# ABE-Cities: An Attribute-Based Encryption System for Smart Cities

Marco Rasori*
*DINFO*
*University of Florence*
Florence, Italy
marco.rasori@unifi.it

Pericle Perazzo
*Department of Information Engineering*
*University of Pisa*
Pisa, Italy
pericle.perazzo@iet.unipi.it

Gianluca Dini
*Department of Information Engineering*
*University of Pisa*
Pisa, Italy
gianluca.dini@iet.unipi.it

***Abstract*—In the near future, a technological revolution will involve our cities, where a variety of smart services based on the Internet of Things will be developed to facilitate the needs of the citizens. Sensing devices are already being deployed in urban environments, and they will generate huge amounts of data. Such data are typically outsourced to some cloud storage because this lowers capital and operating expenses and guarantees high availability. However, cloud storage may have incentives to release stored data to unauthorized entities. In this work we present ABE-Cities, an encryption scheme for urban sensing which solves the above problems while ensuring fine-grained access control on data by means of Attribute-Based Encryption (ABE). Basically, ABE-Cities encrypts data before storing it in the cloud and provides users with keys able to decrypt only those portions of data the user is authorized to access. In ABE-Cities, the sensing devices perform only lightweight symmetric cryptography operations, thus they can also be resource-constrained. ABE-Cities provides planned expiration of keys, as well as their unplanned revocation. We propose methods to make the key revocation efficient, and we show by simulations the overall efficiency of ABE-Cities.**

***Index Terms*—Smart City, Urban Sensing, Attribute-Based Encryption, Internet of Things, Data-Centric Security**

## I. Introduction

Smart cities offer digital services to facilitate the needs of their citizens and improve their quality of life. Such services span across all sectors of society including health, logistics, and mobility. Services often capitalize on the Internet of Things (IoT) as enabling technology. For example, cameras could be deployed in the city so that citizens can monitor traffic and detect possible congestions on the usual route from home to work. In general, smart devices underlying these services produce a large amount of data which can be outsourced to a Cloud Storage Service (CSS). 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 a set of authorized users. Unfortunately, as soon as data lands on the cloud, we lose any control on it and we have to totally trust the CSS. Cloud service providers may have some incentive to release stored information to others [1], [2]. It follows that the CSS is generally assumed *honest-but-curious*, meaning that it trustworthy carries out data distribution and elaboration functions but is interested in accessing the resource contents.

A possible solution to this problem is to store an encrypted version of the data on the CSS. An efficient way to do this while ensuring fine-grained access control on data is to use the *Attribute-Based Encryption* (ABE) [3] technology. ABE allows us to label a ciphertext with a set of *attributes*. A *Trusted Third Party* (TTP) generates *decryption keys* and provides them to users. Decryption keys embed an access policy over the attributes, and they are capable of decrypting a ciphertext only if the set of attributes labelling the ciphertext matches the access policy.

In this paper we present ABE-Cities, an encryption system for urban sensing in a smart city based on Attribute-Based Encryption. ABE-Cities allows for fine-grained access control over the encrypted data stored in the CSS, and it is secure against multiple adversary models: a honest-but-curious cloud service, external adversaries capable of eavesdropping traffic and compromising sensing devices, and colluding users wanting to gain more authorizations illegally. Sensing devices perform only lightweight symmetric cryptography operations. Therefore, ABE-Cities can employ also resource-constrained sensing devices, which makes it suitable for a broader set of IoT applications for smart cities. Our system also provides mechanisms to enforce planned expiration of decryption keys, as well as their unplanned revocation. We propose methods to make the unplanned revocation efficient from the point of view of the cloud service. We show by simulations the effectiveness of such improvements.

The paper is organized as follows: in Section II we compare with related work. In Section III we introduce ABE and other techniques that we use in our system. In Section IV we describe ABE-Cities and our adversary models. In Section V we focus on the access policies, whose structure is paramount for the system performance. In Section VI we evaluate the performance of ABE-Cities. Section VII summarizes our results and concludes the paper.

## II. Related Work

Yu et al. [4] first used ABE techniques for outsourcing sensitive data to a honest-but-curious cloud storage, while enforcing fine-grained access control at the same time. Their

* Also affiliated with University of Pisa, Dept. of Information Engineering.

scheme is well applicable for example in ehealth, where patients produce sensitive healthcare records and enjoy full computational capabilities. However, it is less suitable for IoT applications, in which data is produced by constrained device, which could not have the necessary computational power and energy to produce ABE ciphertexts. In our system, the IoT devices execute only lightweight operations of symmetric cryptography. The authors of [4] also proposed a technique to delegate the burdensome re-encryption of old ciphertexts to the cloud servers, in order to make a revoked key unable to decrypt them. We use such a re-encryption technique as a building block in our scheme.

Wang et al. [5] proposed a hierarchical ABE scheme that allows the TTP to delegate part of the responsibility to other authorities, which independently make decisions on the semantics of their attributes. Hierarchical ABE allows also for proxy re-encryption, but it forces the access policies to have a fixed structure, so it limits the flexibility of the access control system. In this paper, we focused on a non-hierarchical ABE scheme with a single TTP, leaving the possible hierarchical extensions as future work.

Yu et al. [6] proposed an ABE scheme to encrypt data sensed by a wireless sensor network (WSN). The authors used broadcast encryption in order to perform key revocations efficiently. 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 is impossible in these scenarios. Moreover, the authors propose to perform ABE encryption directly on the sensing devices, which could not have the necessary computational power and energy to do it. In our system, we propose a key revocation method whose efficiency is based on a convenient attribute representation of the smart city. In addition, in our system the IoT devices execute only lightweight operations of symmetric cryptography.

