for the attribute i, called $eurk_{i \leftrightarrow i'}$, which the user r is not able to decrypt, signs it and sends it to the cloud server. Upon receiving the message from the data owner, the cloud server verifies the signature on it and stores $eurk_{i \leftrightarrow i'}$ as last element of the related $EURKL_i$.

Components Update Phase. This phase is executed during each data consumption procedure. Let a file requested by a legitimate user u be (FID, CT, EM), and let his/her decryption key be $DK^{(u)}$. During this phase, the cloud server selects the encrypted user re-encryption keys $eurk_{i \leftrightarrow i'}$ to send to the user and updates the decryption key components of $DK^{(u)}$ and the ciphertext components of CT as follows.

For each $i \in \lambda^{(u)} \cap \gamma \setminus \{Att_D\}$, the cloud server obtains the current version of the attribute i from both dk_i and ct_i and selects the one with the oldest version number. Next, it locates the related entry in $EURKL_i$. If the selected entry is not the last one, it retrieves all the entries from the next one to the last one, i.e., $(eurk_{i \leftrightarrow i'}, \ldots, eurk_{i^{(l-1)} \leftrightarrow i^{(l)}})$. Then, the cloud server selects the cloud re-encryption key list $CRKL_i$, executes PRE.UpdateDK$(i, dk_i, CRKL_i)$ to obtain a *partially updated* decryption key component $\dot{dk}_i^{(l)}$, and PRE.UpdateCT$(i, ct_i, CRKL_i)$ to obtain a *partially updated* ciphertext component $\dot{ct}_i^{(l)}$. Moreover, for each $i \in \lambda^{(u)} \setminus \gamma$, the cloud server obtains the current version of the attribute i from dk_i. Next, it locates the related entry in $EURKL_i$. If the selected entry is not the last one, it retrieves all the entries from the next one to the last one, i.e., $(eurk_{i \leftrightarrow i'}, \ldots, eurk_{i^{(l-1)} \leftrightarrow i^{(l)}})$. Then, it selects the cloud re-encryption key list $CRKL_i$ and executes PRE.UpdateDK$(i, dk_i, CRKL_i)$ to obtain a *partially updated* decryption key component $\dot{dk}_i^{(l)}$.

Data Consumption. This procedure is started by a user u who wants to access a piece of data. First, the user requests a file to the cloud server by specifying the identifier FID. The cloud server retrieves the file (FID, CT, EM) and executes the components update phase. Next, it composes the message $\left(\dot{dk}_i^{(l)}, eurk_{i \leftrightarrow i^{(l)}}, \dot{ct}_j^{(l)}\right)$, where i refers to *outdated attributes* in the decryption key, and j refers to *outdated attributes* in the ciphertext *and* present in the decryption key (i.e., $j \in \sigma \subseteq (\lambda^{(u)} \cap \gamma)$). Then, the cloud server signs the message and sends it to the user.

Upon receiving the message from the cloud server, the user verifies the signature on it. Then, for each attribute i, the user executes NM.Decrypt$(eurk_{i \leftrightarrow i'}, NK^{(u)})$ to retrieve the user re-encryption key $urk_{i \leftrightarrow i'}$ and appends it to the user re-encryption key list $URKL_i$. When the user re-encryption key list $URKL_i$ is updated to the last version, the user selects it together with the received partially updated decryption key component $\dot{dk}_i^{(l)}$, and executes PRE.UpdateDK$(i, \dot{dk}_i^{(l)}, URKL_i)$, thus obtaining the *updated* decryption key component $dk_i^{(l)}$. Moreover, for each attribute j, the user selects the user re-encryption key list $URKL_j$ and the received partially updated ciphertext component $\dot{ct}_j^{(l)}$, and executes PRE.UpdateCT$(j, \dot{ct}_j^{(l)}, URKL_j)$, thus obtaining the *updated* ciphertext component $ct_j^{(l)}$. Then, the user composes the message $\left(dk_i^{(l)}, ct_j^{(l)}\right)$, signs it and sends it to the cloud server.

Upon receiving the message from the user, the cloud server verifies the signature on it. Next, for each attribute j, the cloud server verifies that the user updated the ciphertext component ct_j correctly. In order to do that, the cloud server checks that $e\left(ct_j, T_j^{(l)}\right) = e\left(ct_j^{(l)}, T_j\right)$. If all the comparisons are correct, the cloud server updates the ciphertext CT with the ciphertext components $ct_j^{(l)}$, and the decryption key components for the user u with the received ones. Then, the cloud server composes the message $\left(\hat{ct}, \{ct_k\}_{k \in (\lambda^{(u)} \cap \gamma) \setminus \sigma}, EM\right)$, signs it and sends it to the user. Note that the cloud server sends only the ciphertext components that were already up to date *and* present in the user's decryption key. Depending on the application, this optimization can save a lot of communication overhead.

Upon receiving the message from the cloud server, the user verifies the signature on it. Next, it composes the ciphertext $CT = (\hat{ct}, \{ct_j^{(l)}\}_{j \in \{\lambda^{(u)} \cap \gamma\}})$. The user can now decrypt the ciphertext CT by executing KP.Decrypt$(CT, DK^{(u)})$, thus retrieving the symmetric key K_S. Finally, it executes Dec(EM, K_S) and retrieves the data M.

4.3 Security Analysis

In this section, we provide security proofs for our scheme. Our aim is to show that our scheme is not less secure than the GPSW scheme (Goyal et al., 2006), which has been proved secure in the Selective-Set model under the Decisional Bilinear Diffie-Hellman (DBDH) assumption. We first define a game (Game 1) which models collusion among users whose keys cannot decrypt a given ciphertext and users whose keys have been revoked. We formally prove that our scheme is resistant against such an adversary under the DBDH assumption. Then, we define a second game (Game 2) which models a cloud server that comes into possession of revoked keys. We formally prove that our scheme is resistant against such an adversary under the DBDH assumption.

Game 1

Init The adversary declares the universe of attributes $\mathcal{U}$, the challenge set γ, the number of key revocations n, and the access policies of the decryption keys to revoke, i.e., $\mathcal{T}_j, \forall j \in [1, n]$.

Setup The challenger runs the KP.Setup primitive, thus creating the public parameters $PK_{KP}^{(0)}$. Next, for each key revocation j, the challenger determines the updatee set μ_j from $\mathcal{T}_j$ and runs the UpdateAtt primitive for each attribute $i \in \mu_j$, thus creating the user re-encryption keys $\{urk_{i^{(j-1)} \leftrightarrow i^{(j)}}\}$. Then, it creates the public parameters $PK_{KP}^{(j)}$. Finally, the challenger gives the $n+1$ sets of the public parameters, i.e., $\{PK_{KP}^{(0)}, \ldots, PK_{KP}^{(n)}\}$, and the user re-encryption keys to the adversary.

Phase 1 The adversary is allowed to issue queries for: (i) decryption keys of any version whose access policy $\mathcal{T}$ is not satisfied with the challenge set; and (ii) decryption keys with access policy equal to any $\mathcal{T}_j$, but version less than j. The *version* $v \in [0, n]$ specifies which master key the challenger uses to create the decryption key requested.

Challenge The adversary submits two distinct messages m_0 and m_1 of equal length. The challenger sets $b \leftarrow \{0, 1\}$ uniformly at random, runs the KP.Encrypt primitive on m_b with the challenge set, and sends the ciphertext $CT^\star$ to the adversary.

Phase 2 Phase 1 is repeated.

Guess The adversary outputs a guess $b^\star$ of b.

The advantage of an adversary $\mathcal{A}$ in this game is defined as $\Pr[b^\star = b] - \frac{1}{2}$.

Proof of Security

We prove that no Probabilistic Polynomial-Time (PPT) adversary can play Game 1 against our scheme with a non-negligible advantage under the DBDH assumption. To do this, we prove that breaking our scheme reduces to breaking the GPSW scheme, which in turn is proved to be infeasible for a PPT adversary (see Goyal et al. (2006)). More formally, we state the following:

Theorem 1. *If a PPT adversary can play Game 1 against our scheme with a non-negligible advantage, then a simulator can be built to play the Selective-Set game (Game 0) against the GPSW scheme with the same advantage.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ able to win Game 1 against our scheme with advantage ϵ. We build a simulator $\mathcal{B}$ that can play Game 0 against the GPSW scheme with the same advantage ϵ.

