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

Muhammad Rizwan Asghar*
CREATE-NET
Trento, Italy
asghar@create-net.org

Ashish Gehani
SRI International
Menlo Park, California, USA
ashish.gehani@sri.com

Bruno Crispo
University of Trento
Trento, Italy
crispo@disi.unitn.it

Giovanni Russello
University of Auckland
Auckland, New Zealand
g.russello@auckland.ac.nz

## 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, content sharing in particular has attracted significant attention. To support content sharing, opportunistic networks often implement a publish-subscribe system in which users may publish their own content and indicate interest in other content through subscriptions. Using a smartphone, any user can act as a broker by opportunistically forwarding both published content and interests within the network.

Unfortunately, opportunistic networks are faced with serious privacy and security issues. Untrusted brokers can not only compromise the privacy of subscribers by learning their interests but also can gain unauthorised access to the disseminated content. This paper addresses the research challenges inherent to the exchange of content and interests 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 demonstrate the feasibility and efficiency of the solution by implementing a prototype and analysing its performance on smart phones.

## Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access control; E.3 [**Data**]: Data Encryption

*Most of this research was carried out while visiting SRI International as an International Fellow.


*ASIA CCS'14,* June 4–6, 2014, Kyoto, Japan.


## Keywords

Secure Opportunistic Networks; Privacy-preserving Content Sharing; Sensitive Policy Enforcement; Encrypted CP-ABE Policies; Secure Haggle

## 1. INTRODUCTION

Over the last few years, the use of smartphones has grown dramatically and is predicted to increase even more in the coming years [7]. Considering the pervasive nature of smartphones, mobile opportunistic networks can 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 [18].

Haggle [1], an example of such a network architecture, allows smartphones to opportunistically share content via short-range communication [17]. 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 interests. 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.

Opportunistic networks can be used to exchange 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 exchange content and interests that neither (i) compromises the privacy of subscribers nor (ii) provides unauthorised access to untrusted brokers.

Cryptographic approaches such as Attribute-Based Encryption (ABE), which include Ciphertext-Policy ABE (CP-ABE) [4] and Key-Policy ABE (KP-ABE) [9], offer fine-grained control over content but leak information about the policies and attributes that protect that content, respectively. To protect the access policies, state-of-the-art solutions exist to enforce sensitive policies in outsourced environments [2, 3, 12]. 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 – that is, serving as both a client and a server.

This paper presents PIDGIN, an interest and content sharing scheme that preserves privacy, in which:

- brokers match subscribers' interests against content access policies associated without compromising the subscribers' privacy (by learning content attributes or node interests).
- 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 a motivating scenario, and lists some of the major research challenges for interest and content sharing in opportunistic networks with privacy-preserving guarantees. In Section 3, we describe the system model. Next, we explain the proposed scheme in Section 4. Section 5 elaborates on PIDGIN. In Section 6, we provide the concrete construction. Section 7 reports on our security and performance analysis. Related work is reviewed in Section 8. Finally, we conclude in Section 9 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 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 specifies its interests while a publisher node shares content with its neighbouring nodes [17]. These neighbouring nodes are intermediate nodes, known as brokers, that epidemically disseminate interests and content within the network. A resolution takes place when a broker node finds a match between the interests of a subscriber and the tags associated with published content. As a result of resolution, a broker forwards content to the subscriber.


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

## 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.

**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 soldier can express interest in *Curiosity* by subscribing to it, as illustrated in Figure 1. Here we assume that someone acting as a broker receives both the interest and the tagged image. Whenever that happens, the broker checks whether the interests of a subscriber matches any tags 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. 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 interests of subscribers and the tags associated with the 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 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 obtain content without exposing her interests to untrusted brokers?

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

**C5** Assuming the loose coupling of the publish- subscribe model, how do we address the challenges above (i.e., *C1-C4*) without sharing keys between publishers and subscribers?