Yao et al. [7] proposed an ABE scheme based on elliptic curves instead of pairings. This makes ABE operations more lightweight and more suitable for typical IoT constrained devices. In our scheme, the IoT devices execute only lightweight operations of symmetric cryptography. This permits us to use the pairing-based ABE scheme of Goyal et al. [3], which allows us to employ advanced features like proxy re-encryption [4].

Recently, Odelu et al. [8] proposed an ABE scheme suitable for IoT scenarios because encryption and decryption times are $\mathcal{O}(1)$. However, such scheme forces the access policies to have a fixed structure, so it limits the flexibility of the access control system. On the contrary, our system is based on Goyal's ABE, which offers a greater degree of expressiveness.

## III. Preliminaries

### A. Key-Policy Attribute-Based Encryption

Key-Policy Attribute-Based Encryption (KP-ABE) is a public-key encryption scheme based on bilinear pairings. It was introduced by Goyal et al. [3] as an extension of the original ABE proposed by Sahai and Waters [9]. In KP-ABE everyone can encrypt because who encrypts uses only public parameters. The ciphertexts are labelled with a set of attributes which describes them (*encryption attributes*, $\gamma$). Decryption keys embed an *access policy* ($\mathcal{T}$), and they are capable of decrypting a ciphertext only if the encryption attributes on the ciphertext match the policy. Who encrypts data is said a *data producer*, whereas who holds a decryption key and decrypts data is said a *data consumer*.

An access policy is a monotonic Boolean formula on the presence of some attributes on the ciphertext. It can be represented as a tree in which leaf nodes are attributes (*access policy attributes*, $\lambda$), and non-leaf nodes are AND/OR operators. For example, let us suppose that a data consumer holds a decryption key embedding the access policy $\mathcal{T} = (A \wedge (B \vee C \vee D))$ (and thus, its access policy attributes are $\lambda = \{A, B, C, D\}$). This access policy reads as follows: "the attribute $A$ must be present in the ciphertext, and one of the attributes $B$, $C$, and $D$ must be present as well." A ciphertext labelled, for example, with the encryption attributes $\gamma = \{A, C, E\}$ can be decrypted by the above decryption key. Goyal's KP-ABE does not allow for non-monotonic access policies, i.e., Boolean formulas including NOT operators.

KP-ABE is resilient to collusion, meaning that two decryption keys cannot be combined somehow to decrypt a ciphertext that they could not decrypt singly. The set of all the attributes used in a given KP-ABE scheme is the *universe of the attributes* ($\mathcal{U}$) for such scheme. Without losing in generality, in the following we will indicate an attribute in the universe either with its unique name, or with a unique natural number which is more convenient in formulas. The attribute sets $\mathcal{U}$, $\gamma$, and $\lambda$ are thus subsets of $\mathbb{N}$.

In order to ease the reading, we abstract away from this paper the mathematical insight of the KP-ABE scheme. The interested reader can refer to [3] for such details. We model the KP-ABE scheme with the following black-box primitives.

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

*2)* $E = \mathrm{Encrypt}(M, \gamma, Y, T_{i \in \gamma})$*:* This primitive encrypts a plaintext $M$ with the encryption attributes $\gamma$. It takes as input $Y$ and $T_{i \in \gamma}$, which are all public parameters. It produces the ciphertext $E = (\gamma, \tilde{e}, e_{i \in \gamma})$. Each attribute $i$ in the encryption attribute set is associated to a *ciphertext component* $e_i$. The $\mathrm{Encrypt}$ primitive is executed by a data producer.

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

*4)* $M = \mathrm{Decrypt}(E, DK)$*:* This primitive decrypts the ciphertext $E$ given a decryption key $DK$. It produces the plaintext $M$ if the decryption key is able to decrypt the ciphertext, $\perp$ otherwise. The $\mathrm{Decrypt}$ primitive is executed by a data consumer.

### B. 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. Yu et al. [4] introduced a PRE technique for the KP-ABE scheme. 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 honest-but-curious CSS without compromising data confidentiality.

Yu et al.'s PRE implements the revocation of a decryption key by means of a system-wide *update* of a subset $\mu$ of its access policy attributes ($\mu \subseteq \lambda$). In particular, the subset $\mu$ is such that, without those attributes in a ciphertext, the access policy will surely not be satisfied. Updating an attribute $i$ to a new *version* simply means that all the relative cryptographic quantities in the system are changed. Specifically, for all the attributes $i \in \mu$, the quantities $t_i$, $T_i$, the $e_i$'s of all the old ciphertexts, and the $dk_i$'s of all the decryption keys except those of the revoked key are updated with new quantities $t'_i$, $T'_i$, $e'_i$, and $dk'_i$. The burdensome computation of the updated $dk'_i$ can be outsourced to the CSS as well. From now on, we implicitly consider the quantities $t_i$, $T_i$, $e_i$, and $dk_i$ always accompanied by the information about their versions.

The computation of the $e'_i$'s and the $dk'_i$'s is delegated to a honest-but-curious CSS. To do this, the TTP computes a *re-encryption key* ($rk_i$) and sends it to the CSS, which can apply it to an $e_i$ to obtain an $e'_i$, and to a $dk_i$ to obtain a $dk'_i$. The computation of the $dk'_i$'s cannot be completely outsourced, since the CSS would then know all the decryption keys, so it could violate the data confidentiality. To avoid this, the CSS is provided with all the decryption key components except 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 policies of all the decryption keys, and it is included as an encryption attribute in all the 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. Thus, only the data consumer can effectively use his/her decryption key.

