PIDGIN: Privacy-Preserving Interest and Content Sharing in Opportunistic Networks

Abstract

Opportunistic networks have recently received considerable attention from both industry and researchers. These networks can be used for many applications without the need for a dedicated IT infrastructure. In the context of opportunistic networks, the application to content sharing in particular has attracted specific attention. To support content sharing, opportunistic networks may implement a publish-subscribe system in which users may publish their own content and indicate interest in other content through subscription. Using a smartphone, any user can act as a broker by opportunistically forwarding both published content and interest within the network. Unfortunately, despite their provision of this great flexibility, opportunistic networks raise serious privacy and security issues. Untrusted brokers can not only compromise the privacy of subscribers by learning their interest but also can gain unauthorised access to the disseminated content. This paper addresses the research challenges inherent to the exchange of content and interest without: (i) compromising the privacy of subscribers and (ii) providing unauthorised access to untrusted brokers. Specifically, this paper presents an interest and content sharing solution that addresses these security challenges and preserves privacy in opportunistic networks. We demonstrated the feasibility and efficiency of this solution by implementing a prototype and analysing its performance on real smart phones.

Keywords: Secure Opportunistic Networks, Privacy-preserving Data Sharing, Sensitive Policy Enforcement, Encrypted CP-ABE Policies, Secure Haggle

1 Introduction

In the last few years, the usage of smartphones has grown dramatically and is predicted to increase even more in coming years [1]. Considering the pervasive nature of smartphones, mobile opportunistic networks could be leveraged to share information. Several of the concepts behind opportunistic networks originate from Delay Tolerant Networks (DTNs) that offer flexible content sharing without requiring a dedicated IT infrastructure [2]. Haggle [3], an example of such a network architecture, allows smartphones to opportunistically share content via short-range communication [4]. To share content, opportunistic networks such as Haggle implement a publish-subscribe system in which nodes can publish their own content and subscribe to other content by indicating their interest. Any node can also act as a broker (also called a relay) that opportunistically receives content and interest, matches them, and possibly delivers that content to other nodes.

The opportunistic networks could be applied to the exchange of information in a wide range of domains from social media to military applications. However, such networks also present serious privacy and security issues, particularly the need for an approach to the exchange of content and interest that neither (i) compromises the privacy of subscribers nor (ii) provides unauthorised access to untrusted brokers.

For the regulation of access to content, cryptographic approaches such as Attribute-Based Encryption (ABE) which include Ciphertext-Policy ABE (CP-ABE) [5] and Key-Policy ABE (KP-ABE) [6] offer fine-grained control over content but leak information about the policies and attributes that protect that content, respectively. To protect these policies, state-of-the-art solutions exist to enforce sensitive policies in outsourced environments [7–9]. However, such solutions assume that the outsourced server does not collude with any client. Thus, these solutions cannot be applied in opportunistic network settings in which nodes communicate in a peer-to-peer fashion, i.e., serving as both a client and a server.

This paper presents PIDGIN (Privacy-preserving Interest anD content sharinG In opportunistic Networks), an interest and content sharing scheme that preserves privacy. In PIDGIN,

- brokers match subscriber's interest against policies associated with content without compromising the subscriber's privacy (say, by learning attributes or interest).

- unauthorised nodes do not gain access to content, and authorised nodes gain access only if they satisfy fine-grained policies specified by the publishers.
- the system provides scalable key management in which loosely-coupled nodes communicate with each other without any prior contact.

As a proof-of-concept, we have developed and analysed the performance of a prototype running on real smartphones in order to show the feasibility of our approach.

The rest of this paper is organised into the following sections. Section 2 provides a brief overview of opportunistic networks, describes the motivating scenario, and lists some of the major research challenges for interest and content sharing in opportunistic networks with guaranteed preservation of privacy. In Section 3, we draw the system model. Next, we describe the proposed scheme in Section 4. Section 5 elaborates PIDGIN's details. In Section 6, we provide the concrete construction. Section 7 analyses PIDGIN from a security perspective. In Section 8, we report the outcomes of the performance analysis. Section 9 is dedicated for discussion. Section 10 reviews the related work. Finally, we conclude in Section 11 and highlight some directions for future work.

2 Opportunistic Networks

In this section, we provide a brief overview of opportunistic networks, a motivating scenario, and the major research challenges in opportunistic networks that we address.

2.1 Overview

Conceptually, opportunistic networks originate from DTNs that enable content exchange between nodes in a publish-subscribe fashion, generally via short-range communication. In a typical opportunistic network, such as Haggle, a subscriber node subscribes interest while a publisher node publishes content to its neighbouring nodes [4]. These neighbouring nodes are intermediate nodes, known as brokers, that epidemically disseminate interest and content within the network. A resolution takes place when a broker node finds a match between the interest of a subscriber and the tags associated with published content. As a result of resolution, a broker forwards content to the subscriber.

2.2 Motivating Scenario

Curiosity: A Military Mission: Let us consider a battlefield scenario for a mission called *Curiosity* in which soldiers are equipped with smartphones. During the mission, a scout collects some sensitive information about the enemy (for instance, an image of the enemy's position) using her smartphone camera. After acquiring this sensitive information, a scout desires to share it with other soldiers. For this reason, she may tag the image with the mission name, i.e., *Curiosity*. Unfortunately, there is no Internet connectivity on the battlefield and the only way to share is to use the short-range communication offered by smartphones. Therefore, the scout would like to share the image with other soldiers using their smartphones. We assume that the soldiers are interested in getting information about the mission and subscribe using their smartphones.

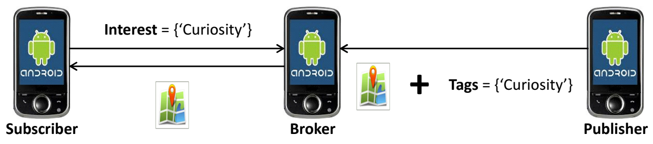


Figure 1. An example of content sharing in an opportunistic network.

Haggle: A Possible Solution: To exchange information in such scenarios, we can leverage opportunistic networks, such as Haggle. Using Haggle, the scout publishes the image with *Curiosity* as a tag. Any solider can show interest in *Curiosity* by subscribing, as illustrated in Figure 1. Here, we assume that someone as a broker receives both interest and image along with the tag. Whenever that happens, the broker checks whether the interest of a subscriber matches any tag associated with the image. If so, the broker forwards the image to the subscriber(s).

Privacy and Confidentiality Issues: First of all, to preserve confidentiality, the information about the *Curiosity* mission should be shared only within a particular group of soldiers. Each content item is associated with an access policy that indicates who should have access to it. For example, information about the *Curiosity* mission might have a policy (P) that content is shared with either a *Major* or a *Soldier* from the *Infantry* unit. Even if the content (i.e., image) is encrypted, the policy itself could reveal sensitive information. That is, an enemy may infer useful information from the fact that some contents are sent to a *Major* or a *Soldier* from the *Infantry* unit. Outsiders (i.e., enemies) and insiders (i.e., soldiers) serving as brokers may gain unauthorised access to contents. Furthermore, the interest of subscribers and the tags associated with content may also reveal sensitive information. Therefore, in addition to the content itself, its associated tags, policies, and subscription information (i.e., interests) should also be protected.

This scenario motivates the need to tackle the security and privacy issues that we generally face in opportunistic networks. In the following section, we list some major research challenges inherent to these issues that we address in this paper.