Init The simulator $\mathcal{B}$ starts Game 1 against the adversary $\mathcal{A}$. $\mathcal{A}$ chooses the universe of attributes $\mathcal{U}$, the challenge set γ, the number of key revocations n, and the access policies of the decryption keys to revoke, i.e., $\mathcal{T}_j, \forall j \in [1, n]$.

Setup In this phase, the simulator $\mathcal{B}$ builds an alternative universe of attributes $\mathcal{U}'$ and an alternative challenge set γ', which will be used in Game 0 against the GPSW scheme. The alternative universe will be composed of an attribute i_0 for each attribute i in the original universe, plus an attribute i_j for each attribute i in the updatee set of each key revocation j. In formulas:

$$\mathcal{U}' = \{i\_0 \mid i \in \mathcal{U}\} \cup \{i\_j \mid i \in \mu_j, j \in [1, n]\}.$$

For convenience, we define the following function:

> $i\_x \leftarrow \text{FindCurrVer}(i, \mathcal{U}', v)$ This function takes as input an attribute named $i \in \mathcal{U}$, the universe of attributes $\mathcal{U}'$, and a version number v. The function outputs the attribute i_x, where $x = \max_{x \leq v}\{x \mid i\_x \in \mathcal{U}'\}$.

The alternative challenge set will be composed of the outputs of $\text{FindCurrVer}(i, \mathcal{U}', n)$ for each attribute i in the original challenge set. In formulas:

$$\gamma' = \{\text{FindCurrVer}(i, \mathcal{U}', n) \mid i \in \gamma\}.$$

The simulator $\mathcal{B}$, which acts as adversary in Game 0, selects the universe of attributes $\mathcal{U}'$ and the challenge set γ', and runs Game 0 Init phase. The challenger replies with the public parameters $PK'_{KP} = \{T_{i\_x} \mid i\_x \in \mathcal{U}'\}$. The simulator creates $n + 1$ sets of public parameters for $\mathcal{A}$. Specifically, it defines each set as

$$PK^{(v)}_{KP} = \{T_{i\_x} \in PK'_{KP} \mid i\_x = \text{FindCurrVer}(i, \mathcal{U}', v), \forall i \in \mathcal{U}\},$$

and calls each T_{i_x} as $T^{(v)}_i$. Then, the simulator chooses $\sum_j \mid \mu_j \mid$ random numbers in $\mathbb{Z}_p$ to create all the user re-encryption keys and gives them and the sets of public parameters $\{PK^{(0)}_{KP}, \ldots, PK^{(n)}_{KP}\}$ to the adversary $\mathcal{A}$.

Phase 1 For each decryption key with policy $\mathcal{T}$ and version v that the adversary $\mathcal{A}$ requests, the simulator $\mathcal{B}$ computes the corresponding policy $\mathcal{T}'$ in the alternative universe by replacing each attribute $i \in \mathcal{T}$ with the attribute $\text{FindCurrVer}(i, \mathcal{U}', v)$. Then, the simulator makes a decryption key request to the GPSW scheme and forwards the obtained decryption key to the adversary $\mathcal{A}$.

Challenge The simulator $\mathcal{B}$ receives the messages m_0 and m_1 from the adversary $\mathcal{A}$, selects them as its own challenge messages with respect to the GPSW scheme, and runs Game 0 Challenge phase. The simulator propagates the obtained ciphertext $CT^\star$ to the adversary $\mathcal{A}$.

Phase 2 The simulator acts as it did in Phase 1.

Guess The simulator $\mathcal{B}$ receives the guess $b^\star$ from the adversary $\mathcal{A}$ and selects it as its own guess with respect to the GPSW scheme.

As shown, the simulator can map any attribute of any version of our scheme onto an attribute of the GPSW scheme, and therefore it can simulate the versioning of the attributes (with the adversary $\mathcal{A}$). The simulator translates the attributes from the universe $\mathcal{U}$ to $\mathcal{U}'$ and maps each decryption key request from $\mathcal{A}$ to an equivalent request compliant with the GPSW scheme. Hence, the advantage of the simulator $\mathcal{B}$ in Game 0, i.e., in the Selective-Set game, against the GSPW scheme is the same as the one of the adversary $\mathcal{A}$ in Game 1 against our scheme. ■

Game 2

Init The adversary declares the universe of attributes $\mathcal{U}$, the challenge set γ, the number of key revocations n, and the access policies of the decryption keys to revoke, i.e., $\mathcal{T}_j, \forall j \in [1, n]$. Note that the universe of attribute, the challenge set, and the access policies include the dummy attribute Att_D.

Setup The challenger runs the KP.Setup primitive, thus creating the public parameters $PK_{KP}^{(0)}$. Next, for each key revocation j, the challenger determines the updatee set μ_j from $\mathcal{T}_j$ and runs the UpdateAtt primitive for each attribute $i \in \mu_j$, thus creating the cloud re-encryption keys $\{crk_{i^{(j-1)} \leftrightarrow i^{(j)}}\}$. Then, it creates the public parameters $PK_{KP}^{(j)}$. Finally, the challenger gives the $n+1$ sets of the public parameters, i.e., $\{PK_{KP}^{(0)}, \ldots, PK_{KP}^{(n)}\}$, and the cloud re-encryption keys to the adversary.

Phase 1 The adversary is allowed to issue queries for: (i) *incomplete decryption keys*, i.e., decryption keys which do not include the decryption key component relative to the dummy attribute (dk_{Att_D}); and (ii) decryption keys with access policy equal to any $\mathcal{T}_j$, but version less than j.

Challenge The adversary submits two distinct messages m_0 and m_1 of equal length. The challenger sets $b \leftarrow \{0, 1\}$ uniformly at random, runs the KP.Encrypt primitive on m_b with the challenge set, and sends the ciphertext $CT^\star$ to the adversary.

Phase 2 Phase 1 is repeated.

Guess The adversary outputs a guess $b^\star$ of b.

The advantage of an adversary $\mathcal{A}$ in this game is defined as $\Pr[b^\star = b] - \frac{1}{2}$.

Proof of Security

We prove that no PPT adversary can play Game 2 against our scheme with a non-negligible advantage under the DBDH assumption. To do this, we prove that breaking our scheme with Game 2 rules reduces to breaking our scheme with Game 1 rules, which we just proved to be infeasible for a PPT adversary (see Theorem 1). More formally, we state the following:

Theorem 2. *If a PPT adversary can play Game 2 against our scheme with a non-negligible advantage, then a simulator can be built to play Game 1 against our scheme with the same advantage.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ able to win Game 2 against our scheme with advantage ϵ. We build a simulator $\mathcal{B}$ that can play Game 1 against our scheme with the same advantage ϵ.

Init The simulator $\mathcal{B}$ starts Game 2 against the adversary $\mathcal{A}$. $\mathcal{A}$ chooses the universe of attributes $\mathcal{U}$, the dummy attribute Att_D, the challenge set γ, the number of key revocations n, and the access policies of the decryption keys to revoke, i.e., $\mathcal{T}_j, \forall j \in [1, n]$.

Setup In this phase, the simulator $\mathcal{B}$ adds another dummy attribute Att'_D to the universe of attributes, selects γ as its challenge set, selects $\mathcal{T}_j, \forall j \in [1, n]$ as the access policies of the decryption keys to revoke, and runs Game 1 Init phase. The challenger replies with the sets of public parameters $\{PK_{KP}^{(0)}, \ldots, PK_{KP}^{(n)}\}$ and the user re-encryption keys. The simulator removes the component of the public parameters relative to the dummy attribute Att'_D in every set of public parameters. Then, it selects the received user re-encryption keys as its cloud re-encryption keys. Finally, the simulator gives the sets of public parameters and the cloud re-encryption keys to the adversary.

Phase 1 For each revoked decryption key with policy $\mathcal{T}_j$ and version $v < j$ that the adversary $\mathcal{A}$ requests, the simulator $\mathcal{B}$ makes a decryption key request to the Game 1 challenger and forwards the obtained decryption key to the adversary.

For each incomplete decryption key with policy $\mathcal{T}$ and version v that the adversary $\mathcal{A}$ requests, the simulator substitutes the dummy attribute Att_D with the new dummy attribute Att'_D and makes a decryption key request to the Game 1 challenger. Then, the simulator removes the component $dk_{Att'_D}$ from the obtained decryption key and sends such an incomplete decryption key to the adversary.