Proxy re-encryption is performed in a lazy fashion (*lazy re-encryption*), meaning that the CSS does not perform any burdensome operation at the moment of the key revocation, but only afterwards when needed. At the moment of revocation, the TTP produces a re-encryption key for each attribute $i$ in $\mu$ and adds it to a *re-encryption key history* ($RKH_i$) stored in the CSS. $RKH_i$ is a list that keeps track of all the re-encryption keys relative to the attribute $i$, one for each version update of the attribute ($rk_i, rk'_i, rk''_i, \dots$). Only when a data consumer asks the CSS for a ciphertext whose components are outdated, then the CSS computes the new components. If a ciphertext component was outdated of more than one version, then the CSS will use more than one re-encryption key from the history to update it. Similarly, if the consumer's decryption key has outdated components, then the CSS computes the new components. If a decryption key component was outdated of more than one version, then the CSS will use more than one re-encryption key from the history to update it. Finally, the CSS provides the data consumer with the ciphertext and, possibly, with the new decryption key components. Note that the new decryption key components can be transmitted to the data consumer in the clear, without compromising the secrecy of the key. Indeed, an adversary cannot use the decryption key components unless she knows also the one relative to the dummy attribute, which is never updated.

In order to ease the reading, we abstract away from this paper the mathematical insight of the PRE scheme. The interested reader can refer to [4] for such details. We model the PRE scheme with the following black-box primitives.

*1)* $(t'_i, T'_i, rk_i) = \mathrm{UpdateAttribute}(i, MK)$*:* This primitive updates the attribute $i$, meaning that it produces the new quantities $t'_i$, $T'_i$, and the re-encryption key $rk_i$. The $\mathrm{UpdateAttribute}$ primitive is executed by the TTP.

*2)* $e'_i = \mathrm{UpdateE}(i, e_i, RKH_i)$*:* This primitive updates a ciphertext component to a new version. The $\mathrm{UpdateE}$ primitive is executed by the CSS.

*3)* $dk'_i = \mathrm{UpdateDK}(i, dk_i, RKH_i)$*:* This primitive updates a decryption key component to a new version. The $\mathrm{UpdateDK}$ primitive is executed by the CSS.

### C. Segment Trees

A segment tree is a data structure that represents a set of intervals [10]. It allows us to efficiently query which intervals contain a given point. Segment trees are useful in ABE schemes to implement efficient point-in-interval access policies [11]. Although segment trees can in principle represent any point and interval in $\mathbb{R}$, in this paper we focus on *discrete* segment trees, i.e., segment trees that represent points and intervals in $\mathbb{Z}$. Specifically, we focus on discrete segment trees capable of representing any point and any interval 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 type of discrete segment tree.

A segment tree over the range $[1, \rho]$ is a binary tree in which each number in $[1, \rho]$ is relative to a leaf. For example, Fig. 1 shows a segment tree over the range $[1, 7]$. The leaf $l1$ is relative to the number 1, the leaf $l2$ to the number 2, and so on. A generic point $\nu$ in the range $[1, \rho]$ is represented by a *point representation set* $RS_\nu$, 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 Fig. 1, the point $\nu = 5$ is represented by the point representation set $RS_\nu = \{n6, n5, n3, l5\}$. It can be shown that every point representation set is $\mathcal{O}(\log \rho)$ nodes. On the other hand, a generic interval $\iota$ included in the range is represented by an *interval representation set* $RS_\iota$, which is the minimum set of nodes whose descendant leaves form

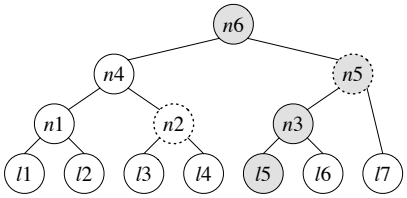



Fig. 1. Example of a segment tree. Grey nodes form the point representation set of 5, dashed-border nodes form the interval representation set of $[3, 7]$.

the interval. For example, in the segment tree of Fig. 1, the interval $\iota = [3, 7]$ is represented by the interval representation set $RS_\iota = \{n2, n5\}$. It can be shown that every interval representation set is $\mathcal{O}(\log \rho)$ nodes. By construction, $RS_\nu$ and $RS_\iota$ have non-empty intersection iff the point $\nu$ belongs to the interval $\iota$: $\nu \in \iota \Leftrightarrow RS_\nu \cap RS_\iota \neq \emptyset$. For example, in the segment tree of Fig. 1, the point $\nu = 5$ belongs to the interval $\iota = [3, 7]$ because $RS_\nu \cap RS_\iota = \{n5\} \neq \emptyset$.

Segment trees are useful in ABE schemes to implement efficient point-in-interval access policies [11], i.e., access policies which are satisfied iff a given point $\nu$ belongs to a given interval $\iota$. According to the terminology in [11], 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 ABE attribute. The ciphertext is labelled with the attributes representing the nodes of the point representation set $RS_\nu$. The access policy is an OR operator between the nodes representing the interval representation set $RS_\iota$. By construction, the access policy is satisfied iff $RS_\nu$ and $RS_\iota$ have non-empty intersection, that is, iff $\nu$ is in $\iota$.