2.3 Research Challenges

To guarantee the preservation of privacy for interest and content sharing in opportunistic networks, the following major research challenges related to both (i) privacy and confidentiality (i.e., *C1-C3*) and (ii) functionality (i.e., *C4-C5*) need to be addressed:

C1 In the presence of unauthorised brokers, how do we regulate access to disseminated content and preserve confidentiality?

C2 In the presence of curious brokers, how does the network exchange content without compromising the privacy of its subscribers?

C3 How can a subscriber subscribe to content without exposing her interest to untrusted brokers?

C4 In order to minimise the flood of unnecessary traffic on the communication network, how do we ensure that a subscriber receives content if and only if authorised to decrypt?

C5 Assuming the loosely-coupled nature of the publish-subscribe model, how do we address the challenges above (i.e., *C1-C4*) without sharing any keys between the subscriber, publisher and broker nodes?

3 System Model

Before presenting our threat model and assumptions, we identify the entities involved in the system:

A Publisher is a node that can publish the content.

A Subscriber is a node that can subscribe interest.

A Broker is a node that may receive and disseminate both content and interest. It evaluates whether any content matches known interest. On successful evaluation, it forwards content to the subscribers.

Trusted Key Management Authority (TKMA) is an offline trusted entity that distributes keying material (including private keys and/or public parameters) to all nodes out of the band (usually once in the lifetime of a node, typically when the node is initialised).

Threat Model. We assume that brokers are *honest-but-curious*, i.e., they honestly follow the protocol, but remain curious to learn about content and interest. Also, we assume that brokers may collude. Furthermore, we consider that the TKMA is fully trusted and plays a role at the time of system initialisation. Last but not least, we assume only passive adversaries and do not consider active adversaries that can manipulate the exchanged information.

4 Approach

In this section, we describe the proposed scheme for preserving privacy during interest and content sharing in opportunistic networks. As a starting point, we consider some basic schemes that partially address research challenges listed in Section 2.3. Next, we gradually address all research challenges and finally describe the proposed scheme.

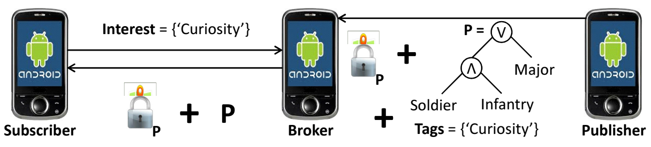


Figure 2. Regulation of access on contents using CP-ABE policies.

4.1 Scheme I: Regulate Access on Content

To preserve the confidentiality of content, a publisher might specify who can gain access. A possible approach for the publisher could be to regulate access on content by employing ABE, such as CP-ABE [5] or KP-ABE [6]. ABE offers fine-grained policies for content access. In this scheme, we consider CP-ABE because it enables a publisher to exert control over access to content, as described in the use case scenario. In contrast, in KP-ABE, a key generation authority exerts control over who can access content. Figure 2 illustrates this scheme in which the image is encrypted according to the policy: *either a Major or a Solider from the Infantry unit can get access*. The policy is expressed as a tree whose leaf nodes represent the attributes; non-leaf nodes denote the AND, OR and threshold gates. In this scheme, a broker forwards content to the subscribers if a subscriber's interest matches with any tag associated with the content.

This approach preserves the confidentiality of disseminated contents without providing access to unauthorised brokers. This scheme, however, has a drawback. A broker might send content to subscribers who might not be able to decrypt it. In fact, a broker's role is merely to match the interest of subscribers against tags associated with content without checking whether a subscriber has access authorisation. For instance, consider a subscriber who is a soldier but neither a *Major* nor a member of the *Infantry* unit.

In summary, this scheme resolves the access control problem *(C1)* while raising the problem of a communication network flooded with unnecessary traffic *(C4)*.

4.2 Scheme II: Perform an Authorisation Check

This scheme extends Scheme I and resolves the flooding problem *C4*. In this scheme, a subscriber may send attributes and interest to brokers so that a broker can perform an authorisation check prior to forwarding the contents. To perform the authorisation check, a broker matches leaf nodes in the policy tree with the subscriber's attributes. If there is a match, a leaf node will be marked as satisfied. After evaluating leaf nodes, a broker evaluates intermediate nodes (including AND, OR and threshold) in the policy. A broker will forward encrypted content to subscriber if and only if (i) the root node of the policy is marked as satisfied and (ii) the interest matches to the tags.

This scheme targets both the access control problem *(C1)* and the flooding problem *(C4)*. However, it still raises some privacy issues. First, both the cleartext attributes of subscribers and the cleartext CP-ABE policies can compromise the privacy of subscribers, i.e, *C2*. For example, the enemy may learn from policies that there is some information intended for a *Major*. Second, the cleartext interest of a subscriber may also leak information, i.e., *C3*. For instance, the enemy may learn that this content or interest concerns the *Curiosity* mission.

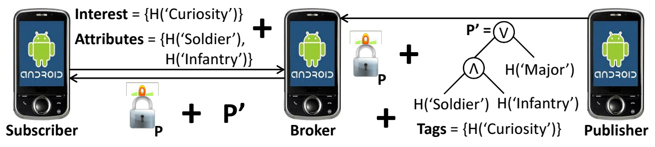


Figure 3. Private information is hidden through replacement of leaf nodes in the CP-ABE policy, tags, attributes and interest items with their corresponding hashes.

4.3 Scheme III: Hide Private Information Using a Hash

In order to partially overcome the issue of subscriber privacy *(C2)*, a subscriber and a publisher may hash both attributes and leaf nodes in the policy tree, respectively. Similarly, a subscriber's interest could be protected by calculating the hash values of interest items and tags associated with contents. In this scheme, a broker forwards encrypted content to subscribers if and only if (i) the hash value of the interest matches the hash value of the tag (i.e., h('Curiosity')) and (ii) hash values of attributes (i.e., {h('Soldier'), h('Infantry')}) satisfy the policy P' whose leaf nodes are also hashed, as shown in Figure 3.

Unfortunately, this scheme is vulnerable to a pre-computed dictionary attack. That is, the enemy may pre-calculate a list of hashes for possible attributes (and leaf nodes in the policy tree) and a list of hashes for potential interest items (and tags). The pre-calculated list of hashes may easily reveal the original attributes (and leaf nodes in the policy tree) and interest (and tags).

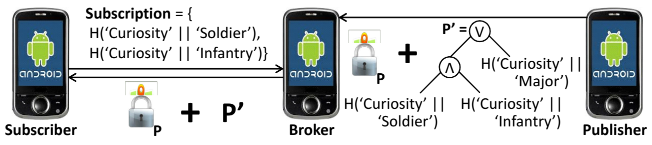

Subscription = {
H('Curiosity' || 'Soldier'),
H('Curiosity' || 'Infantry')}
P' =
∨
∧
H('Curiosity' || 'Major')
H('Curiosity' || 'Soldier')
H('Curiosity' || 'Infantry')
+
P
+
P'
Subscriber
Broker
Publisher

Figure 4. Hardening against a pre-computed dictionary attack through concatenation a pair of (i) a leaf node in the CP-ABE policy and a tag (ii) an attribute and an interest item, then calculation of the hash on the final string.