## 3. SYSTEM MODEL

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

**Publishers** are nodes that originate content.

**Subscribers** are nodes that express interests.

**Brokers** are nodes that may receive and disseminate both content and interests. They evaluate whether available content matches known node interests. On successful evaluation, they forward content to subscribers.

**Trusted Key Management Authority (TKMA)** is an offline entity that distributes cryptographic key material (including private keys and public parameters) to all nodes. This is done out-of-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. Further, we consider the TKMA to be fully trusted and assume that it only 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 information being exchanged.

## 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 the research challenges listed in Section 2.3. Next, we incrementally address each of the research challenges. Finally, we describe the proposed scheme.


**Figure 2: Regulation of access to content using CP-ABE policies.**

### Scheme I: Regulate Access to Content

To preserve the confidentiality of content, a publisher can specify who should have access. One possible approach for the publisher to regulate access to content is by employing an ABE scheme, such as CP-ABE [4] or KP-ABE [9]. ABE supports fine-grained policies for limiting content access. In this work, we consider CP-ABE because it allows a publisher to exert control over access to content, as described in the use case scenario. In contrast, with the use of KP-ABE, a key generation authority controls who can access content.

Figure 2 illustrates the scheme where the image is encrypted according to this policy: *either a Major or a Solider from the Infantry can gain 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 subscribers if their interests match the content's tags.

This approach preserves the confidentiality of the 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)*.

### Scheme II: Refined Authorisation Checks

This scheme extends Scheme I and resolves the flooding problem *C4*. In this scheme, a subscriber may send attributes and interests to brokers so that a broker can perform an authorisation check prior to forwarding the content. 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 a subscriber if and only if (i) the root node of the policy is marked as satisfied and (ii) the interests match the tags.

This scheme targets both the access control problem *(C1)* and the flooding problem *(C4)*. However, it still has 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 the access policies that there is information intended for a *Major*. Second, the cleartext interests of a subscriber may also leak information, i.e., *C3*. For instance, the enemy may learn that the content or interests concern the *Curiosity* mission.

### Scheme III: Hashing Private Information

In order to partially overcome the issue of subscriber privacy *(C2)*, a subscriber and a publisher may hash attributes and leaf nodes in the policy tree, respectively. Similarly, a subscriber's interests could be protected by calculating the hash values of interest items and content tags. In this scheme, a broker forwards encrypted content to subscribers if and only


**Figure 3: Private information is hidden through replacement of leaf nodes in the CP-ABE policy (that are tags, attributes, or interests) with their corresponding hashes.**

if (i) the hashed values of the interests match the hashed values of the tags (i.e., h('Curiosity')) and (ii) hashed 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. 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 content tags). The pre-calculated list of hashes may reveal the original attributes, leaf nodes in the policy tree, interests, and tags.

## Scheme IV: Countering Dictionary Attacks

To harden the system against pre-computed dictionary attacks, a publisher can replace each leaf node in the policy with the hash of a concatenated pair of a tag and an attribute. Similarly, a user 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 the system against pre-computed dictionary attacks but also decreases the number of comparisons performed by the broker when 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 must perform two separate checks: one to check the authorisation and one to match the interest. Though this significantly expands the range of elements that must be computed in a dictionary attack, the scheme is still vulnerable to adversaries with sufficient storage.

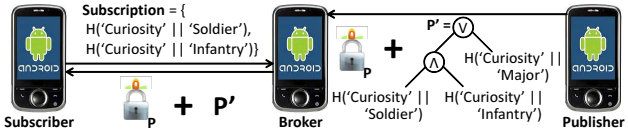


**Figure 4: Hardening against a pre-computed dictionary attack through concatenating (i) each content tag with a leaf node in the CP-ABE policy, and (ii) each interest item with an attribute, then calculating the hash of each concatenated string.**

## PIDGIN: The New Scheme

Our proposed scheme, PIDGIN, aims to address all the 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 [5] to protect attributes, interests, tags, and leaf nodes in the policy tree.