## IV. Proposed Scheme

In our system, a city is represented by a *street network*, i.e., a graph in which the edges represent *streets*. Each street is characterized by an identifier and is partitioned into *road segments*. A *sensing device* is a possibly constrained device placed on a road segment that acquires data (*sensed data*, $SD$) relative to such a road segment. The sensed data is stored in an encrypted form in a *Cloud Storage Service* (CSS), where it is made read-only accessible to the *users*. The users are the data consumers of our system.

A user can be authorized to access only data sensed from some regions of the city, specified as a set of road segments. For example, supposing the sensing devices to be cameras, the user can monitor the traffic congestion on the route he/she usually travels from home to work. To do this, the cameras could store in the CSS many short video files to implement an encrypted streaming service. The user can also monitor multiple routes, in order to identify the least congested one day by day. Moreover, some advanced-privilege users could be provided, for example a transportation management service which can monitor all the road segments of the city.

The architecture of our system is shown in Fig. 2.

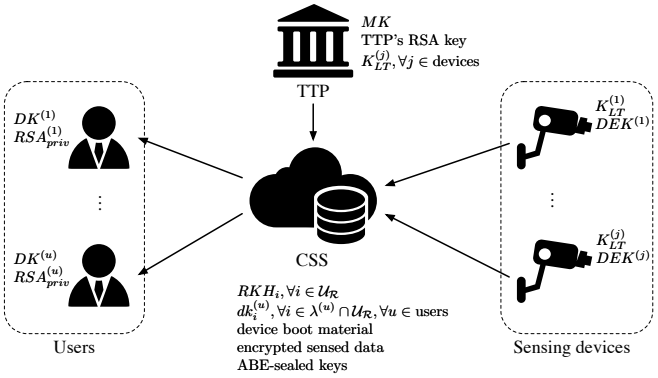



Fig. 2. System architecture.

In the following, we will give a general and intuitive description of the system while in Section IV-B we will describe the system procedures in detail. The TTP holds the KP-ABE master key and an RSA key which it uses to sign messages. There is nothing special in choosing RSA as digital signature algorithm, elliptic-curve digital signature algorithms (EC-DSA) are suitable as well. The notation $\mathrm{Sign}(\cdot)$ is used to represent the TTP's signature on something. The TTP also shares with each sensing device $j$ a *long-term symmetric key* ($K_{LT}^{(j)}$), which is supposed to be preloaded in the sensing devices. Each sensing device holds also a *data encryption key* ($DEK^{(j)}$), which is the actual symmetric key used to encrypt the sensed data. The data encryption key is frequently renewed by the sensing device, once for each produced piece of data. Each user $u$ holds a KP-ABE decryption key $DK^{(u)}$, and an RSA key $RSA_{priv}^{(u)}$ which he/she uses to decrypt messages received from the TTP. We denote by $RSA_{pub}^{(u)}$ the corresponding public key. The CSS maintains a database of re-encryption key histories and decryption key components to perform proxy re-encryption. In addition, it stores *device boot material*, *encrypted sensed data*, and *ABE-sealed keys*. The device boot material is needed by the sensing devices to encrypt their data. The encrypted sensed data is the actual data produced by the sensing devices and encrypted with a symmetric encryption algorithm. The ABE-sealed keys are necessary to decrypt the sensed data and are in turn encrypted (*sealed*) with KP-ABE. The CSS stores an ABE-sealed key for each sensing device and for each day of system operation. To access a piece of sensed data, users must first decrypt the corresponding ABE-sealed key and then decrypt the sensed data.

The universe of the attributes $\mathcal{U}$ is logically divided into two subsets: $\mathcal{U}_\mathcal{R}$ and $\mathcal{U}_\mathcal{X}$. $\mathcal{U}_\mathcal{R}$ includes the attributes to represent road segments. $\mathcal{U}_\mathcal{X}$ includes the attributes to represent time. Each ABE-sealed key relative to the sensing device $j$ is labelled with the encryption attributes $\gamma^{(j)}$, which are logically divided into two subsets: $\gamma_\mathcal{R}^{(j)}$ and $\gamma_\mathcal{X}^{(j)}$. The attributes in $\gamma_\mathcal{R}^{(j)}$ identify the device's road segment, while the attributes in $\gamma_\mathcal{X}^{(j)}$ identify the day in which data has been sensed (*data production date*). Similarly, the access policy $\mathcal{T}$ embedded in

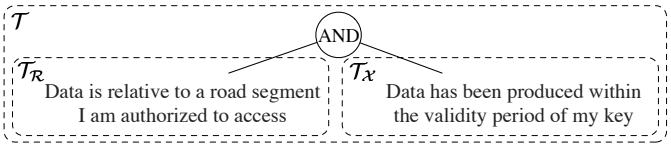



Fig. 3. Example of access policy for a user's decryption key in our system. In order to decrypt an ABE-sealed key, the user's decryption key must satisfy both the subtrees.

each decryption key is logically divided into two subtrees: $\mathcal{T}_\mathcal{R}$ and $\mathcal{T}_\mathcal{X}$ (Fig. 3). 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. The $\mathcal{T}_\mathcal{R}$ subtree identifies the road segments the user is authorized to monitor (*authorized road segments*) while $\mathcal{T}_\mathcal{X}$ specifies the *validity period* of the key. The validity period is the period of time from the day the user joins the system to the day in which his/her subscription ends. Hence, the user is able to decrypt sensed data produced during his/her validity period by sensing devices belonging to his/her authorized road segments.