4.4 Scheme IV: Hardening Against a Pre-Computed Dictionary Attack

To harden against the pre-computed dictionary attack, a publisher may replace each leaf node in the policy with a hash of a concatenated pair of a tag and an attribute. Similarly, a subscriber may subscribe using the hash of a concatenated pair of an interest item and an attribute (i.e., {H('Curiosity' || 'Soldier'), H('Curiosity' || 'Infantry')}) as illustrated in Figure 4. In this scheme, a broker just needs to check whether the items in a subscription satisfy the hashed policy P'. Upon successful evaluation, the broker will forward the content to subscribers. The advantage of this scheme is that it not only hardens against pre-computed dictionary attacks but also decreases the number of comparisons performed at the broker's end as compared to Scheme III. This is because a broker performs integrated checks that cover both authorisation and interest matching simultaneously in contrast to Scheme III in which a broker performs two different checks: one to check the authorisation and one to match the interest. Though it enlarges the key space (which could be computationally extensive), this scheme is still vulnerable to a pre-computed dictionary attack.

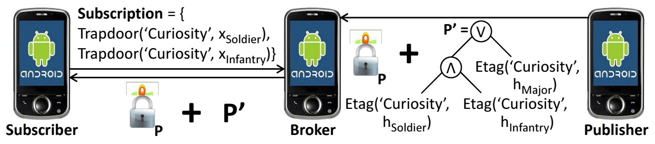

Subscription = {
Trapdoor('Curiosity', x_Soldier),
Trapdoor('Curiosity', x_Infantry)}
Subscriber
P
+ P'
Broker
P
+
P' = ∨
∧
Etag('Curiosity', h_Major)
Etag('Curiosity', h_Soldier)
Etag('Curiosity', h_Infantry)
Publisher

Figure 5. The PIDGIN scheme protects the content, the policy, the tags associated with content, and the subscriber's interest and attributes.

4.5 PIDGIN: The Proposed Scheme

Our proposed scheme, PIDGIN, aims at addressing all research challenges (i.e., *C1-C5*) listed in Section 2.3. The main idea behind PIDGIN is regulation of access to content using CP-ABE and extension of cleartext CP-ABE policies with the Public-key Encryption with Keyword Search (PEKS) scheme [10] to protect attributes, interest, tags and leaf nodes in the policy tree. The PEKS scheme consists of four basic functions including **Keygen**, **Etag**[1], **Trapdoor** and **Test**. For each attribute, we run **Keygen** to calculate a key pair consisting of both public (i.e., $h_{Soldier}$) and private (i.e., $x_{Soldier}$) keys corresponding to a given attribute (i.e., *Soldier*). To protect policies and tags, a publisher can replace each leaf node in the policy tree with the **Etag** function of the PEKS scheme, which takes as input a tag (i.e., *Curiosity*) and the public key of the attribute as shown in Figure 5. A subscriber protects attributes and interest by replacing each interest item in the subscription list with a **Trapdoor** function which takes as input an interest item (i.e., *Curiosity*) and the private key (i.e., generated by the PEKS scheme) corresponding to the attribute.

A broker performs encrypted matching between encrypted policies and encrypted subscriptions. It runs the **Test** function, a building block that matches a trapdoor to an encrypted tag. If an encrypted tag in the policy tree P' matches with any encrypted trapdoor in the subscription list, the tree node is marked as satisfied. The broker evaluates all nodes in the policy tree starting from leaf nodes to root. If the root is satisfied, the broker will forward content along with the encrypted policy to the subscribers.

5 Details of PIDGIN

5.1 Initialisation and Key Generation Phases

During the initialisation phase, the system is set up to initialise both CP-ABE and PEKS schemes. In PIDGIN, the TKMA generates and distributes keys during the key generation phase. The TKMA generates a private set of attributes (i.e., CP-ABE private key) and sends it securely to the subscriber out of the band. The TKMA publishes the public part of attributes (i.e., CP-ABE public key) to all publishers. Since the attributes are protected using the PEKS scheme, the TKMA also generates a pair of keys corresponding to each attribute. Similar to the CP-ABE key distribution, the TKMA sends the private and public parts of the PEKS key pair to the subscriber and publishers, respectively. The major difference between the CP-ABE private key set and the PEKS private key set is that the former is unique for each user, while the latter is not.

5.2 Publisher's Encryption Phase

To protect the content and preserve the privacy of subscribers, a publisher encrypts content with CP-ABE policies and protects those policies as well. The contents could be encrypted with a symmetric key, such as Advanced Encryption Standard (AES), which is further encrypted with the CP-ABE policy. Since the CP-ABE policy may compromise the privacy of subscribers, the CP-ABE policies are encrypted using PEKS. While encrypting CP-ABE policies using PEKS, PIDGIN also incorporates tags that are associated with content.

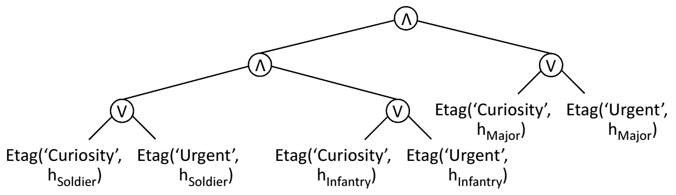

∧
∧
∨
∨
∨
Etag('Curiosity', h_Major)
Etag('Urgent', h_Major)
Etag('Curiosity', h_Soldier)
Etag('Urgent', h_Soldier)
Etag('Curiosity', h_Infantry)
Etag('Urgent', h_Infantry)

Figure 6. The extended CP-ABE policy with two tags, i.e., 'Curiosity' and 'Urgent'.

To extend CP-ABE policies for PEKS, a publisher considers each leaf node in the policy tree as well as number of tags that are associated with contents. If there is only a single tag then a publisher replaces the leaf node with the **Etag** function as already illustrated in Figure 5. The **Etag** function takes a tag keyword to be encrypted and the public key corresponding to the leaf node under consideration. After running the **Etag** function, a publisher gets an encrypted tag. The **Etag** function does not leak information about the tags or leaf nodes in the policy tree. In the case that there is more than one tag then a publisher runs the **Etag** function for each tag item and encrypts it with the public key corresponding to the leaf node under consideration. Finally, the leaf node attribute is replaced with the subtree where all newly generated Etags corresponding to tags are disjuncted

[1]The Etag function is called PEKS in [10].

using OR. Figure 6 illustrates an example of the policy involving two tags, i.e., 'Curiosity' and 'Urgent'.

5.3 Subscriber's Encryption Phase

In order to protect the interest of a subscriber and its attributes, a subscriber encrypts each interest item using the private key (i.e., generated by the PEKS scheme) corresponding to the attribute. PIDGIN considers that a subscriber might have multiple attributes and interest items. Generally, each interest item is encrypted with each private key (i.e., generated by the PEKS scheme) that corresponds to the attribute. Figure 5 describes the case in which a subscriber holds two attributes and subscribes with a single interest item. Let us assume that a subscriber has two interest items, say 'Curiosity' and 'Urgent', while holding attributes Solider and Infantry. The subscription list would contain four items including **Trapdoor**('Curiosity', $x_{Soldier}$), **Trapdoor**('Curiosity', $x_{Infantry}$), **Trapdoor**('Urgent', $x_{Soldier}$) and **Trapdoor**('Urgent', $x_{Infantry}$). The trapdoor representation does not leak information about the interest item and the attribute.

5.4 Broker's Matching Phase

A broker opportunistically exchanges both content and subscriptions. Once a broker receives both the encrypted subscription and the encrypted content along with the encrypted policies, it evaluates whether the encrypted subscription satisfies the encrypted policy. For this evaluation, the broker runs a matching function that recursively evaluates the encrypted policy tree. The **Test** function matches each encrypted leaf node in the policy against the encrypted interest item in the subscription.