The PEKS scheme consists of four basic functions – **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 the given attribute (i.e., *Soldier*). To protect policies and tags, a publisher can replace each leaf node in the policy tree with the output of 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 interests by replacing each interest item in the subscription list with the output of a **Trapdoor** function that takes as input an interest item (i.e., *Curiosity*) and the private key (generated by the PEKS scheme) corresponding to the attribute.

A broker performs matching under encryption between transformed policies and 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 the leaf nodes, and progresses to the root. If the root is satisfied, the broker will forward the content along with the encrypted policy to the subscribers.


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

# 5. DESIGN

## Setup and Key Generation

During the initialisation phase, the system sets up both the CP-ABE and PEKS primitives. The PIDGIN TKMA generates and distributes keys. It also generates a private set of attributes (i.e., CP-ABE private keys) and sends these securely to the subscriber via out-of-band communication. The TKMA broadcasts the public part of attributes (i.e., CP-ABE public keys). Since the attributes are protected using the PEKS scheme, the TKMA also generates a pair of keys corresponding to each attribute. Similar to CP-ABE key distribution, the TKMA sends the private and public parts of the PEKS key pairs to the subscribers 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.

[1]The Etag function is called PEKS in the original paper [5].

## Publisher's Encryption

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

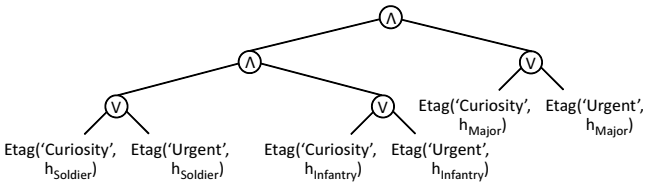


**Figure 6: An extended CP-ABE policy with two tags, i.e., 'Curiosity' and 'Urgent'.**

To extend CP-ABE policies for PEKS, a publisher considers each leaf node in the policy tree as well as the tags that are associated with the content. If there is just a single tag then the publisher replaces the leaf node with the **Etag** function's output, as was 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, the 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 that is the disjunction (i.e. OR) of the newly generated encrypted tags. Figure 6 illustrates an example of a policy with two tags, i.e., 'Curiosity' and 'Urgent'.

## Subscriber's Encryption Phase

In order to protect the privacy of the interests and attributes of a subscriber, each interest item is encrypted using the subscriber's private key (generated by the PEKS scheme) that corresponds to the attribute. PIDGIN considers the possibility that a subscriber might have multiple attributes and interests. Each interest item is encrypted with each private key (generated by the PEKS scheme) that corresponds to an attribute. Figure 5 describes the case in which a subscriber has two attributes and subscribes with a single interest item. Let us assume that a subscriber instead 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 or the attribute.

## 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 policy, it evaluates whether they match. For this, the broker runs a function that recursively evaluates the encrypted policy tree. The **Test** function checks each encrypted leaf node in the policy against the encrypted interest item in the subscription.

The **Test** function returns either *TRUE* or *FALSE*, indicating only whether the trapdoor matched the encrypted tag or not, respectively. By running the **Test** function, a broker does not learn the tag or the interest item because both are encrypted and the matching only occurs under encryption. If an encrypted tag in the policy tree matches with any trapdoor in the subscription list, that node is marked as satisfied. After evaluating the 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.

## Subscriber's Decryption Phase

Once a subscriber receives a piece of encrypted content along with its encrypted policy, they must first recover the original CP-ABE policy. Each leaf node (with a single tag, as in Figure 2) or subtree of tags (if more than one tag is used, as in Figure 6) is replaced with the corresponding attribute. Before sharing their encrypted interests, a subscriber checks a lookup table that associates attributes with corresponding trapdoors. If a trapdoor matches any encrypted tag in the leaf node of the policy, the lookup table will used to find the corresponding attribute. Next, each leaf node (in the case of a single tag) or subtree of tags (if multiple tags are used) will be replaced with the attribute that was found. If no match was found, a dummy attribute will be inserted. This recovers the original CP-ABE policy (i.e., one shown in Figure 2) that can be used by the CP-ABE decryption function to get the symmetric key needed to decrypt the content.