Note that, in order to implement proxy re-encryption, the CSS is let know of all the re-encryption keys except those relative to the attributes in $\mathcal{U}_\mathcal{X}$, and all the decryption key components except those relative to the $\mathcal{T}_\mathcal{X}$ subtree. The presence of at least one attribute in $\mathcal{U}_\mathcal{X}$ is ANDed at the root of the access policy of all the decryption keys. In practice, the attributes in $\mathcal{U}_\mathcal{X}$, which are never updated to a new version, replace the dummy attribute of the Yu et al.'s PRE scheme [4].

### A. Adversary Model(s) and Security Analysis

We consider three types of adversary: (i) the honest-but-curious CSS, (ii) an external adversary capable of eavesdropping traffic and compromising sensing devices, and (iii) a set of colluding users wanting to gain more authorizations illegally. The CSS is assumed to be honest-but-curious, meaning that it trustworthy carries out data distribution and elaboration functions but is interested in reading the stored sensed data. To decrypt the stored sensed data, the CSS should first decrypt the ABE-sealed key, which is encrypted with KP-ABE. However, the CSS does not know any *complete* decryption key to do this. In particular, it knows none of the decryption key components relative to the attributes in $\mathcal{U}_\mathcal{X}$, whose presence is always ANDed at the root of all the access policies.

The external adversary can eavesdrop all the traffic between the entities of our system. Her goal is to read the sensed data. To achieve this, she could either steal decryption keys, or inject malicious device boot material to make the sensing devices encrypt data with a compromised key. However, none of these tactics are viable. Stealing decryption keys is infeasible since the TTP sends them to newly joined users encrypted with RSA. Injecting malicious device boot material to a sensing device is infeasible too, since it is authenticated with the long-term secret key shared between the device and the TTP. Even if the external adversary compromises a sensing device, she can only read the new sensed data produced by such device after the compromise, but not the old one. Indeed, as we will explain in detail in Section IV-B, each device changes the data encryption key at each new piece of produced data. At the moment of compromise, only the current data encryption key is actually compromised. The old data encryption keys are securely destroyed by the sensing device and are not recoverable from the current one.

The third type of adversary is a set of users who collude to gain more authorizations illegally. This is infeasible because KP-ABE is resilient to collusion, meaning that two decryption keys cannot be combined somehow to decrypt an ABE-sealed key that they could not decrypt singly.

### B. System Procedures

*1) Setup Procedure:* This procedure initializes the system. The TTP defines the universe of the attributes $\mathcal{U} = \mathcal{U}_\mathcal{R} \cup \mathcal{U}_\mathcal{X}$ and then executes the $\mathrm{Setup}(\mathcal{U})$ primitive which produces the master key and the public parameters. The TTP keeps the master key secret and stores in the CSS an empty re-encryption key history for each attribute in $\mathcal{U}_\mathcal{R}$.

*2) Key Distribution Procedure:* This procedure provides a user with a new decryption key. 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 expired or it has been compromised.

The TTP first defines the access policy of the new user $u$. Which access policy assigning to each user depends strictly on the application. Once the TTP defined the user's access policy $\mathcal{T}$, it creates the user's decryption key $DK^{(u)}$ by executing the $\mathrm{KeyGen}(MK, \mathcal{T})$ primitive. The TTP encrypts the message $(DK^{(u)}, \text{cur.date}, \mathrm{Sign}(DK^{(u)}, \text{cur.date}))$, where cur.date is the current date, with the user's public key and provides it to the user either with a direct channel or through the CSS. The user decrypts it and verifies the TTP signature to be valid, and cur.date to be the actual current date. If everything is correct, the user accepts $DK^{(u)}$ as his/her decryption key. In the meanwhile, the TTP sends to the CSS the message $(u, dk_{i \in \lambda^{(u)} \cap \mathcal{U}_\mathcal{R}}, \text{cur.date}, \mathrm{Sign}(u, dk_{i \in \lambda^{(u)} \cap \mathcal{U}_\mathcal{R}}, \text{cur.date}))$, where $\lambda^{(u)}$ are the access policy attributes of the user. The CSS verifies the TTP signature to be valid, and cur.date to be the actual current date. If everything is correct, the CSS stores the decryption key components $dk_{i \in \lambda^{(u)} \cap \mathcal{U}_\mathcal{R}}$.

*3) Seal Procedure:* This procedure is executed once a day at the midnight.

For each sensing device $j$, the TTP generates a random data encryption key $DEK^{(j)}$ and encrypts it with the $\mathrm{Encrypt}(DEK^{(j)}, \gamma^{(j)}, PK)$ primitive. The result constitutes the ABE-sealed key ($ASK^{(j)}$) for the sensing device $j$, and it is stored in the CSS. The CSS stores an ABE-sealed key for each sensing device and for each day of system operation. Also, for each sensing device $j$ the TTP encrypts with the long-term symmetric key $K_{LT}^{(j)}$ the message $(DEK^{(j)}, \text{cur.date}, \mathrm{MAC}_{K_{LT}^{(j)}}(DEK^{(j)}, \text{cur.date}))$, where $\mathrm{MAC}_{K_{LT}^{(j)}}(\cdot)$ denotes a message authentication code keyed by $K_{LT}^{(j)}$. The result constitutes the device boot material for the sensing device $j$, and it is stored in the CSS too. The

CSS stores a device boot material for each sensing device. Each sensing device retrieves its device boot material and checks the MAC to be valid, and cur.date to be the actual current date. If everything is correct, the device accepts $DEK^{(j)}$ as its current data encryption key and initializes a *data encryption counter* ($c^{(j)}$) to zero.