Challenge The simulator $\mathcal{B}$ receives the messages m_0 and m_1 from the adversary $\mathcal{A}$, selects them as its own challenge messages and runs Game 1 Challenge phase. The simulator propagates the obtained ciphertext $CT^\star$ to the adversary $\mathcal{A}$.

Phase 2 Phase 1 is repeated.

Guess The simulator $\mathcal{B}$ receives the guess $b^\star$ from the adversary $\mathcal{A}$, selects it as its own guess, and runs Game 1 Guess phase.

As shown, the simulator can accommodate every decryption key request from the adversary. Notably, the simulator is able to provide the adversary with *any* incomplete decryption key. Indeed, during Phase 1, it may happen that the adversary requests incomplete decryption keys relative to decryption keys whose access policy $\mathcal{T}$ is satisfied with the challenge set. In such a case, the simulator cannot propagate the request to the Game 1 challenger. However, by substituting Att_D with Att'_D in $\mathcal{T}$, the simulator ensures that the access policy is not satisfied with the challenge set and that the request will be compliant with those accepted by the Game 1 challenger. Therefore, the advantage of the simulator $\mathcal{B}$ in Game 1 against our scheme is the same as the one of the adversary $\mathcal{A}$ in Game 2 against our scheme. ■

4.4 Performance Evaluation

In this section we analyze the computational cost of the various operations in our system. Such costs are mainly determined by the ABE operations as they are the most complex and onerous. Table 4.1 shows the complexity of ABE operations executed by each entity during the various procedures, and it shows a comparison between our scheme and the YWRL one. Note that in order to implement the key revocation mechanism, our scheme requires access policies with only one negated attribute embedded in the NM-ABE decryption keys, and only one attribute labeling the NM-ABE ciphertexts. For this reason we set $d = |\Lambda| = 1$.

The data owner initializes the scheme through the setup procedure. Differently from YWRL scheme which initializes only the KP-ABE scheme, our scheme initializes also the NM-ABE scheme. This cost is only 1 pairing operation and 3 operations (point-scalar multiplications) in $\mathbb{G}_1$ to create the public parameters. Also in

Data owner

Procedure	**Scheme**	**Bilinear Pairings**	**Operations in $\mathbb{G}_1$**	**Operations in $\mathbb{G}_T$**
Setup	Our	2	$\lvert\mathcal{U}\rvert + 3$	–
	YWRL	1	$\lvert\mathcal{U}\rvert$	–
User Join	Our	–	$\lvert\lambda\rvert + 3$	–
	YWRL	–	$\lvert\lambda\rvert$	–
Data Production	Our	–	$\lvert\gamma\rvert$	1
	YWRL	–	$\lvert\gamma\rvert$	1
Key Revocation	Our	–	$4\lvert\mu\rvert$	$\lvert\mu\rvert$
	YWRL	–	$\lvert\mu\rvert$	–

Cloud server

Procedure	**Scheme**	**Bilinear Pairings**	**Operations in $\mathbb{G}_1$**	**Operations in $\mathbb{G}_T$**
Data Consumption (in case of components update)	Our	$2(\lvert\gamma\rvert - 1)$	$\lvert\lambda\rvert + \lvert\gamma\rvert - 2$	–
	YWRL	–	$\lvert\lambda\rvert + \lvert\gamma\rvert - 2$	–

User

Procedure	**Scheme**	**Bilinear Pairings**	**Operations in $\mathbb{G}_1$**	**Operations in $\mathbb{G}_T$**
Data Consumption	Our	$\lvert\lambda\rvert$	–	–
	YWRL	$\lvert\lambda\rvert$	–	–
Data Consumption (in case of components update)	Our	$\lvert\lambda\rvert + 3(\lvert\lambda\rvert - 1)R$	$\lvert\lambda\rvert + \lvert\gamma\rvert - 2$	–
	YWRL	$\lvert\lambda\rvert$	–	–

Table 4.1: Computational costs comparison.

the user join procedure, our scheme adds a constant and moderate cost due to the employment of the NM-ABE scheme. In particular, the data owner executes the NM.KeyGen primitive which requires 3 operations in $\mathbb{G}_1$. The data production procedure is the procedure that is executed most frequently by the data owner. The cost of this procedure for both our and YWRL scheme is determined only by the execution of the KP.Encrypt, whose complexity depends on the number of encryption attributes. Specifically, it requires $|\gamma|$ operations in $\mathbb{G}_1$ and 1 operation (modular exponentiation) in $\mathbb{G}_T$ to blind the message, i.e., MY^s. The key revocation procedure is composed of two phases. The data owner is involved only in the first one, i.e., the attribute update phase. Both in our scheme and YWRL one, during this phase, the data owner first determines the updatee set μ, and then it generates a new version of the quantities related to the attributes $i \in \mu$. The main cost for these operations is determined by the computation of a new version of the public parameters T_i to T'_i, $\forall i \in \mu$. Each of these updates requires one operation in $\mathbb{G}_1$. In addition to this cost, our scheme also requires the execution of the NM.Encrypt primitive for each attribute in the updatee set in order to secure the user re-encryption keys. The cost of each NM.Encrypt is 3 operations in $\mathbb{G}_1$ and 1 operation in $\mathbb{G}_T$.

The cloud server performs ABE operations only during the data consumption procedure, both in our and in YWRL scheme. If both the user's decryption key and the requested ciphertext are up to date, the data consumption procedure has no ABE-related costs for the cloud server. On the other hand, when ciphertext and/or decryption key components need to be updated (components update phase), the cloud server executes the PRE.UpdateCT and PRE.UpdateDK primitives. In the worst case, these primitives are executed for each attribute labeling the ciphertext and for each attribute embedded in the user's decryption key, except for the dummy attribute. Hence, the total cost is $|\lambda| - 1 + |\gamma| - 1$ operations in $\mathbb{G}_1$, both in our and in YWRL scheme. Note that, even in the case an attribute has to be updated of more than one version, the cost is still one operation in $\mathbb{G}_1$ per component since these primitives first use the re-encryption key list to compute an equivalent re-encryption key able to update the attribute from *any* old version to the latest version. Then, such re-encryption key is used as argument of the operation in $\mathbb{G}_1$. In addition to this cost, our scheme also includes $2(|\gamma| - 1)$ pairings. This cost is related to the verifiability property, which lets the cloud server check that the user updated all the ciphertext components correctly.

The user performs ABE operations only during the data consumption procedure, both in our and in YWRL scheme. If both the user's decryption key and the requested ciphertext are up to date, the data consumption procedure's cost includes $|\lambda|$ pairings operations in the worst case, both in our scheme and in YWRL one. On the other hand, when ciphertext and/or decryption key components need to be updated (components update phase), the cost of our scheme is higher than YWRL scheme since the user participates in re-encryption and re-keying. However, this adds security to our scheme and allows us to prevent the cloud server from undoing key revocations. In our scheme, the user executes the NM.Decrypt primitive to retrieve the user re-encryption keys. This primitive is executed for each attribute in λ (except for the dummy attribute) and for the number of key revocations R occurred since his/her last execution of the data consumption procedure. The cost of a single NM.Decrypt primitive is 3 pairings, so the total cost in the worst case is $3(|\lambda| - 1)R$ pairings. Also, the user executes the PRE.UpdateCT and PRE.UpdateDK primitives. In the worst case, these primitives are executed for each attribute labeling the ciphertext and for each attribute embedded in the user's decryption key, except for the dummy attribute. Hence, the total cost is $|\lambda| - 1 + |\gamma| - 1$ operations in $\mathbb{G}_1$.

The setup, user join, data production, and data consumption procedures have a small additional cost compared to YWRL scheme. On the contrary, the key revocation procedure and data consumption in case of components update have a non-negligible additional cost, which depends on the average complexity of the access policies. Indeed, the more the access policies are complex, the more the cardinality of the sets λ, μ, and γ will grow. However, such an additional cost is present only when key revocations occur and lasts for few data consumption procedures following the key revocations. Considering that a key revocation is a rare event compared to data consumption and data production, the additional cost introduced by our scheme should be acceptable in order to provide additional security that limits the cloud server capabilities and inhibits it from accessing the data stored on it when in possession of a revoked key.

Chapter 5

fABElous: An Attribute-Based Scheme for Industrial Internet of Things

Do not say a little in many words
but a great deal in a few.