The **Test** function returns either *TRUE* or *FALSE*, indicating whether the encrypted tag is matched with the trapdoor or not, respectively. By running the **Test** function, a broker does not learn about the tag or the interest item because both are encrypted and they are matched in an encrypted manner. If an encrypted tag in the policy tree matches with any trapdoor in the subscription list, that node is marked as satisfied. After evaluating leaf nodes, a broker can evaluate intermediate AND, OR and threshold nodes in the policy tree to finally identify whether the root node of the policy tree is satisfied or not. If the root node is satisfied, the broker will forward content along with the encrypted policy to the subscriber.

5.5 Subscriber's Decryption Phase

Once a subscriber receives the encrypted content along with the encrypted policy, it first recovers the original CP-ABE policy. For this recovery, either leaf node (if a single tag, see Figure 2) or a subtree of tags (if more than one tag, see Figure 6) is replaced with their corresponding attribute. Before sharing the encrypted interest, a subscriber builds the subscription history as a lookup table containing an attribute and its corresponding trapdoor. If the trapdoor is matched with any encrypted tag in the leaf node of the policy, the subscription history will be looked up to find the attribute corresponding to the matched trapdoor. Next, a leaf node (if a single tag) or a subtree of tags (if more than one tag) will be replaced with the found attribute. If no match is found, then a dummy attribute will be placed. This recovers the original CP-ABE policy (i.e., one shown in Figure 2) that can finally be used by the CP-ABE decryption function to get the symmetric key that is required for decryption of the contents.

6 Concrete Construction

In this section, we provide some definitions and details of core functions used in different phases of the PIDGIN lifecycle.

6.1 Definitions

Policy Structure P**.** We assume a policy tree P that represents an access structure. Each non-leaf node represents an AND, an OR or a threshold gate. Let us consider that num_x denotes number of children of a node x and k_x represents the threshold value. For OR and AND gates, k_x is 1 and num_x, respectively. For the threshold gate, the value of k_x is: $0 < k_x \leq num_x$. Let us consider that **parent**(x) represents the parent of a node x, **att**(x) denotes the attributes associated with leaf node x, and **index**(x) returns the number associated with a node x, with nodes numbered from 1 to num.

Bilinear Maps. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}_1$ and $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a bilinear map. The bilinear map e satisfies the following properties:

- Computability: given $g, h \in \mathbb{G}_1$, there is a polynomial time algorithm to compute $e(g, h) \in \mathbb{G}_2$.
- Bilinearity: $\forall u, v \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
- Non-degeneracy: if g is a generator of $\mathbb{G}_1$ then $e(g, g)$ is a generator of $\mathbb{G}_2$, where $e(g, g) \neq 1$.

Notice that the bilinear map e is symmetric since $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$.

Hash Functions. We consider the hash functions

$H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^{\log p}$.

Lagrange Coefficient. We define the Lagrange coefficient $\Delta_{i,A}$ for $i \in \mathbb{Z}_p$ and a set A of elements in $\mathbb{Z}_p$: $\Delta_{i,A}(x) = \prod_{j \in A, j \neq i} \frac{x-j}{i-j}$.

6.2 Construction Details

Init(1^K). The init algorithm takes as input the security parameter k that determines the size of p. It randomly picks two exponents $\alpha, \beta \in \mathbb{Z}_p$ and outputs the public key $PK = (\mathbb{G}_1, g, h = g^\beta, e(g,g)^\alpha)$ and the master key $MK = (\beta, g^\alpha)$. The public key PK is published while the master key MK is kept securely by the TKMA. Moreover, two stores, the Search Key Secret Store ($SKSS$) and the Search Key Public Store ($SKPS$), which are managed by the TKMA, are initialised as:

$$SKSS \leftarrow \phi$$
$$SKPS \leftarrow \phi$$

KeyGen(MK, A). The key generation algorithm is run by the TKMA. It takes as input a list of attributes A and outputs a CP-ABE decryption key and a set of search key pairs. To generate the decryption key, it first chooses a random $r \in \mathbb{Z}_p$ and then a random $r_j \in \mathbb{Z}_p$ for each attribute $j \in A$. Next, it computes the decryption key as:

$$DK = (D = g^{(\alpha+r)/\beta}, \\ \forall \in A : D_j = g^r \cdot H_1(j)^{r_j}, D'_j = g^{r_j})$$

Before the generation of a search key pair for an attribute $j \in A$, a search key store (either $SKSS$ or $SKPS$) can be looked up. If the search key pair already exists, then the public and private keys will be collected from $SKPS$ or $SKSS$, respectively. Otherwise, the algorithm chooses a random $x_j \in \mathbb{Z}_p^*$, calculates $h_j = g^{x_j}$, and updates both private and public key stores as:

$$SKSS \leftarrow SKSS \cup (j, x_j)$$
$$SKPS \leftarrow SKPS \cup (j, h_j)$$

Next, it computes the search key secret as: $SKS = (\forall \in A : x_j)$. Finally, the $SKPS$ is publicised while the decryption key DK and the search key secret SKS are securely transmitted to the subscriber.

Etag(PK, h_i, t). The **Etag** algorithm encrypts a given tag t with h_i. It chooses a random $r \in \mathbb{Z}_p^*$ and computes $z = e(H_1(t), h^r)$. Next, it computes $A = g^r$ and $B = H_2(z)$ and outputs the encrypted tag as: $ET = (A, B)$.

Pub-Enc($PK, SKPS, C, P, T$). The publisher encryption algorithm encrypts content C under the access policy P with a list of tags T. It also encrypts P. In reality, it randomly generates a symmetric key K and encrypts C as $\{C\}_K$ and then encrypts K under P. To encrypt K under P, it chooses a polynomial q_x for each node x in a top-down manner, starting from the root R, such that it sets degree d_x one less than the threshold value k_x, i.e., $d_x = k_x - 1$. Starting from the root R, it chooses a random $s \in \mathbb{Z}_p$, sets $q_R(0) = s$ and chooses other d_R points randomly. For any other non-root node x, it sets $q_R(0) = q_{parent(x)}(index(x))$ and chooses other d_x points randomly. Let Y be the set of leaf nodes in P. The ciphertext is computed as:

$$CT = (\tilde{E} = Ke(g,g)^{\alpha s}, E = h^s, \\ \forall y \in Y : E_y = g^{q_y(0)}, E'_y = H_1(att(y))^{q_y(0)})$$

Next, the policy P is encrypted as follows. For each leaf node i, it looks up the corresponding private secret key h_i from the $SKPS$. Then, it runs **Etag**(h_i, t) for each tag $t \in T$ and combines all encrypted tags corresponding to an attribute to form an OR subtree. The original leaf node attribute is replaced with this OR subtree. If only one tag exists in T, the original attribute is replaced with the output of the **Etag** function. This basically generates the encrypted policy P'. Finally, this algorithm returns $PE = (P', CT, \{C\}_K)$.

Trapdoor(x_i, t). The **Trapdoor** algorithm encrypts interest item t using x_i. It returns the encrypted interest item $TD = H_1(t)^{x_i}$.

Sub-Enc(I, SKS). The subscriber encryption algorithm encrypts interest I using the attributes SKS. For each interest item $t \in I$, it runs **Trapdoor**(x_i, t) using search key secret x_i corresponding to each attribute $i \in SKS$. A subscriber also maintains a history of subscription HS to keep track of all trapdoors belonging to a subscription. HS is initialised as $HS \leftarrow \phi$ and updated as:

$$\forall i \in SKS : HS \leftarrow HS \cup (i, TD_i)$$

HS maintains each trapdoor with its corresponding attribute. Finally, this algorithm publicises $SE = (TD_1, TD_2, \ldots, TD_{|I|.|SKS|})$ and keeps HS securely.