*4) Data Production Procedure:* This procedure is executed every time a sensing device produces a new piece of data.

First, the device encrypts the sensed data $SD^{(j)}$ with its current data encryption key $DEK^{(j)}$. The result constitutes an encrypted sensed data $ESD^{(j)}$, which is stored in the CSS together with the current data encryption counter $c^{(j)}$. The CSS stores many tuples $(ESD^{(j)}, c^{(j)})$ for each sensing device. Then, the sensing device computes a new data encryption key as a one-way hash of the old one: $DEK^{(j)} \leftarrow \mathrm{H}(DEK^{(j)})$. Finally, the device securely destroys the old data encryption key and increments its data encryption counter. This key renewal method prevents an external adversary who physically compromises a sensing device from decrypting old encrypted sensed data, thus ensuring backward secrecy.

*5) Data Consumption Procedure:* This procedure is executed every time a user wants to access a piece of sensed data. The user $u$ first sends a data request to the CSS specifying which sensed data he/she wants to access. On receiving the data request, the CSS checks whether some of the stored decryption key components are out of date by comparing their versions with the latest version of the relative attribute. For each out-of-date component $dk_i$, the CSS updates it by executing: $dk'_i = \mathrm{UpdateDK}(i, dk_i, RKH_i)$, and it provides $dk'_i$ to the user. Then, the CSS checks whether some of the 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 $e_i$, the CSS updates it by executing: $e'_i = \mathrm{UpdateE}(i, e_i, RKH_i)$, and it replaces $e_i$ with the updated version. Finally, the CSS provides the ABE-sealed key $ASK^{(j)}$ and the tuple $(ESD^{(j)}, c^{(j)})$ relative to the requested data to the user. On receiving this, the user decrypts the ABE-sealed key by executing the $\mathrm{Decrypt}(ASK^{(j)}, DK^{(u)})$ primitive. Of course, if the user is not authorized to access the requested data, such primitive will return $\perp$. Otherwise, the user applies $c^{(j)}$ times the one-way hash $\mathrm{H}(\cdot)$ on the result, thus obtaining the data encryption key $DEK^{(j)}$ with which the encrypted sensed data $ESD^{(j)}$ was encrypted. The user can thus read the sensed data $SD$.

*6) Key Revocation Procedure:* Whenever a user $u$'s decryption key must be revoked, for example when his/her key is compromised, the system makes it ineffective to decrypt any ABE-sealed key by executing the key revocation procedure.

At first, the TTP determines the subset $\mu^{(u)} \subseteq \lambda^{(u)}$, 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 user $u$. In our system, the subset $\mu^{(u)}$ is formed by the attributes $\lambda^{(u)} \cap \mathcal{U}_{\mathcal{R}}$. For each attribute $i \in \mu^{(u)}$, the TTP updates the related quantities by executing: $(t'_i, T'_i, rk_i) = \mathrm{UpdateAttribute}(i, MK)$, and it replaces $t_i$ and $T_i$ with the updated versions. Then, the TTP sends the message $(u, rk_{i,\forall i \in \mu^{(u)}}, \text{cur.date}, \mathrm{Sign}(u, rk_{i,\forall i \in \mu^{(u)}}, \text{cur.date}))$ to the CSS. The CSS verifies the TTP signature to be valid, and cur.date to be the actual current date. If everything is correct, the CSS adds each re-encryption key $rk_i$ to the proper $RKH_i$ and erases 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. The user is now revoked, and his/her key is not capable of decrypting any ABE-sealed key anymore.

## V. Universe of the Attributes and Access Policies

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

*1) Basic Representation:* The simplest representation consists in mapping each road segment onto one attribute. In this way, the $\gamma^{(j)}_{\mathcal{R}}$ set of each ABE-sealed key is formed by just one attribute. On the other hand, the $\mathcal{T}_{\mathcal{R}}$ subtree of a decryption key is an OR between many attributes, one for each road segment the user is authorized to monitor. We will refer to this implementation as the *basic representation*.

By using this representation, the number of attributes of the key access policy, i.e., its *key size*, grows linearly with the number of authorized road segments. When a user is revoked, the attributes in the $\mathcal{T}_{\mathcal{R}}$ subtree of his/her key access policy must be updated. The users whose decryption keys shared any attribute, and thus any authorized road segment, with the revoked key are called *affected users*. An affected user is involved in the revocation process of another user, and his/her key must be updated by the CSS during the data consumption procedure, as explained in Section IV-B. Our aim is making the key revocation more efficient by limiting the number of affected users. This improvement lightens the load on the CSS in terms of proxy re-encryption operations.

*2) Segment Tree Representation:* In the basic representation, it is very likely that two users share some road segments, and, in this case, if one of them is revoked, the other will be surely an affected user, thus generating a *collision*. For example, a user authorized to monitor an entire street will collide with any other user authorized to monitor any subset of road segments of that street.

Segment trees, already used in ABE to reduce the key size [11], can help us to reduce the average number of affected users, too. For example, a user authorized to monitor an entire street will not collide with another user authorized to monitor only some road segments of that street. Let $\rho$ be the number of road segments of a generic street. We denote road segments of that street with identifiers from 1 to $\rho$ and build a segment tree in which leaf nodes are the road segment identifiers. Each node of the segment tree is represented by an ABE attribute.