# 6. 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.** 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 $\mathbf{parent}(x)$ represents the parent of a node $x$, $\mathbf{att}(x)$ denotes the attributes associated with leaf node $x$, and $\mathbf{index}(x)$ 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

$e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ 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:

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

**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 Primitives

**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 $A'_x = index(z) : z \in A_x$)

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

(by construction)

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

(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. EVALUATION

PIDGIN is designed to optimize performance by encrypting content with a symmetric key. This keys is then encrypted with the CP-ABE policy. The leaf nodes in the policy tree are protected using the PEKS **Etag** primitive described by Boneh *et al.* [5]. PEKS is semantically secure against a chosen keyword attack in the random oracle model, assuming that the Bilinear Diffie-Hellman (BDH) problem is hard (as proved by Theorem 3.1 of the PEKS paper [5]). However, the CP-ABE policy structure is not protected and leaks information about the number of attributes or tags used. This leak could partially be tackled by the inclusion of dummy attributes at the cost of increased complexity. PIDGIN addresses the case where brokers collude but must still not gain access to content, policies, or subscriptions. If a broker colludes with a subscriber, together they 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, PIDGIN remains safe due to the randomness embedded in each subscriber's (CP-ABE) decryption key.

## Platform

As a proof of concept, we developed a prototype of PIDGIN. It is based on an extension of the open source *libfenc*[2] library that is written in C. The library includes an implementation of CP-ABE. Since we proposed to extend CP-ABE with PEKS, we have implemented PEKS in C using the PBC[3] library. (PBC also used by libfenc.) PBC is based on Elliptic Curve Cryptography (ECC). The curve we use in our experimentation is of type A.