Test(ET, TD). The **Test** algorithm takes the encrypted tag and trapdoor and returns *TRUE* if $H_2(e(TD, A) \stackrel{?}{=} B$ is *TRUE* and *FALSE* otherwise.

Bro-Match(P', SE). This algorithm takes the publisher encrypted policy P' and the subscriber encrypted interest SE and returns *TRUE* if they match and *FALSE* otherwise. To perform the match, a broker runs **Test**(ET_i, TD_j) for each leaf node i in P' and trapdoor $TD_j \in SE$. If an encrypted leaf node matches with any trapdoor, it is marked as satisfied (i.e., *TRUE*). After evaluating leaf nodes, the algorithm evaluates intermediate nodes (AND, OR and threshold). After this evaluation, if the root node of the encrypted policy P' is satisfied, that is, *TRUE*, then this algorithm returns *TRUE* and *FALSE* otherwise.

Sub-Dec(PE, HS, DK) This algorithm decrypts the policy P' and then decrypts the encrypted contents PE. First, it matches encrypted leaf nodes with a trapdoor in HS by running **Test**. If a match is found, the corresponding attribute is selected from HS. The leaf node (if a single tag) or a subtree of encrypted tags conjuncted with OR (if more tags) will be replaced with the selected attribute. If no match is found, then a dummy attribute will be placed. This recovers the original policy, which will be used to decrypt the symmetric key: if node x is a leaf node then we assume $i = att(x)$ and run the following function if $i \in A$:

$$\begin{aligned} DecryptNode(CT, DK, x) &= \frac{e(D_i, E_x)}{e(D'_i, E'_x)} \\ &= \frac{e(g^r.H(i)^{r_i}, g^{q_x(0)})}{e(g^{r_i}, H(i)^{q_x(0)})} \\ &= e(g,g)^{rq_x(0)} \end{aligned}$$

If $i \notin A$ then $DecryptNode(CT, DK, x) = \perp$. For a non-leaf node x, the algorithm runs $DecryptNode(CT, DK, z)$ for each child z of x and stores output as F_z. Let A_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$. If no such set exists then the node was not satisfied and the function returns $\perp$. Otherwise, it computes:

$$F_x = \prod_{z \in A_x} F_z^{\Delta_{i,A'_x}(0)}$$

(where $i = index(z)$ and $A'_x = index(z) : z \in A_x$)

$$= \prod_{z \in A_x} (e(g,g)^{r.q_z(0)})^{\Delta_{i,S'_x}(0)}$$

$$= \prod_{z \in A_x} (e(g,g)^{r.q_{parent(z)}(index(z))})^{\Delta_{i,A'_x}(0)}$$

(by construction)

$$= \prod_{z \in A_x} (e(g,g)^{r.q_x(0)})^{\Delta_{i,A'_x}(0)}$$

$$= (e(g,g)^{r.q_x(0)}$$

(using polynomial interpolation)

If the tree is satisfied by A, we set

$$\begin{aligned} G &= DecryptNode(CT, DK, R) \\ &= e(g,g)^{rq_R(0)} \\ &= e(g,g)^{rs} \end{aligned}$$

The symmetric key is decrypted by computing: $\tilde{E}/(e(E,D)/G) = \tilde{E}/(e(h^s, g^{(\alpha+r)/\beta})/e(g,g)^{rs}) = K$.

Finally, K is used to decrypt $\{C\}_K$ in order to access contents C.

7 Security Analysis

In PIDGIN, the contents are encrypted using a symmetric key, which is encrypted with the CP-ABE policy. The leaf nodes in the policy tree are further encrypted using **Etag** as proposed in PEKS by Boneh *et al.* [10]. The PEKS is semantically secure against a chosen keyword attack in the random oracle model, assuming that the Bilinear Diffie-Hellman Problem (BDH) is hard (for proof, see Theorem 3.1 in [10]). However, the CP-ABE policy structure is not protected and leaks information about number of attributes or tags used. This leak could partially be tackled by inclusion of some dummy attributes at the cost of an increase in complexity. In PIDGIN, brokers may collude but they cannot gain access to contents, policies or subscriptions. If a broker colludes with a subscriber, they together learn no more information than is already available to the subscriber alone. In the case that two subscribers collude to receive content that each of them alone cannot get otherwise, our scheme prevents such collusion attacks because each subscriber's (CP-ABE) decryption key includes a randomness value that will prevent access to the content.

8 Performance Evaluation

As a proof-of-concept, we have developed a prototype of PIDGIN. The prototype is based on an extension of the open source libfenc[2] library written in the C language, a library of functional encryption that includes CP-ABE. Since we proposed to extend CP-ABE with PEKS, we have implemented PEKS in C using the PBC[3] library, which is an underlying library also required by the libfenc library. After extending the CP-ABE with PEKS (on the x86 architecture),

[2]`https://code.google.com/p/libfenc/`
[3]`http://crypto.stanford.edu/pbc/`

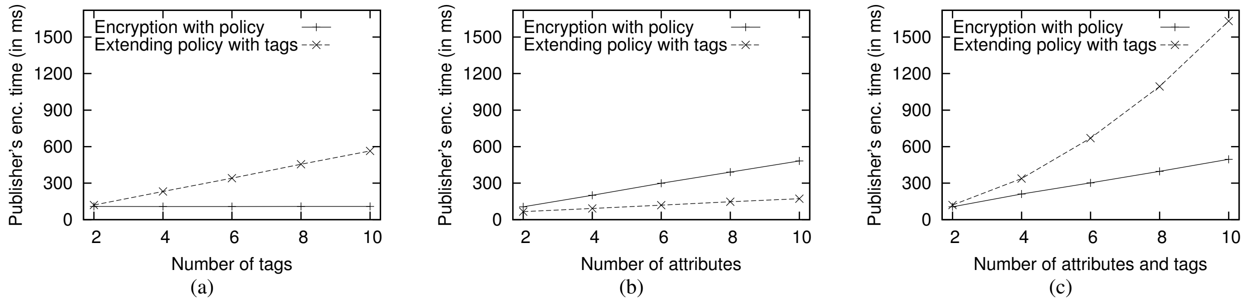


Figure 9. Effect of (a) tags, (b) attributes and (c) both tags and attributes on publisher's encryption time.

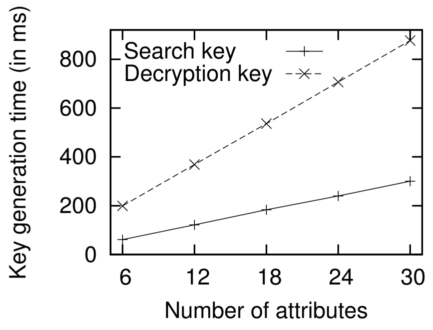


Figure 7. Effect of attributes on the key generation time.

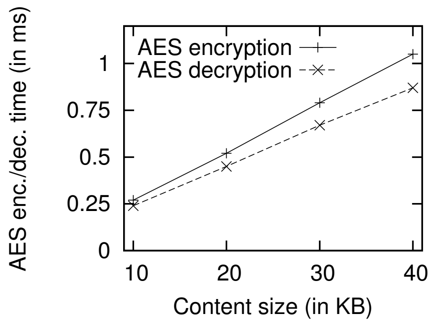


Figure 8. Effect of content size on AES encryption/decryption time.

we cross-compiled it for the ARM architecture to test our prototype on a Samsung Galaxy SIII smartphone (Android version 4.1.2, kernel version 3.0.31, 1 GB RAM, and the ARMv7 processor). For the deployment of this prototype, we cross-compiled both GMP[4] (the GNU Multiple Precision arithmetic library required by PBC) and PBC libraries for the ARM architecture and installed both on the smartphone. The presented results are averaged over 20 runs.