The $\gamma^{(j)}_{\mathcal{R}}$ set of each ABE-sealed key is formed by a point representation set and contains $\mathcal{O}(\log \rho)$ attributes. Thus, the TTP is required of a little more effort to produce the ABE-sealed keys since the complexity of the Encrypt primitive depends on the size of the encryption attribute set $\gamma$. On the other hand, we use a point-in-interval access policy to identify the subset of consecutive road segments of a street which a user is authorized to monitor. The $\mathcal{T}_{\mathcal{R}}$ subtree of an access policy is an OR between many point-in-interval access

policies, one for each street the user is authorized to monitor. Fig. 4 shows a graphic example.

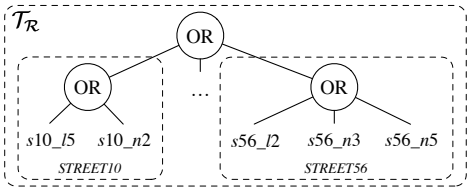



Fig. 4. Example of $\mathcal{T}_\mathcal{R}$ subtree of the key access policy with segment tree representation. The subtree $s10\_l5 \vee s10\_n2$ is a point-in-interval access policy of consecutive road segments of the street *STREET10*.

We will refer to this implementation as the *segment tree representation*.

In addition to reduce the average number of affected users, this representation reduces the key size if compared to the basic representation. Indeed, the size of an interval representation set is always less than or equal to the number of represented elements.

*3) Attribute-Pool Representation:* We introduce a third representation which extends the segment tree representation to further reduce the number of affected users due to 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 $\varepsilon$ attributes $\{i_\omega\}$, with $\omega \in [1, \varepsilon]$ (*replicas*). Each replica $i_\omega$ in the pool has the same meaning.

The $\gamma_\mathcal{R}^{(j)}$ set of each ABE-sealed key is formed by $\varepsilon$ point representation sets and contains $\mathcal{O}(\varepsilon \log \rho)$ attributes. On the other hand, the key size of a decryption key remains the same of the segment tree representation. Indeed, the $\mathcal{T}_\mathcal{R}$ subtree of a decryption key has the same structure of the segment tree representation, but each attribute included in the $\mathcal{T}_\mathcal{R}$ subtree is one among the $\varepsilon$ replicas. We will refer to this implementation as the *attribute-pool representation*.

To implement this representation, the TTP maintains a field named $num_{i_\omega}$ for each replica in $\mathcal{U}_\mathcal{R}$, which represents the number of non-revoked keys which are using that replica. The field is initialized to zero. Its value is incremented when the TTP issues a new decryption key whose access policy attribute set contains the attribute $i_\omega$. It is decreased in case of key revocation. When the TTP executes the $\mathrm{KeyGen}$ primitive, for each attribute which forms the $\mathcal{T}_\mathcal{R}$ subtree, it chooses the least frequently used replica of the attribute, i.e., $\min_\omega(num_{i_\omega})$. This representation reduces the number of affected users due to a key revocation. For example, two users who monitor exactly the same road segments may share no attributes at all if they have different replicas of each attribute in their decryption keys. In such a case, if one of the two users is revoked, the other will not be an affected user.

### B. Time Representation: Universe Subset $\mathcal{U}_\mathcal{X}$

In our system, the level of granularity of the time is one day. We define the *maximum system lifetime* as the maximum number of days of operation of the system. We represent the days from 1 to the maximum system lifetime by means of a segment tree. The $\gamma_\mathcal{X}^{(j)}$ set of each ABE-sealed key is formed by the point representation set of the data production date. On the other hand, the validity period of a decryption key is obtained through a point-in-interval access policy, which forms the $\mathcal{T}_\mathcal{X}$ subtree. Our system provides the planned expiration of the decryption keys. When a decryption key expires, it is not capable of decrypting new ABE-sealed keys anymore. From the CSS point of view, key expiration is more lightweight than key revocation because no proxy re-encryption is needed. However, an expired key can still be used to decrypt past ABE-sealed keys produced within the validity period of the key. If we want to avoid this, we need instead to revoke the key.

Note that applying attribute-pool representation to the universe subset $\mathcal{U}_\mathcal{X}$ does not improve the performance of the revocation since these attributes are never updated.

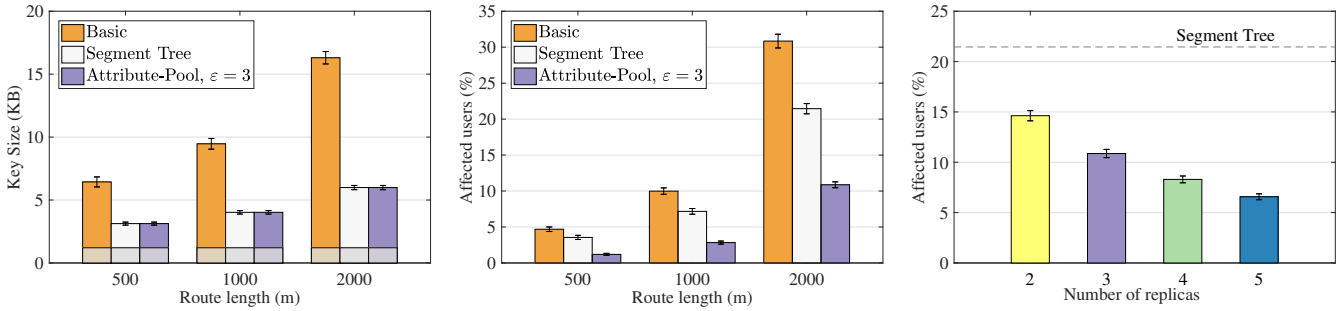
## VI. Experimental Evaluation

To test our scheme, we used a street network representing the city of Pisa[1] obtained from OpenStreetMap. We assumed a maximum system lifetime of $100$ years. Then, we considered a dataset of $300$ users with $365$-day subscriptions. In our simulation, every user subscribes to a *route* composed by consecutive road segments. A route is characterized by a length (*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 source point. Within each scenario we tested, we fixed a value for the route length which is the same for all the users. We tested the system with route lengths of $500\,\mathrm{m}$, $1000\,\mathrm{m}$, and $2000\,\mathrm{m}$. We used a KP-ABE implementation written in C [12] which realizes the four KP-ABE primitives described in Section III-A with 80-bit security.