Pythagoras

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 has a deep impact on our everyday life, as common objects can be empowered with communication and cooperation capabilities. However, also the industrial sector 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). Furthermore, in a smart warehouse, connected sensors can help automated guided vehicles to find particular goods which have to be loaded on a given truck. In all these systems, security is a key requirement, especially for integrity, confidentiality, and access control on data. Again, in these scenarios, ABE is a valid cryptographic scheme to obtain the aforementioned requirements. Nevertheless, though ABE techniques can offer a high level of security and an intrinsic fine-grained access control, they do not fit easily in the IoT world, especially for real-time applications. Here, despite one may think, the computation power required to execute ABE operations might not be the major concern, as ABE was shown to be suitable for IoT devices (Ambrosin et al., 2016; Girgenti et al., 2019). Instead, the most challenging aspect here 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 Low Energy (BLE), 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)).

In this chapter, we propose fABElous, an ABE scheme suitable for industrial IoT applications which aims at minimizing the communication overhead introduced by ABE encryption. The fABElous scheme ensures data integrity, confidentiality, and access control, and reduces the communication overhead up to 52.5 % compared to a scheme which uses ABE techniques naively.

This chapter is organized as follows. Section 5.1 compares with relevant related work. Section 5.2 introduces some background on Ciphertext-Policy Attribute-Based Encryption. Section 5.3 describes fABElous in detail: its architecture, a reference use case, system procedures, and the reference threat model. Section 5.4 analyzes the performances of fABElous in terms of communication overhead.

5.1 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 section we will call *users* humans that take part in the services/systems described. We will also call *nodes* the independent devices (mainly sensors and actuators) that take part in the services/systems described.

Yu et al. (2011) realized the first distributed fine-grained data access control for Wireless Sensor Networks (WSNs) using ABE. In their work, the system is composed of one controller, many users and many nodes. The network controller assigns a secret key to each user, according to a policy that describes the type of data such user should decrypt. Every node possesses a set of attributes, which are generated from the controller and loaded on the node before its installation. Their system is able to revoke a key with a single broadcast message. However, their system does not take into account the possibility of actuators which receive and use ABE-encrypted data. This precludes the possibility of having actuators which receive and use ABE-encrypted data in complex IoT scenarios like a smart factory.

Picazo-Sanchez et al. (2014) proposed a secure publish-subscribe protocol for medical Wireless Body Area Networks (WBANs) using ABE. In their work, the system is composed of a star-topology network where a smartphone (or a similar device) communicates with various nodes placed over the user's body area, monitoring the user's health conditions. The system allows any node to publish data and to subscribe to data generated from other nodes.

Singh et al. (2015) proposed a secure MQTT for IoT, allowing the usage of ABE. In their work, the architecture is composed of one Key Authority, one broker, and several nodes. Every node can be a subscriber, a publisher or both. Every node knows the public key, and a secret key associated with some attributes which describe its characteristics. Every node subscribes to the data it needs for its proper functioning. Both Picazo-Sanchez et al.'s and Singh et al.'s schemes follow the CP-ABE paradigm like ours, but they use a publish-subscribe method, which is unsuitable for our objectives since it introduces too much latency.

5.2 Preliminaries

Ciphertext-Policy Attribute-Based Encryption

The CP-ABE paradigm was first introduced by Bethencourt, Sahai, and Waters (Bethencourt et al., 2007) as the *dual* scheme of KP-ABE. In the following, we will refer to this scheme as the BSW scheme. In CP-ABE an encrypting party labels each piece of encrypted data (*CP-ABE ciphertext*) with an *access policy* ($\mathcal{T}$) defined over some attributes (*access policy attributes*, λ). On the contrary, decryption keys embed a set of attributes (*decryption attributes*, γ).

While the KP-ABE paradigm provides a content-based access control as the ciphertext is associated with a set of attributes which describes its *content*, the CP-ABE paradigm provides a role-based access control as the decryption key is associated with a set of attributes which describes the characteristics–or better yet, the *role*–of the key owner.

We recall that an encrypting party is called *data producer*, whereas who holds a decryption key and decrypts data is called a *data consumer*. The entity which deploys and manages the system is called *trusted party*.

In this work we build on BSW scheme. In order to ease the reading, we abstract away the mathematical insight of such a scheme. The interested reader can refer to Bethencourt et al. (2007) for such details. We model BSW scheme with the following black-box primitives.

$(MK, PK) = \text{CP.Setup}()$ This primitive initializes the scheme. It outputs a *master key MK*, which must be kept secret, and an associated set of *public parameters PK*. The CP.Setup primitive is executed by the trusted party.

$CT = \text{CP.Encrypt}(M, \mathcal{T}, PK)$ This primitive encrypts a message M under the access policy $\mathcal{T}$. It takes as input the message M, the policy $\mathcal{T}$, and the public parameters

PK. It outputs the CP-ABE ciphertext CT. The CP.Encrypt primitive is executed by a data producer.

$DK = \text{CP.KeyGen}(MK, \gamma)$ This primitive generates a decryption key DK, which is provided to a data consumer. It takes as input the master key MK and a set of attributes γ which describes the data consumer. It outputs a decryption key DK. The CP.KeyGen primitive is executed by the trusted party.

$M = \text{CP.Decrypt}(CT, DK)$ This primitive retrieves the message M from the CP-ABE ciphertext CT if the access policy $\mathcal{T}$ in the CP-ABE ciphertext is satisfied by means of the decryption attributes γ embedded in the decryption key. It produces $\perp$ otherwise. The CP.Decrypt primitive is executed by a data consumer.

5.3 Architecture

We assume a low-bitrate *Wireless Sensor and Actuator Network* (WSAN) in which the nodes exchange encrypted data with each other. As use-case example, consider a smart factory that implements a real-time application (Chen et al., 2009). Because of the strict requirements on the communication delay, we consider a direct communication between the nodes. Moreover, we assume that the WSAN uses a low-energy and low-bitrate link-layer protocol, e.g., IEEE 802.15.4. As a consequence, communications and encryption/decryption operations should be as lightweight as possible.

We consider the WSAN to be composed of sensors, actuators, and a controller (Figure 5.1). A *sensor* is a data producer that measures some quantity, encrypts it, and sends it to a set of actuators. An *actuator* is a data consumer that receives encrypted data from a set of sensors and uses it to control some mechanism. The received data can be, for example, a command to be executed, or a sensor measurement the actuator uses to take a decision. For the sake of simplicity, we keep sensor and actuator roles separated; however, a single device may act as both. Sensors and actuators are regulated by a special node called *controller*, or C, and they are characterized by a unique identifier called *Sensor ID* (S_{ID}) and *Actuator ID* (A_{ID}), respectively. These identifiers are chosen and assigned by the controller. Every party p in the system holds a public/private key pair $(K_{pub}^{(p)}, K_{priv}^{(p)})$ through which it can perform digital signature algorithms. The notation $\text{Sign}(p, (m))$ represents the signature algorithm of the party p on the message m. Controller and actuators maintain a local *sensor table*; every tuple of the table identifies a sensor through its sensor ID and the related public key. Moreover, the controller maintains an *actuator table*; every tuple of the table identifies an actuator through its actuator ID and the related public key.

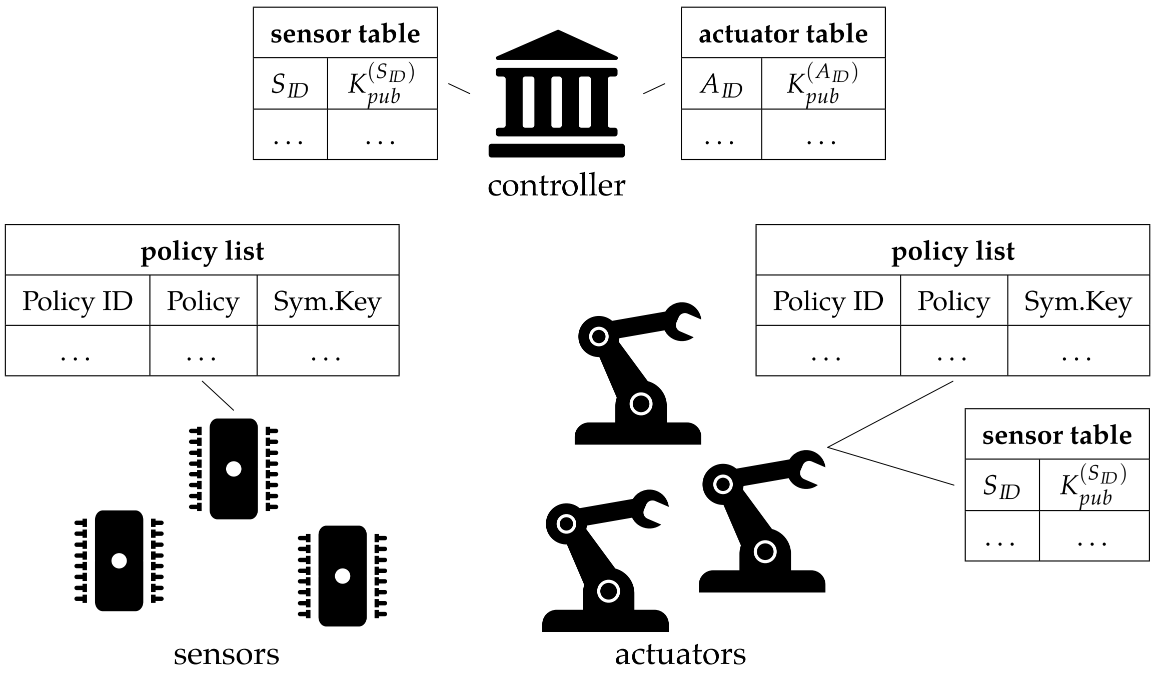


Figure 5.1: An overview of fABElous architecture.