8.1 Initialisation and Key Generation Phases

During the initialisation phase, the system-level keying material is generated. During the key generation phase, both search and decryption keys are generated for a given set of attributes. Both phases could be run on a PC because keys are distributed out of the band. However, we consider running both phases on a smartphone (with specifications already described above). The initialisation phase takes 108.5 milliseconds (ms). The generation time of search keys grows linearly with increase in number of attributes as illustrated in Figure 7, where 30 search keys take 300 ms (i.e., an average of 10 ms per attribute). Similarly, the key generation time of decryption keys also grows linearly with increase in number of attributes, where 30 decryption keys take approximately 877 ms (i.e., an average of 29.25 ms per attribute). Asymptotically, the complexity of the key generation is $\Theta(|A|)$, where $|A|$ indicates number of attributes in list A.

8.2 Publisher's Encryption Phase

In this phase, a publisher encrypts content with a randomly generated symmetric key. In our prototype we use AES keys. The symmetric key is encrypted with the CP-ABE policy. The the CP-ABE policy is extended with tags

[4]`http://gmplib.org/`

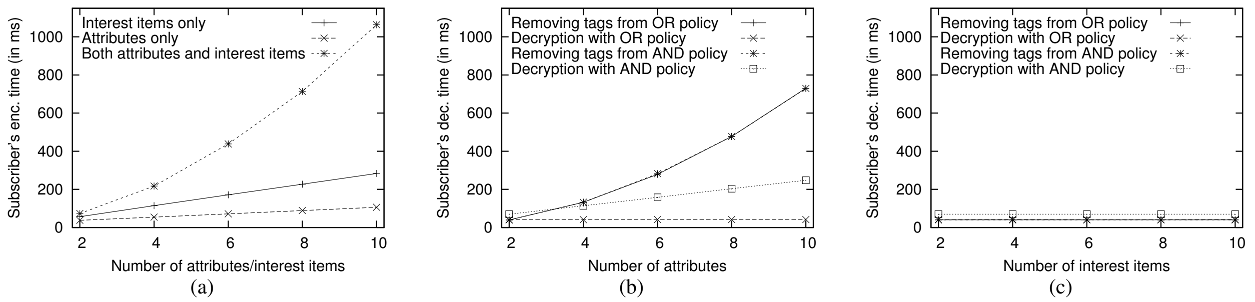


Figure 10. Effect of (a) attributes/interest items on subscriber's encryption time and effect of (b) attributes and (c) tags on subscriber's decryption time.

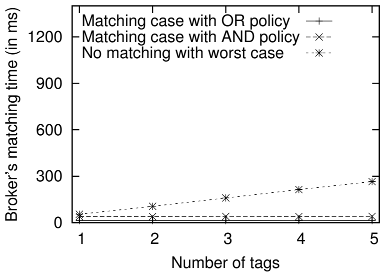
that are also encrypted. Figure 8 shows the symmetric encryption time, which grows linearly with the increase in content size (C). Encryption of a piece of content of size 40 Kilobyte (KB) takes 0.105 ms (i.e., an average of 0.026 ms per KB). To measure the performance overhead for the encryption time, we varied the numbers of tags and/or attributes (A_P^*), as shown in Figure 9. In Figure 9(a) and Figure 9(b), we observe the effect of tags and attributes on publisher's encryption time, respectively. In Figure 9(a), we observe effect of tags (ranging from 2 to 10) while keeping the number of attributes constant (i.e., 2 attributes - the minimum attributes required to make AND/OR policy). As we can expect, the time to extend a policy with tags grows linearly with increase in number of tags. In Figure 9(b), we observe the effect of attributes (ranging from 2 to 10) in a policy while considering a single tag. The time for encryption of the symmetric key with the policy grows linearly with increase in number of attributes. Since the number of attributes increases, it also linearly increases the time to extend the policy with tags. In Figure 9(c), we show the most complex case in which we increase both attributes and tags simultaneously. The growth of the time needed to extend a policy with tags is quadratic, depending on the number of attributes and the number of tags. In our experimentation, we considered the number of tags as equal to the number of attributes. In a policy with 2 attributes each with 2 tags, it takes approximately 120 ms to extend the policy tags, while in a policy with 10 attributes with 10 tags each, it takes approximately 1632 ms. Generally, the asymptotic complexity of publisher's encryption is $\Theta(|A_P^*| \cdot |T| + |C|)$.

8.3 Subscriber's Encryption Phase

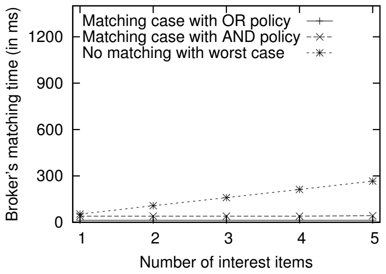
Figure 10 shows the performance overhead incurred during the encryption (see Figure 10(a)) and decryption phases (see Figure 10(b) and Figure 10(c)). In the subscriber's encryption phase, a subscriber encrypts the subscription, which is based on the number of interest items (I) and attributes (A_S^*). In our experimentations, we observed the effect of how different values for the number of attributes and interest separately and together affect the subscription's encryption time. To observe the effect of the number of attributes, we increased the attributes from 2 to 10 while keeping interest items constant (i.e., 1 interest item). Generation of trapdoors for 10 attributes with a single interest item each took approximately 106 ms. Second, we observed the effect of number of interest items on the subscription's time by increasing interest items from 2 to 10 while keeping attributes constant (i.e., 2 attributes conjuncted with either AND or OR). The subscriber took approximately 284 ms to encrypt an interest containing 10 items. As illustrated in Figure 10(a), attributes alone or interest items alone linearly affect the subscriber's encryption time. However, we also consider the case when we see effects of both attributes and interest items together. For this purpose, we assumed that number of attributes is equal to that of interest items; that is, if there are two attributes, it means there are two interest items per attribute. Similarly, we assumed 10 attributes with 10 interest items each, which took 1063 ms. The combined effect of attributes and interest items indicates that its growth has quadratic effect on the subscriber's encryption time as shown in Figure 10(a). The asymptotic complexity of the subscriber's encryption is: $\Theta(|A_S^*| \cdot |I|)$.

8.4 Broker's Matching Phase

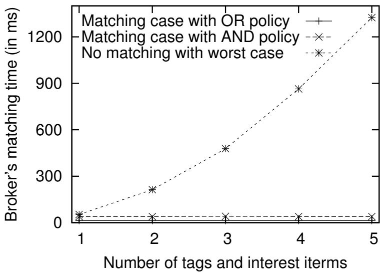
This is the key phase in the lifecycle of PIDGIN. During this phase, a broker matches the encrypted subscription against the encrypted policy associated with the encrypted content. In our analysis, we observe the effect of the numbers of tags and interest items separately and together while keeping the number of attributes constant (i.e., 2 attributes, necessary to have OR or AND policy). Furthermore, we consider the matching case with both OR and AND policies, as well as a zero match case which is the worst case situation. Figure 11 shows the performance analysis of this



(a)



(b)



(c)

Figure 11. Effect of (a) tags, (b) interest items and (c) both tags and interest items on broker's encrypted matching time.

Table 1. Description of frequently used symbols.