In Fig. 5a we show a comparison between the average key sizes for the three 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. By using any representation, a user's device, e.g., a smartphone, can easily store a decryption key of a few kilobytes. The CSS 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 $1.4\,\mathrm{MB}$ using either the segment tree or the attribute-pool representation with route length of $2000\,\mathrm{m}$. The key size of the segment tree and the attribute-pool representations are about a third compared to the basic representation. Nevertheless, the key sizes are very small and all the representations can be used from the point of view of both the CSS and the user.

By using the same dataset, we revoked each user in turn and measured how many other users were affected. In Fig. 5b we show the average percentage of affected users by a single revocation. As we expected, the segment tree and the attribute-pool representations show less affected users than the basic representation. This holds for all the values of the route length tested. Fig. 5c shows the affected users with respect to the number of alternatives in the attribute-pool representation. It

[1]Lat: 43.7052–43.7264, Lon: 10.3867–10.4266.




(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. $L = 2000\,\mathrm{m}$.

Fig. 5. Performance evaluation. All the plots show 95% confidence intervals.

is evident from the figure that the affected users can be reduced even further by increasing the parameter $\varepsilon$. The proposed representations are capable to reduce the number of affected users, and this lightens the load for the CSS due to a key revocation, which is distributed over time thanks to the lazy fashion of proxy re-encryption. The drawback is that the size of the universe subset $\mathcal{U}_{\mathcal{R}}$ and the number of attributes of each encryption attribute set $\gamma$ grows with $\varepsilon$.

In our system, the load on the TTP is mainly determined by the complexity of the $\mathrm{Encrypt}$ primitive, which grows linearly with the cardinality of the encryption attribute set. The TTP executes such primitive once a day for each device during the seal procedure. We 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 sensing devices deployed randomly on the street network. We used the attribute-pool representation with $\varepsilon = 5$, which represents the worst case in terms of encryption 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 produced during the daily seal procedure. We evaluated the key sealing time on a desktop computer equipped with 16GB of RAM, an Intel® Core™ i5-6600 CPU, and running Ubuntu 16.04.3 LTS 64-bit operating system.

From our tests, the average cardinality of an encryption attribute set $\gamma$ is about 39, and $\gamma_{\mathcal{R}}$ is about 21. We observed that the key sealing time was on average 5.4 seconds, and the total size of the ABE-sealed keys was about 550 KB. Hence, the TTP is asked to carry out a task of a few seconds per day and then send a few kilobytes of data to the CSS.

## VII. Conclusions

We presented ABE-Cities, an encryption system for urban sensing in a smart city based on Attribute-Based Encryption. ABE-Cities allows for fine-grained access control over the encrypted data stored in the CSS, and it is secure against multiple adversary models. Sensing devices perform only lightweight symmetric cryptography operations. Therefore, ABE-Cities can employ also resource-constrained sensing devices, which makes it suitable for a broader set of IoT applications for smart cities. Our system also provides mechanisms to enforce planned expiration of decryption keys, as well as their unplanned revocation. We proposed methods to make the unplanned revocation efficient from the point of view of the cloud service. We showed by simulations the effectiveness of such improvements.

## References


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

[2] L. Coppolino, S. D'Antonio, G. Mazzeo, and L. Romano, "Cloud security: Emerging threats and current solutions," *Computers & Electrical Engineering*, vol. 59, pp. 126–140, 2017.

[3] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 89–98.

[4] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Infocom, 2010 proceedings IEEE*. IEEE, 2010, pp. 1–9.

[5] G. Wang, Q. Liu, and J. Wu, "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*. ACM, 2010, pp. 735–737.

[6] S. Yu, K. Ren, and W. Lou, "FDAC: Toward fine-grained distributed data access control in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 4, pp. 673–686, 2011.

[7] X. Yao, Z. Chen, and Y. Tian, "A lightweight attribute-based encryption scheme for the internet of things," *Future Generation Computer Systems*, vol. 49, pp. 104–112, 2015.

[8] V. Odelu, A. K. Das, M. K. Khan, K.-K. R. Choo, and M. Jo, "Expressive CP-ABE scheme for mobile devices in IoT satisfying constant-size keys and ciphertexts," *IEEE Access*, vol. 5, pp. 3273–3283, 2017.

[9] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2005, pp. 457–473.

[10] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf, "Segment trees," in *Computational geometry*. Springer, 2000, pp. 231–237.

[11] N. Attrapadung, G. Hanaoka, K. Ogawa, G. Ohtake, H. Watanabe, and S. Yamada, "Attribute-based encryption for range attributes," in *International Conference on Security and Cryptography for Networks*. Springer, 2016, pp. 42–61.

[12] Y. Zheng, "kpabe," https://github.com/gustybear/kpabe, 2014.