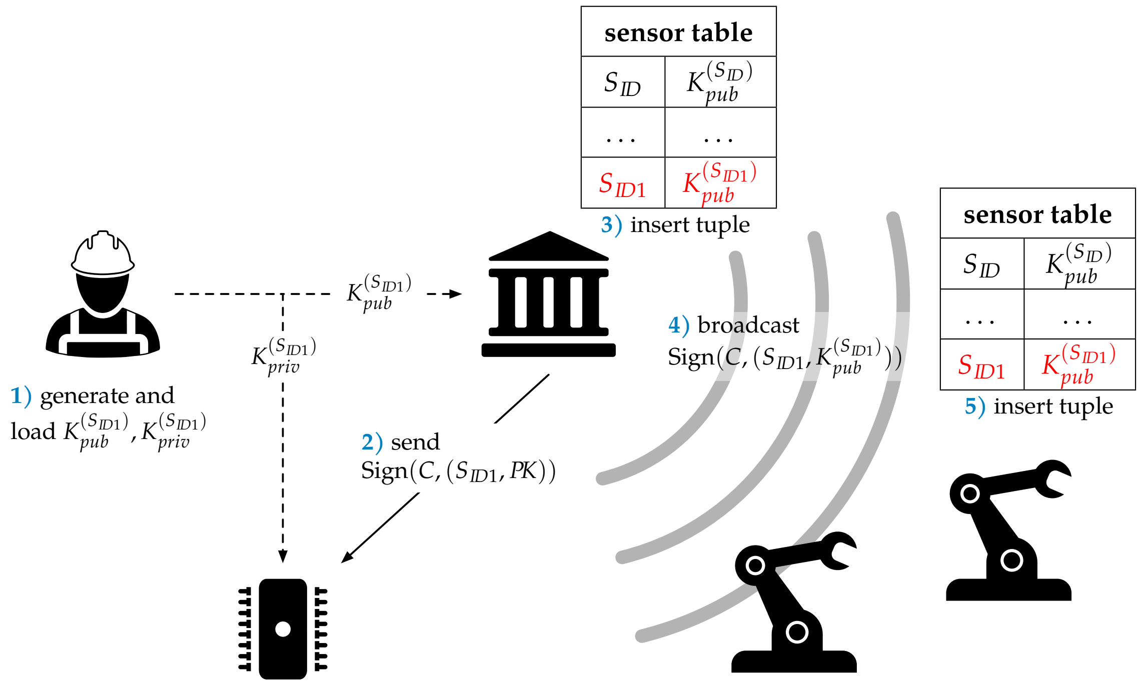
Key Distribution Mechanism

In order to satisfy the strict requirements of our model regarding security and messages size, fABElous diminishes the use of CP-ABE heavier ciphertext and primitives. Indeed, in our scheme each sensor executes the CP.Encrypt primitive only once for securing multiple data described by the same access policy. Similarly, each actuator executes the CP.Decrypt primitive only once for extracting data generated by the same sensor and described by the same access policy. The basic idea is to distribute symmetric keys using the BSW scheme as a reliable tool for achieving fine-grained multicast. To this aim, a sensor encrypts a symmetric key with the CP.Encrypt primitive under a certain policy and broadcasts it. All the actuators will receive the ciphertext, but only some of them will be able to retrieve and store the symmetric key by successfully executing the CP.Decrypt primitive. Henceforth, when the sensor needs to transmit a new piece of data which has to be encrypted under the same access policy, it simply encrypts the data with the symmetric key. In the following, we describe the procedures of our system in detail.

System Procedures

Setup. This procedure initializes the system. The controller executes the CP.Setup primitive, thus producing the master key, which is kept secret, and the public parameters.

Sensor Join. The sensor join procedure is executed whenever a new sensor joins the WSAN. Figure 5.2 shows the various steps of this procedure.



sensor table
S_{ID} $K_{pub}^{(S_{ID})}$
… …
S_{ID1} $K_{pub}^{(S_{ID1})}$
3) insert tuple
sensor table
S_{ID} $K_{pub}^{(S_{ID})}$
… …
S_{ID1} $K_{pub}^{(S_{ID1})}$
5) insert tuple
$K_{pub}^{(S_{ID1})}$
$K_{priv}^{(S_{ID1})}$
1) generate and load $K_{pub}^{(S_{ID1})}, K_{priv}^{(S_{ID1})}$
2) send Sign$(C, (S_{ID1}, PK))$
4) broadcast Sign$(C, (S_{ID1}, K_{pub}^{(S_{ID1})}))$

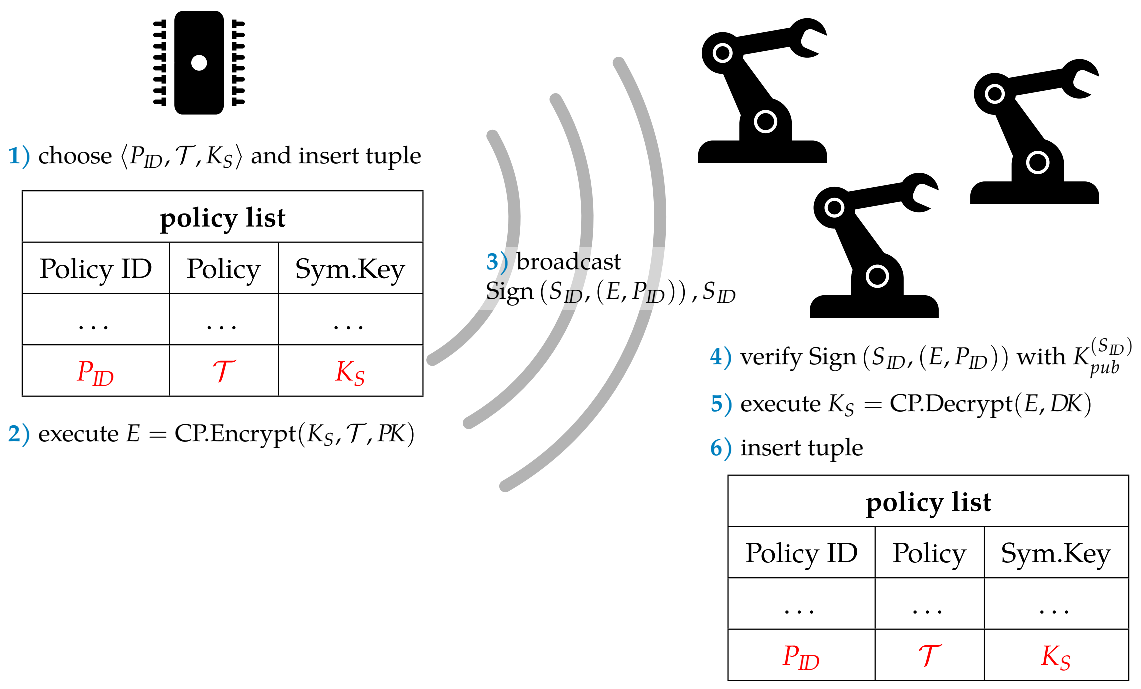
Figure 5.2: Sensor join procedure. Dashed lines represent human-device communication.