Symbol	Description
A	A list of attributes
A_P^*	A list of attributes used to encrypt content
A_S^*	A list of attributes used to encrypt interest
C	Content
I	A list of keywords a subscriber is interested in
T	A list of search tags associated with content

Table 2. Summary of complexity of each phase in the lifecycle of PIDGIN.

Phase Name	Best Case	Worst Case
Key generation	$\Theta(\|A\|)$	
Publisher encryption	$\Theta(\|A_P^*\| \cdot \|T\| + \|C\|)$	
Subscriber encryption	$\Theta(\|A_S^*\| \cdot \|I\|)$	
Broker matching	$\Omega(1)$	$O(\|A_P^*\| \cdot \|T\| \cdot \|A_S^*\| \cdot \|I\|)$
Subscriber decryption	$\Omega(\|C\|)$	$O(\|A_P^*\| \cdot \|T\| \cdot \|A_S^*\| \cdot \|I\| + \|C\|)$

phase. In Figure 11(a), we observe the effect of number of tags on the matching time while keeping the number of interest items as constant, i.e., 1. As the graph shows, the matching time increases linearly with the increase in number of tags. Similarly, we measure the effect of the number of interest items on the matching time while keeping the number of tags constant, i.e., 1. As Figure 11(b) indicates, the matching time grows linearly with the increase in the number of interest items. In both Figure 11(a) and Figure 11(b), the OR policy takes less time as compared to that of the AND policy when we consider the matching case because we use a short circuit evaluation (explained in Section 9.1) to evaluate both OR and AND gates. Finally, we consider the most complex case in which we increase the number of tags and the number of interest items together (equally) with 2 attributes. Similar to Figure 11(a) and Figure 11(b), it takes less time to evaluate the OR policy as compared to that of the AND policy. Next, we consider the worst case in which there are 5 tags and 5 interest items with a 2-attribute policy conjuncted using OR. Since there are 2 attributes in the policy tree with 5 tags each, there will be 10 leaf nodes in the encrypted policy. Furthermore, 2 attributes with 5 interest items each will make 10 trapdoors in the subscription list. The broker checks whether any encrypted leaf node in the policy matches with any trapdoor in the subscription list. In this worst case, the broker runs the **Test** function 100 times, thus taking approximately 1324 ms. In addition to this experiment, we measured the overhead for running the **Test** function and discovered that it takes 13.28 ms. This implies that the real overhead comes from the **Test** function, that is, in fact, a bilinear pairing operation. Hence, the matching operation is dependent on how efficient the bilinear pairing is. The best and worst case complexities of this phase are $\Omega(1)$ and $O(|A_P^*| \cdot |T| \cdot |A_S^*| \cdot |I|)$, respectively.

8.5 Subscriber's Decryption Phase

A subscriber receives the encrypted content (along with the encrypted policy) from the broker if the encrypted interest satisfies the encrypted policy associated with the encrypted content. During the decryption phase, first a subscriber strips off the tags from the policy and then performs decryption with the policy to recover the symmetric key, which is finally used to decrypt the contents. Figure 10(b) and Figure 10(c) show the effect of the number of attributes and interest items, respectively, on the subscriber's decryption time. In Figure 10(b), where we increase attributes from 2 to 10 while keeping the number of interest items constant i.e., 1, we consider both OR and AND policies to see the effect of attributes on the stripping of tags from the policy. Also, we show the performance overhead for the decryption that recovers the symmetric key. In Figure

10(c), we describe the case in which the number of interest items are increased from 2 to 10 (but attributes are kept constant i.e., 2), assuming the matching case, i.e., both the publisher and the subscriber are using the same tags and interest items, respectively. Here, the overhead does not increase with the increase in the number of interest items because we have implemented the short circuit evaluation to evaluate AND, OR and threshold gates. In fact, the trapdoor in the subscription matches interest items against tags in the policy, thus making policy evaluation successful without requiring further matches. Finally, the encrypted contents are decrypted using the symmetric key, which is recovered after we perform the CP-ABE decryption. Figure 8 shows the time required for decryption of the content using the AES key. Decryption of a piece of content of size 40 KB takes 0.87 ms (i.e., an average of 0.22 ms per KB). Overall, the complexity of subscriber's decryption is: $O(|A_P^*| \cdot |T| \cdot |A_S^*| \cdot |I| + |C|)$ in the worst case and $\Omega(|C|)$ in the best case.

Table 2 summarises the complexity of each phase in the lifecycle of PIDGIN while Table 1 describes the most frequently symbols used.

9 Discussion

9.1 Optimisation and Scalability

Optimisation using Short Circuit Evaluation. The real bottleneck is matching at brokers a set of encrypted policies against encrypted subscriptions. The large scale matching requires efficiency and some optimisations. One of the optimisations at brokers is implementation of short circuit evaluation for evaluating internal (i.e., non-leaf) nodes of the encrypted policy tree including OR and AND gates. That is, if the node is an OR gate then a broker can stop its evaluation and mark it satisfied once a single child node is satisfied, without performing further matches. Similarly, a broker can mark an AND gate unsatisfied when a single child node is marked unsatisfied. The short circuit evaluation can significantly reduce number of encrypted matches at brokers. This might be useful for the large policies involving a number of children in the policy tree. However, this might not speed up the performance when the set of policies or the number of subscriptions is very large.

Scalability. For matching a large set of encrypted policies against a large number of encrypted subscriptions, PIDGIN can take into account additional information that can drastically improve the overall performance. That is, a publisher can specify the content creation date while a broker can log time when the content was received. A subscriber can take advantage of this extra information by expressing additional constraints in subscription. For instance, a subscriber can express her subscription as: *all pieces of content matching with my interest, where the content is created or received in last two hours*. The content creation date and the content received time may help brokers to check whether subscriptions satisfy the published contents, without requiring encrypted matching. Furthermore, a publisher can publish content with Time To Live (TTL), meaning brokers should remove that particular piece of content after expiration of TTL. Similarly, subscribers also can include TTL with subscriptions to indicate that brokers can remove subscriptions from the network after expiration of TTL. The inclusion of TTL, in both the content and the subscription, will reduce both computation and storage needs.

9.2 Key Management

Deployment in Practical Scenarios. There are various options to setup the TKMA, an offline trusted entity that distributes keying material. It mainly depends on the scenario for which PIDGIN is deployed. For instance, for the military scenarios, it can be administrated by a military headquarter; similarly, in organisations, the admin department can manage it. However, it is challenging to setup the TKMA for various civilian applications. For those kinds of applications, the town or city administration could be one option. For emerging scenarios, such as social events, the organising authorities (such as event organisers) might own the TKMA.

Distributed TKMA. Without loss of generality, we can make the TKMA distributed. There are two main types of keys that are generated by the TKMA, the CP-ABE and the search keys. There are alraedy solutions for setting up multi-authority ABE [11, 12], where the CP-ABE key authorities can be distributed. Whereas, the key authority for generating the search keys is inherently distributed.

10 Related Work

The problem of encrypted matching in opportunistic networks is an instance of the wider problem of a search over encrypted data. Song *et al.* [13] propose a search scheme over encrypted data based on symmetric keys. The symmetric nature of the scheme rules out its applicability where mobile nodes communicate with each other without any prior contacts. The PEKS scheme [10] supports a search on encrypted data in the public key setting. In PIDGIN, we use the PEKS scheme as a building block; moreover, its usage in isolation does not solve privacy and confidentiality issues in opportunistic networks because it lacks the ability to regulate access on content while providing collusion-resistant decryption keys.