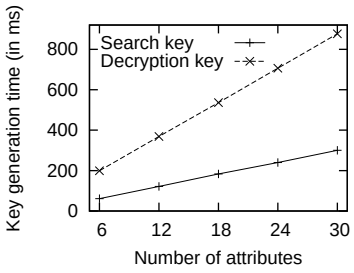


**Figure 7: Effect of attributes on key generation time.**

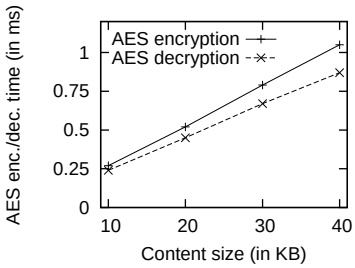


**Figure 8: Effect of content size on AES encryption and decryption time.**

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

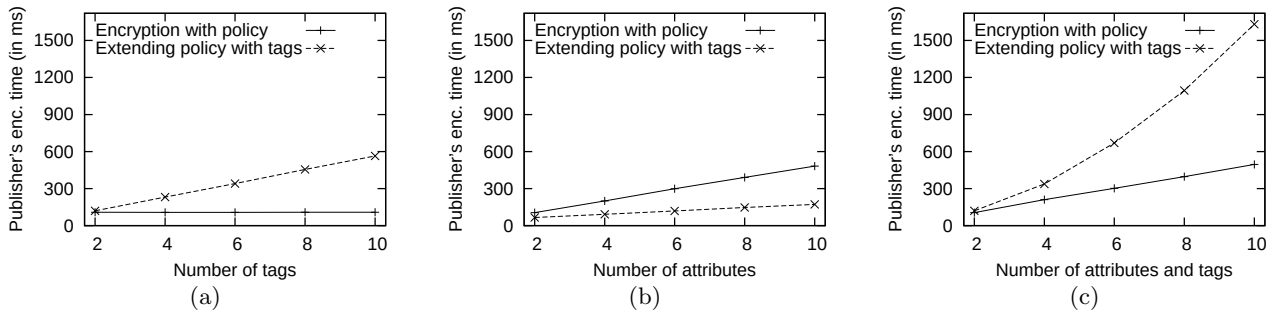


**Figure 9: Effect of (a) tags, (b) attributes, and (c) both tags and attributes on publisher's encryption time.**

After extending CP-ABE with PEKS (on the x86 architecture), we cross-compiled it for the ARM architecture. This allowed us to test the prototype on a Samsung Galaxy SIII smartphone (running Android version 4.1.2 with kernel version 3.0.31, and using 1 GB RAM and 1.4 GHz processor). To use the prototype, we cross-compiled both GMP[4] (the GNU Multiple Precision arithmetic library required by PBC) and PBC for the ARM architecture. These were all installed on the smartphone. We present results that are averaged over 20 runs.

Our analysis does not consider battery consumption since the prototype does not implement energy-related optimizations (that could be deployed without compromising private information). For instance, adding Time-To-Live (TTL) tags to content and subscriptions can bound the quantity of matching operations performed by brokers. Similarly, subscriptions could specify bounds on the freshness of content.

## Setup and Key Generation

During the initialisation phase, system-wide key material is generated. In the key generation phase, both search and decryption keys are generated for a given set of attributes. Both phases can be run on a computer since the keys are distributed out-of-band. However, we have considered running both phases on a smartphone (with the specifications described above), should these need to occur in the field. The initialisation phase takes 108.5 milliseconds (ms). The time for generating search keys grows linearly with an increase in the 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 for decryption keys also grows linearly with an increase in the number of attributes – 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 $A$.

## Publisher's Encryption

A publisher encrypts content with a randomly generated symmetric key. Our prototype uses AES for the symmetric cipher. The symmetric key is encrypted with the CP-ABE policy for the content. The CP-ABE policy is extended with tags that are also encrypted. Figure 8 shows the time to perform symmetric encryption, which is seen to grow linearly with the content size ($C$). Encryption of a piece of content of size 40 kilobytes (KB) takes 0.105 ms (at an average of 0.026 ms per KB). To measure the performance overhead of the encryption time, we varied the numbers of tags and attributes ($A_P^*$), as shown in Figure 9.

Figure 9(a) and Figure 9(b) show the effect of the number of tags and attributes on the publisher's encryption time, respectively. Figure 9(a) shows the effect of tags (ranging from 2 to 10) while keeping the number of attributes constant (i.e., 2 attributes - the minimum attributes required to create a policy with an AND or an OR operation). As expected, the time to extend a policy with tags grows linearly with the number of tags. Figure 9(b) reports the effect of varying the number of attributes (from 2 to 10) in a policy while using a single tag. The time for encryption of the symmetric key with the policy grows linearly with the number of attributes. As the number of attributes increases, the time to extend a policy with a set of tags grows linearly.

Figure 9(c) reports the most complex case, where we increase both attributes and tags simultaneously. The growth of the time needed to extend a policy with tags is quadratic, depending on both the number of attributes and the number of tags. We ran experiments with the same number of tags and attributes to analyze this case. Extension of a policy with 2 attributes, using 2 tags for each, takes approximately 120 ms, while extension of a policy with 10 attributes, using 10 tags for each, takes 1632 ms. More generally, the asymptotic complexity of the publisher's encryption phase is $\Theta(|A_P^*| \cdot |T| + |C|)$.

## Subscriber's Encryption

Figure 10 shows the performance overhead incurred during the encryption (in Figure 10(a)) and decryption phases (in Figure 10(b) and Figure 10(c)). The subscriber's encryption phase complexity depends on the number of interest items ($I$) and attributes ($A_S^*$).