In step 1, a human operator physically deploys the sensor and generates a public/private key pair $(K_{pub}^{(S_{ID})}, K_{priv}^{(S_{ID})})$. The operator loads the private key on the sensor and the public key on the controller. In step 2, the controller chooses a unique identifier S_{ID} for the sensor and sends the message Sign$(C, (S_{ID}, PK))$ to the sensor. In step 3, the controller adds the tuple $\langle S_{ID}, K_{pub}^{(S_{ID})} \rangle$ to its sensor table. In step 4, the controller signs and broadcasts the message Sign$(C, (S_{ID}, K_{pub}^{(S_{ID})}))$. Upon receiving the message, each actuator verifies the signature to be valid and adds the tuple to its sensor table (step 5).

Actuator Join. This procedure is executed whenever a new actuator joins the WSAN. In step 1, a human operator physically deploys the actuator and generates a public/private key pair $(K_{pub}^{(A_{ID})}, K_{priv}^{(A_{ID})})$. The operator loads the private key on the actuator and the public key on the controller. The controller chooses a unique identifier A_{ID} and an attribute set γ for the actuator; next, it executes the CP.KeyGen primitive, thus obtaining the decryption key DK. Then, the controller signs the message (A_{ID}, DK), encrypts it with the actuator's public key, and sends it to the actuator.

The controller adds the tuple $\langle A_{ID}, K_{pub}^{(A_{ID})}\rangle$ to its actuator table and, finally, sends a copy of its sensor table to the actuator through a signed message.

New Policy Installation. This procedure is executed by a sensor to share a symmetric key with some actuators. When a sensor performs this procedure for the first



1) choose $\langle P_{ID}, \mathcal{T}, K_S\rangle$ and insert tuple
policy list
Policy ID
Policy
Sym.Key
...
P_{ID}
$\mathcal{T}$
K_S
2) execute $E = \text{CP.Encrypt}(K_S, \mathcal{T}, PK)$
3) broadcast Sign $(S_{ID}, (E, P_{ID}))$, S_{ID}
4) verify Sign $(S_{ID}, (E, P_{ID}))$ with $K_{pub}^{(S_{ID})}$
5) execute $K_S = \text{CP.Decrypt}(E, DK)$
6) insert tuple
policy list
Policy ID
Policy
Sym.Key
...
P_{ID}
$\mathcal{T}$
K_S

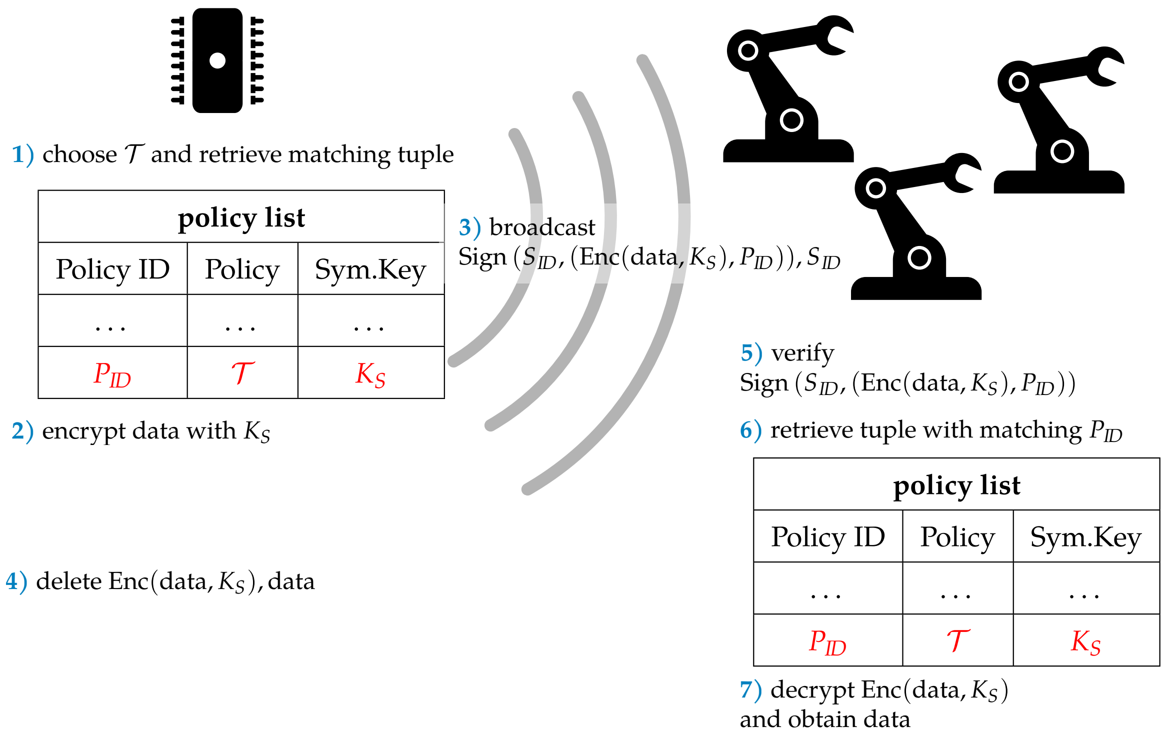
Figure 5.3: New policy installation procedure.

time, it creates a *policy list*. This list links an access policy $\mathcal{T}$ to a symmetric key K_S through a *policy identifier* (P_{ID}). Each tuple of the list is in the form $\langle P_{ID}, \mathcal{T}, K_S\rangle$. Each sensor creates and maintains its policy list, therefore different sensors have different policy lists. Similarly, each actuator maintains its own policy list which contains tuples that sensors shared by means of this procedure.

The following steps, shown also in Figure 5.3, allow a sensor to create a tuple for its policy list and share such tuple with some actuators over the WSAN. In step 1, the sensor chooses an access policy $\mathcal{T}$, a random symmetric key K_S, and a new policy identifier P_{ID}; next, the sensor composes the tuple and adds it to its policy list. In step 2, the sensor encrypts the symmetric key K_S under the access policy $\mathcal{T}$ by executing the CP.Encrypt primitive, thus obtaining the CP-ABE ciphertext CT. In step 3, the sensor creates a *new policy message*, which is a signed message that includes the ciphertext CT, the policy identifier P_{ID}, and the sensor ID S_{ID}, and broadcasts the new policy message. Upon receiving the message, each actuator verifies the signature to be valid and checks whether its attribute set γ satisfies the access policy

$\mathcal{T}$ (step 4). If not, the message is discarded. Otherwise, the actuator decrypts the ciphertext CT by executing the CP.Decrypt primitive, thus obtaining the symmetric key K_S (step 5). Finally, the actuator adds the tuple to its policy list (step 6).

Data Exchange. This procedure (Figure 5.4) is executed by a sensor to transmit sensed data to one or more actuators in a low-latency fashion within the WSAN.



1) choose $\mathcal{T}$ and retrieve matching tuple
policy list
Policy ID
Policy
Sym.Key
...
P_{ID}
$\mathcal{T}$
K_S
2) encrypt data with K_S
3) broadcast
Sign $(S_{ID}, (\text{Enc}(\text{data}, K_S), P_{ID})), S_{ID}$
4) delete $\text{Enc}(\text{data}, K_S)$, data
5) verify
Sign $(S_{ID}, (\text{Enc}(\text{data}, K_S), P_{ID}))$
6) retrieve tuple with matching P_{ID}
policy list
Policy ID
Policy
Sym.Key
...
P_{ID}
$\mathcal{T}$
K_S
7) decrypt $\text{Enc}(\text{data}, K_S)$ and obtain data

Figure 5.4: Data exchange procedure.

In step 1, the sensor chooses an access policy $\mathcal{T}$ for the sensed data, and checks in the policy list if such access policy is present. If not, the sensor performs a new policy installation procedure. Otherwise, the sensor retrieves the matching tuple. In step 2, the sensor encrypts the data with the symmetric key K_S and creates a *data message*, which is a signed message that includes the sensor ID S_{ID}, the encrypted data $K_S(\text{data})$, and the policy identifier P_{ID}. In step 3, the sensor broadcasts the data message, and then, in step 4, it securely deletes the sensed data. Upon receiving the message, each actuator verifies the signature to be valid and checks in the policy list if the received policy identifier is present (step 5). If not, the message is discarded. Otherwise, the actuator retrieves the matching tuple and decrypts the ciphertext with the symmetric key K_S, thus obtaining the sensed data (step 6).