The ABE schemes can regulate access to content while guaranteeing collusion resistance. However, both variants of ABE including CP-ABE [5] and KP-ABE [6] do not protect the policies and attributes associated with content, respectively. In PIDGIN, we use CP-ABE [5] as a building block but only after we protect the policies because the original CP-ABE scheme does not specifically protect them. The complimentary KP-ABE [6] scheme does not protect attributes. While, Goyal *et al.* leave the problem of encrypted attributes as open [6], we address this challenging issue in this paper.

ESPOON [8] can protect security policies in outsourced environments. In [7], Asghar *et al.* propose ESPOON$_{ERBAC}$ that extends ESPOON with Encrypted Role-Based Access Control (ERBAC) that is deployable in outsourced environments. However, these solutions [7–9] assume no collusion between a user and a server. Thus, none of these solutions [7–9] are applicable to opportunistic networks in which each node can serve as all three roles including publisher, broker and subscriber.

There are schemes that preserve predicate privacy [14, 15] and assume that the predicate is evaluated at the receiver's end. Furthermore, schemes offering hidden credentials [16] and hidden policies [17] assume direct interaction between the sender and the receiving parties. Unfortunately, all such schemes cannot work in opportunistic networks where policy enforcement is delegated to untrusted brokers.

Shikfa *et al.* [18] propose a method that provides privacy and confidentiality in the context-based forwarding. However, their method is a different dimension of work than ours. In fact, their proposed scheme disseminates information in one direction, i.e., from publishers, without taking into account whether a subscriber is interested or not. In other words, it does not provide opportunity for a subscriber to subscribe. Moreover, our proposed scheme regulates access to content while offering more expressive and fine-grained policies as compared to the one proposed in [18].

Nabeel *et al.* [19] provide a solution for preserving privacy in content based publish-subscribe systems. In their approach, brokers in outsourced environments make routing decisions without knowing the content. However, they assume that subscribers get registered with publishers prior to any communication and publishers share the symmetric key with subscribers. This solution cannot work in opportunistic network settings where loosely-coupled publishers and subscribers do not require any registration or key sharing with each other.

In the context of publish-subscribe systems, there many solutions that address privacy and security issues [20–22]. However, the state-of-the-art techniques are mainly based on centralised solutions that cannot be applied to opportunistic networks, where each node may serve as a publisher, a broker and a subscriber.

11 Conclusion and Future Work

This paper presents PIDGIN, a privacy-preserving interest and content sharing scheme for opportunistic networks that does not leak information to untrusted parties. To show the feasibility of our approach, we implemented PIDGIN and evaluated its performance by measuring the overhead incurred by cryptographic operations when run on a smartphone.

As evident from the performance evaluation, the real bottleneck is the overhead incurred by pairing operations at the brokers. In fact, an efficient pairing implementation would drastically improve the performance of the system. As future work, we would investigate possible optimisations and the use of an efficient pairing implementation, such as one proposed in [23].

References

[1] Emarketer, "Smartphones, tablets drive faster growth in ecommerce sales - mobile will take a greater percentage of total ecommerce retail sales," http://goo.gl/48Mbp, April 2013, last accessed: August 7, 2013.

[2] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic networking: data forwarding in disconnected mobile ad hoc networks," *Communications Magazine, IEEE*, vol. 44, no. 11, pp. 134–141, 2006.

[3] "Haggle: An eu funded project," http://www.haggleproject.org/, June 2010, last accessed: August 7, 2013.

[4] E. Nordström, P. Gunningberg, and C. Rohner, "A search-based network architecture for mobile devices," Department of Information Technology, Uppsala University, Tech. Rep., 2009.

[5] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, 2007. SP '07. IEEE Symposium on*, may 2007, pp. 321 –334.

[6] 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*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 89–98. [Online]. Available: http://doi.acm.org/10.1145/1180405.1180418

[7] M. R. Asghar, M. Ion, G. Russello, and B. Crispo, "ESPOON$_{ERBAC}$: Enforcing security policies in outsourced environments," *Elsevier Computers & Security (COSE)*, vol. 35, pp. 2–24, 2013, special Issue of the International Conference on Availability, Reliability and Security (ARES). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404812001824

[8] ——, "ESPOON: Enforcing Encrypted Security Policies in Outsourced Environments," in *The Sixth International Conference on Availability, Reliability and Security*, ser. ARES'11. IEEE Computer Society, August 2011, pp. 99–108.

[9] A. Kapadia, P. P. Tsang, and S. W. Smith, "Attribute-based publishing with hidden credentials and hidden policies," in *NDSS*. The Internet Society, 2007.

[10] D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology - EUROCRYPT 2004*, ser. Lecture Notes in Computer Science, C. Cachin and J. Camenisch, Eds. Springer Berlin Heidelberg, 2004, vol. 3027, pp. 506–522. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24676-3_30

[11] M. Chase and S. S. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *Proceedings of the 16th ACM conference on Computer and communications security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 121–130. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653678

[12] M. Chase, "Multi-authority attribute based encryption," in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, S. Vadhan, Ed. Springer Berlin Heidelberg, 2007, vol. 4392, pp. 515–534. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-70936-7_28

[13] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on*, 2000, pp. 44–55.

[14] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, O. Reingold, Ed. Springer Berlin Heidelberg, 2009, vol. 5444, pp. 457–473. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00457-5_27

[15] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Advances in Cryptology EUROCRYPT 2008*, ser. Lecture Notes in Computer Science, N. Smart, Ed. Springer Berlin Heidelberg, 2008, vol. 4965, pp. 146–162. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-78967-3_9

[16] J. E. Holt, R. W. Bradshaw, K. E. Seamons, and H. Orman, "Hidden credentials," in *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, ser. WPES '03. New York, NY, USA: ACM, 2003, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/1005140.1005142

[17] K. Frikken, M. Atallah, and J. Li, "Attribute-based access control with hidden policies and hidden credentials," *Computers, IEEE Transactions on*, vol. 55, no. 10, pp. 1259–1270, 2006.

[18] A. Shikfa, M. Önen, and R. Molva, "Privacy and confidentiality in context-based and epidemic forwarding," *Computer Communications*, vol. 33, no. 13, pp. 1493 – 1504, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366410002136

[19] M. Nabeel, N. Shang, and E. Bertino, "Efficient privacy preserving content based publish subscribe systems," in *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, ser. SACMAT '12. New York, NY, USA: ACM, 2012, pp. 133–144. [Online]. Available: http://doi.acm.org/10.1145/2295136.2295164

[20] S. Choi, G. Ghinita, and E. Bertino, "A privacy-enhancing content-based publish/subscribe system using scalar product preserving transformations," in *Database and Expert Systems Applications*, ser. Lecture Notes in Computer Science, P. Bringas, A. Hameurlain, and G. Quirchmayr, Eds. Springer Berlin Heidelberg, 2010, vol. 6261, pp. 368–384. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15364-8_32

[21] N. Shang, M. Nabeel, F. Paci, and E. Bertino, "A privacy-preserving approach to policy-based content dissemination," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, 2010, pp. 944–955.

[22] M. Srivatsa and L. Liu, "Secure event dissemination in publish-subscribe networks," in *Distributed Computing Systems, 2007. ICDCS '07. 27th International Conference on*, 2007, pp. 22–22.

[23] G. Grewal, R. Azarderakhsh, P. Longa, S. Hu, and D. Jao, "Efficient implementation of bilinear

pairings on arm processors," in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, L. Knudsen and H. Wu, Eds. Springer Berlin Heidelberg, 2013, vol. 7707, pp. 149–165. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35999-6_11