We first examined the effect on the subscription time that resulted from varying the number of attributes and interests independently. We increased the attributes from 2 to 10 while keeping the number of interest items constant (i.e., 1 interest item). Generation of trapdoors for 10 attributes with a single interest item each took 106 ms. Next, we observed the effect of varying the number of interest items from 2 to 10 while keeping number of attributes constant (i.e., 2 attributes combined with either an AND or OR operation). It took approximately 284 ms to encrypt an interest containing 10 items. As illustrated in Figure 10(a), varying only the number of attributes or interest items affects the subscriber's encryption time linearly.

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

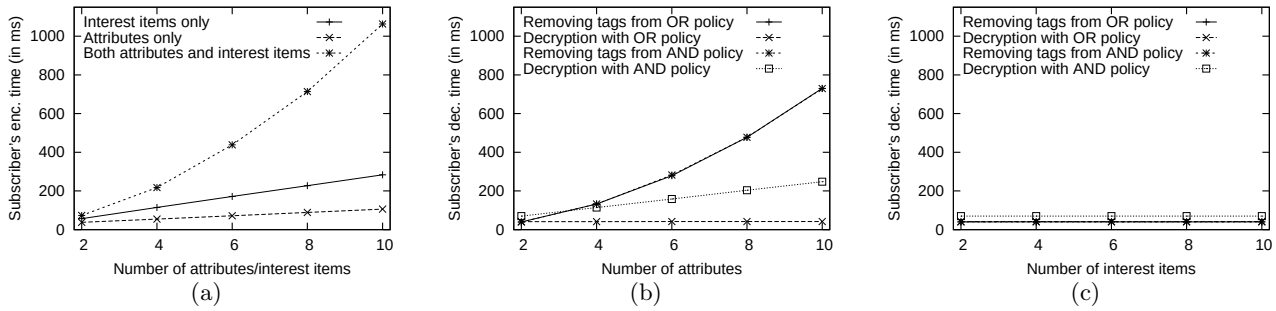


**Figure 10: Effect of (a) attributes and interest items on subscriber's encryption time. Effect of (b) attributes and (c) tags on subscriber's decryption time.**

We also considered the case where both the number of attributes and interest items varied. For this experiment, we assumed that the number of attributes and interest items are the same – that is, if there are two attributes, it means there are two interest items per attribute as well. Similarly, 10 attributes were used with 10 interest items for each. This took 1063 ms. The combined effect of attributes and interest items causes quadratic growth in the subscriber's encryption time, as shown in Figure 10(a). The asymptotic complexity is: $\Theta(|A_S^*| \cdot |I|)$.

As explained above, the curve we used in our experiment is of type A. Using this curve, the storage required for an encrypted tag and a trapdoor are 256 and 128 bytes, respectively.

## Broker's Matching

This is the most important step in PIDGIN. During this phase, a broker attempts to match an encrypted subscription with an encrypted policy associated with encrypted content. We first examined the effect of varying the numbers of tags and interest items independently, while keeping the number of attributes constant (i.e., 2 attributes, the minimum necessary to have a policy with an AND or OR operation). We also consider both the cases where the policy contains AND and OR operations, as well as the worst case when no match occurs.

Figure 11 shows the performance analysis of this phase. Figure 11(a) illustrates the effect of the number of tags on the matching time while keeping the number of interest items constant, i.e., 1. As the graph shows, the matching time increases linearly with the number of tags. Similarly, we measured the effect of the number of interest items on the matching time while keeping the number of tags constant, i.e., 1. Figure 11(b) shows that the matching time grows linearly with 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 a match occurs. This is because both OR and AND operations use short circuit evaluation to halt further policy analysis once it is known to be true or false.

Finally, we consider the most complex case where the number of tags and interest items are increased together. Similar to the cases examined in Figure 11(a) and Figure 11(b), it takes less time to evaluate an OR policy when compared to an AND policy. We considered the worst case in which there are 5 tags and 5 interest items with a 2-attribute policy combined with an OR. Since there are 2 attributes in the policy tree with 5 tags each, there are a total of 10 leaf nodes in the encrypted policy. The 2 attributes with 5 interest items each will require 10 trapdoors in the subscription.