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 surely a threat to confidentiality. An eavesdropper can try to gain information by examining the traffic between sensors, actuators and the controller. However, every exchange of information is protected. If an eavesdropper is able to obtain a data message, he cannot access the data since he does not have the symmetric key. Even if he obtains the CP-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 controller during the actuator join procedure, he cannot retrieve the decryption key since it is safely encrypted with the actuator's public key.

Compromised Sensor. Suppose that an adversary gains complete access to a sensor. Such adversary would obtain: (i) the data generated by the sensor from the moment of compromise on; (ii) the sensor's private key; and (iii) its policy list, which contains the symmetric keys used for data encryption. Note that this adversary cannot, in any way, obtain data generated by other sensors. Moreover, since each sensor securely deletes past data, the adversary cannot retrieve it from the compromised sensor. However, if an adversary intercepted and stored past transmissions, he would be able to retrieve past data by using the stolen symmetric keys. In order to mitigate this, sensors could periodically refresh the symmetric keys by deleting some tuples from their policy list and executing again the new policy installation procedure on the same policies. In this way, the adversary cannot retrieve data produced before the last refresh of the symmetric key.

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

Compromised Actuator. Suppose that an adversary gains complete access to an actuator. Such adversary would obtain: (i) the actuator's ABE decryption key; (ii) its private key; and (iii) its policy list, which contains the symmetric keys used for data decryption. Each actuator may delete past data securely after consumption, so the adversary would not be able to retrieve it from the actuator. However, if an adversary intercepted and stored past transmissions, he would be able to retrieve past data by using the stolen symmetric keys. With this set of information, the adversary can decrypt every past and future data message that the compromised actuator has access to. Note that this does not imply that the adversary 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 access policy for such data, then the adversary 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 completely worked out by revoking its decryption key. We plan to add this functionality to a future version of fABElous.

5.4 Performance Evaluation

In this section we evaluate the communication overhead introduced by our scheme, and we compare it to other schemes. We assume the data, i.e., the message to be encrypted through symmetric cryptography, being composed of 120 bytes of raw data, 4 bytes of P_{ID}, and 4 bytes of timestamp (to avoid replay attacks). The symmetric key encryption scheme used was AES with 128-bit keys in CBC mode. The digital signature algorithm used was ECDSA, which has the benefit of adding a constant size signature of 40 byte (considering 80-bit security). As for CP-ABE, we used the tool developed by Bethencourt et al. (2007), which implements the CP-ABE primitives with 80-bit security. The access policy used for the evaluation was a simple, yet effective $\mathcal{T}_1 = (A \text{ AND } B \text{ AND } C)$. We used AND operators without loss of generality since the specific Boolean operator does not affect the communication overhead.

Table 5.1 shows the communication overhead of fABElous compared to other schemes. The *No security* scheme refers to sensors transmitting raw data without any

Scheme	Size (bytes)	Overhead
No security	120	0 %
Authentication only	160	25 %
“Naive” CP-ABE	1250	90 %
fABElous	192(+1122[†])	100 % – 37.5 %

[†]once per policy installation

Table 5.1: Transmission size comparison.

kind of protection. The *Authentication only* scheme refers to sensors transmitting data authenticated by means of ECDSA. The *“Naive” CP-ABE* scheme refers to sensors constantly transmitting data encrypted with CP-ABE. The fABElous communication overhead varies with the number of executions of the data exchange procedure, and it can be expressed as:

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

where N is the number of data messages sent, 192 is the total size in bytes of each data message, 72 is the amount of overhead in bytes of each data message, and 1122 is the size in bytes of the CP-ABE ciphertext containing an AES key. As shown in Table 5.1, the overhead introduced by fABElous varies from 100 % to 37.5 %. The overhead is at its maximum when $N = 0$, that is when no data exchange procedure has been executed after a new policy installation procedure execution, and this means that we exchanged only a new policy message, which is pure overhead. However, as the number of execution of data exchange procedure grows, the communication overhead considerably decreases down to 37.5 %.

Compared to the No security scheme, fABElous introduces 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 the Authentication only scheme, fABElous introduces more than twice the amount of 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 the “Naive” CP-ABE scheme, fABElous introduces less communication overhead since the second data exchange execution $(N \geq 2)$. However, in addition to data confidentiality and fine-grained access control, fABElous also grants data integrity, which is a feature required by our use case.

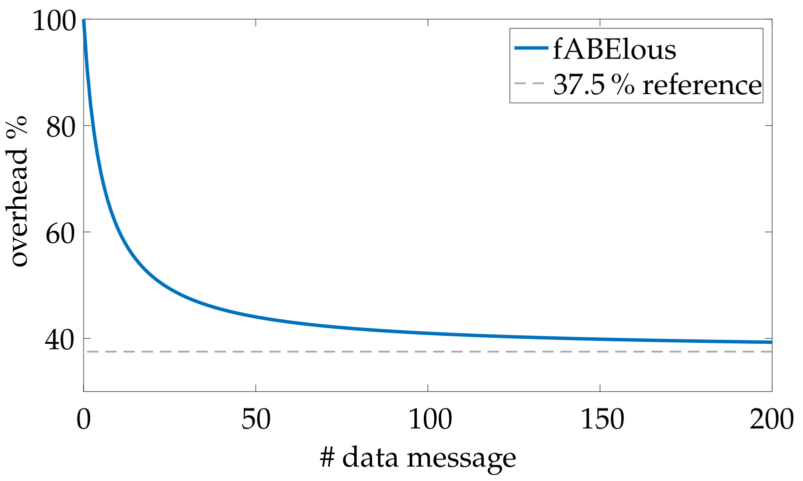


Figure 5.5: fABElous communication overhead.

Figure 5.5 shows how fABElous communication overhead drops with each execution of the data exchange procedure. The communication overhead asymptotically reaches 37.5 % and noticeably drops below 41 % after only 100 executions of the data exchange procedure.

Chapter 6

Conclusions

In this Ph.D. dissertation we approached some security and privacy problems related to IoT applications.

First, we addressed the problem of data and position trustworthiness in participatory sensing, and we proposed a low-cost approach that uses a swarm of UAVs to securely verify devices' positions by means of received power outdoor environments. We modeled the system and ran a set of experimental evaluations. The choice of the formation was a crucial point to make the system secure. We faced adversaries reporting false locations, and adversaries trying to deceive the location verification process through transmission power adaptation. Compared to state-of-the-art solutions, which either make use of special hardware or require strict clocks synchronization to detect attacks with high precision (false reported positions within one meter), our location verification approach achieves good results in term of attack detection in the order of meters. By using common hardware, our solution is intended to be easily integrated in participatory sensing applications, where users are equipped with their smartphones.

Secondly, we proposed the use of Attribute-Based Encryption in IoT scenarios to protect the data from unauthorized access. We proposed a secure scheme for smart city applications in which the data is outsourced to a semi-trusted cloud server. We addressed open challenges related to the employment of ABE in IoT, namely the high cost of KP-ABE encryption on resource-constrained devices and the computational burden caused by key revocation operations. In ABE-Cities, the sensing devices execute only lightweight symmetric-key encryption, thus we can employ constrained sensing devices such as battery-powered motes. This makes ABE-Cities suitable for a broader set of smart city applications. Moreover, ABE-Cities takes into account the peculiarities of a smart city, such as its street network, the geographical distribution of the sensor network, etc., and maps them onto the underlying KP-ABE scheme in order to gain efficiency in key revocation procedures. We proved that ABE-Cities scales well with the number of users and the number of streets by simulating it on large cities, i.e., Houston and Beijing.

Next, we extended YWRL scheme (Yu et al., 2010a) to improve its security against the cloud server, otherwise capable of accessing the stored data under the assumption that it comes in possession of a revoked key. In order to do that, the proposed scheme splits the key revocation task on both the cloud server and the users. We compared our scheme with the original YWRL, and we showed that the additional costs required to protect against such threat mainly weigh on the user.

Finally, we showed how ABE can be integrated in industrial IoT systems, specifically in low-bitrate wireless sensor and actuator networks. We proposed fABElous, a CP-ABE scheme which provides integrity, authentication, confidentiality, and access control on data. We addressed the challenge of providing all the aforementioned security properties while keeping the communication overhead as low as possible, and we showed how it outperforms a system which integrates CP-ABE naively.

Appendix A

Publications

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

Journal papers

1. **M. Rasori**, P. Perazzo, G. Dini, "A Lightweight and Scalable Attribute-Based Encryption System for Smart Cities", *Journal of Computer Communications*, volume: 149, pages: 78–89, 2020. **Candidate's contributions**: Conceptualization, data curation, formal analysis, investigation, methodology, software, validation, visualization, writing - original draft, writing - review & editing.

Peer reviewed conference papers

2. **M. Rasori**, P. Perazzo, G. Dini, "A Low-Cost UAV-Based Secure Location Verification Method", *Proceedings of the 12th International Conference on Availability, Reliability and Security*, pages: 30:1–30:6, 2017. **Candidate's contributions**: Conceptualization, data curation, formal analysis, investigation, methodology, software, validation, visualization, writing - original draft, writing - review & editing.

3. S. Chessa, M. Girolami, F. Mavilia, G. Dini, P. Perazzo, **M. Rasori**, "Sensing the Cities with Social-Aware Unmanned Aerial Vehicles", *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages: 278–283, 2017. **Candidate's contributions**: Software.

4. **M. Rasori**, P. Perazzo, G. Dini, "ABE-Cities: An Attribute-Based Encryption System for Smart Cities", *2018 IEEE International Conference on Smart Computing*

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

(*SMARTCOMP*), pages: 65–72, 2018. **Candidate's contributions**: Conceptualization, data curation, formal analysis, investigation, methodology, software, validation, visualization, writing - original draft, writing - review & editing.

Workshop papers

5. M. La Manna, P. Perazzo, **M. Rasori**, G. Dini, "fABElous: An Attribute-Based Scheme for Industrial Internet of Things", *2019 IEEE International Conference on Smart Computing* (*SMARTCOMP*), pages: 33–38, 2019. **Candidate's contributions**: Visualization, writing - review & editing.

Other

6. **M. Rasori**, P. Perazzo, G. Dini, "Attribute-Based Encryption for Smart Cities", *4th Italian Conference on ICT for Smart Cities And Communities* (*I-CiTies 2018*). **Candidate's contributions**: Conceptualization, data curation, formal analysis, investigation, methodology, software, validation, visualization, writing - original draft, writing - review & editing.

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 analyse or synthesize study data.
Funding acquisition	Acquisition of the financial support for the project leading to this publication.
Investigation	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.

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

Bibliography

Akinyele, J. A., Pagano, M. W., Green, M. D., Lehmann, C. U., Peterson, Z. N., and Rubin, A. D. (2011). Securing electronic medical records using attribute-based encryption on mobile devices. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 75–86. ACM.

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.

Andersen, J. B., Rappaport, T. S., and Yoshida, S. (1995). Propagation measurements and models for wireless communications channels. *IEEE Communications Magazine*, 33(1):42–49.

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

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.

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

Baden, R., Bender, A., Spring, N., Bhattacharjee, B., and Starin, D. (2009). Persona: an online social network with user-defined privacy. *SIGCOMM Comput. Commun. Rev.*, 39.

Baker, R. and Martinovic, I. (2016). Secure location verification with a mobile receiver. In *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, pages 35–46. ACM.

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.

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.

Coppolino, L., D'Antonio, S., Mazzeo, G., and Romano, L. (2017). Cloud security: Emerging threats and current solutions. *Computers & Electrical Engineering*, 59:126–140.

Costa, F. G., Ueyama, J., Braun, T., Pessin, G., Osório, F. S., and Vargas, P. A. (2012). The use of unmanned aerial vehicles and wireless sensor network in agricultural applications. In *2012 IEEE International Geoscience and Remote Sensing Symposium*, pages 5045–5048. IEEE.

De Berg, M., Van Kreveld, M., Overmars, M., and Schwarzkopf, O. C. (2000). Segment trees. In *Computational geometry*, pages 231–237. Springer.

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.

Ding, S., Li, C., and Li, H. (2018). A novel efficient pairing-free CP-ABE based on elliptic curve cryptography for IoT. *IEEE Access*, 6:27336–27345.

Du, W., Deng, J., Han, Y. S., Chen, S., and Varshney, P. K. (2004). A key management scheme for wireless sensor networks using deployment knowledge. In *INFOCOM 2004. Twenty-third AnnualJoint conference of the IEEE computer and communications societies*, volume 1. IEEE.

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

Fischer, M., Scheerhorn, A., and Tönjes, R. (2019). Using attribute-based encryption on IoT devices with instant key revocation. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 126–131. IEEE.

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

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.

Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006). 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.

Halfond, W. G., Viegas, J., Orso, A., et al. (2006). A classification of SQL-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, volume 1, pages 13–15. IEEE.

Hu, C., Zhang, N., Li, H., Cheng, X., and Liao, X. (2013). Body area network security: a fuzzy attribute-based signcryption scheme. *IEEE journal on selected areas in communications*, 31(9):37–46.

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.

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. (2010). Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1214–1221.

Ibraimi, L., Asim, M., and Petković, M. (2009). Secure management of personal health records by applying attribute-based encryption. In *Wearable Micro and Nano Technologies for Personalized Health (pHealth), 2009 6th International Workshop on*, pages 71–74. IEEE.

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.

Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. (2018). Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203*.

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.

Li, J., Jia, C., Li, J., and Chen, X. (2012). Outsourcing encryption of attribute-based encryption with mapreduce. In *International Conference on Information and Communications Security*, pages 191–201. Springer.

Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., et al. (2018). Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 973–990.

Lounis, A., Hadjidj, A., Bouabdallah, A., and Challal, Y. (2012). Secure and scalable cloud-based architecture for e-health wireless sensor networks. In *Computer communications and networks (ICCCN), 2012 21st international conference on*, pages 1–7. IEEE.

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.

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.

Montenegro, G., Kushalnagar, N., Hui, J., and Culler, D. (2007). RFC 4944. *Transmission of IPv6 packets over IEEE*, 802(4).

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.

Oualha, N. and Nguyen, K. T. (2016). Lightweight attribute-based encryption for the internet of things. In *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*, pages 1–6. IEEE.

Perazzo, P., Ariyapala, K., Conti, M., and Dini, G. (2015). The verifier bee: A path planner for drone-based secure location verification. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a*, pages 1–9. IEEE.

Perazzo, P., Sorbelli, F. B., Conti, M., Dini, G., and Pinotti, C. M. (2016a). Drone path planning for secure positioning and secure position verification. *IEEE Transactions on Mobile Computing*.

Perazzo, P., Taponecco, L., D'amico, A. A., and Dini, G. (2016b). Secure positioning in wireless sensor networks through enlargement miscontrol detection. *ACM Transactions on Sensor Networks (TOSN)*, 12(4):27.

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.

Rasmussen, K., Srivastava, M., et al. (2008). Secure location verification with hidden and mobile base stations. *IEEE Transactions on Mobile Computing*, 7(4):470–483.

Rasori, M., Perazzo, P., and Dini, G. (2017). A low-cost UAV-based secure location verification method. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, pages 30:1–30:6. ACM.

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.

Roy, S. and Chuah, M. (2009). Secure data retrieval based on ciphertext policy attribute-based encryption (CP-ABE) system for the DTNs. *Lehigh CSE Tech. Rep.*

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.

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.

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.

Čapkun, S. and Hubaux, J.-P. (2006). Secure positioning in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(2):221–232.

Vora, A. and Nesterenko, M. (2004). Secure location verification using radio broadcast. In *International Conference on Principles of Distributed Systems*, pages 369–383. Springer.

Waharte, S., Trigoni, N., and Julier, S. (2009). Coordinated search with a swarm of UAVs. In *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops*, pages 1–3. IEEE.

Wang, G., Liu, Q., and Wu, J. (2010). Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 735–737. ACM.

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.

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.

Yanmaz, E., Kuschnig, R., and Bettstetter, C. (2011). Channel measurements over 802.11a-based UAV-to-ground links. In *GLOBECOM Workshops (GC Wkshps), 2011 IEEE*, pages 1280–1284. 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.

Yokoyama, R. S., Kimura, B. Y. L., and dos Santos Moreira, E. (2014). Secure positioning in a UAV swarm using on-board stereo cameras. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 769–774. 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.

Zeng, Y., Cao, J., Hong, J., Zhang, S., and Xie, L. (2013). Secure localization and location verification in wireless sensor networks: a survey. *the Journal of Supercomputing*, pages 1–17.

Zheng, Y. (2011). kpabe. `https://github.com/altu341com/kpabe`. (Last accessed: 1 April 2019).

Zuo, C., Shao, J., Wei, G., Xie, M., and Ji, M. (2018). CCA-secure ABE with outsourced decryption for fog computing. *Future Generation Computer Systems*, 78:730–738.