The broker checks whether any encrypted leaf node in the policy matches any trapdoor in the subscription. In this worst case, the broker runs the **Test** function 100 times, taking approximately 1324 ms. In addition to this experiment, we measured the overhead of running the **Test** function and discovered that it takes 13.28 ms. We can infer that the primary overhead is from the **Test** function, which is implemented with a bilinear pairing operation. Hence, the matching operation heavily depends on the efficiency of the bilinear pairing primitives. The best and worst case complexities of this phase are $\Omega(1)$ and $O(|A_P^*| \cdot |T| \cdot |A_S^*| \cdot |I|)$, respectively.

## Subscriber's Decryption

A subscriber receives a piece of encrypted content (along with the corresponding encrypted policy) from the broker if its encrypted interests matched. During the decryption phase, a subscriber first removes the tags from the policy. If they are successful, they can then perform decryption with the policy to recover the symmetric key, which is finally used to decrypt the content.

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), we increase the number of attributes from 2 to 10 while keeping the number of interest items constant i.e., 1. we consider both AND and OR policies to see the effect of attributes on tag removal. The performance overhead for the decryption that recovers the symmetric key is also shown.

Figure 10(c) reports on the case where the number of interest items are increased from 2 to 10 (but the attribute count is kept constant i.e., 2). The publisher and subscriber use the same tags and interest items, respectively, to ensure that the matching succeeds. The overhead does not increase with the number of interest items. This is due to our implementation of short circuit evaluation of policies with AND and OR operations. The trapdoor in the subscription matches interest items against tags in the policy. This allows the policy evaluation to proceed without further matching.

In the final step, a symmetric key is recovered if the CP-ABE decryption succeeds. This symmetric key is used to decrypt the content. Figure 8 shows the time required for content decryption using AES. Decrypting a 40 LB piece of

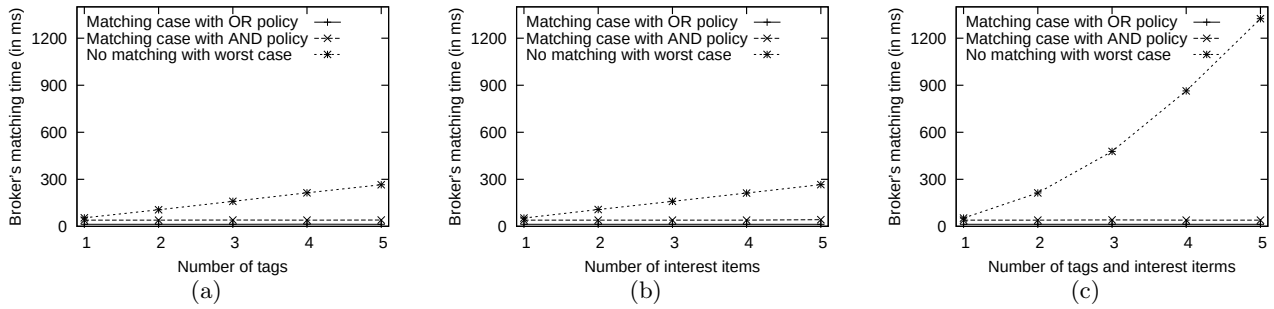


**Figure 11: Effect of (a) tags, (b) interest items, and (c) both tags and interest items, on broker's time to perform matching under encryption.**

**Table 1: Description of symbols used.**

| Symbol | Description |
|---|---|
| $A$ | List of attributes |
| $A_P^*$ | Llist of attributes used to encrypt content |
| $A_S^*$ | List of attributes used to encrypt interests |
| $C$ | Content |
| $I$ | List of keywords a subscriber is interested in |
| $T$ | List of search tags associated with content |

**Table 2: Complexity of each phase of PIDGIN.**

| Phase | 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 | $\Omega(\|C\|)$ | $O(\|A_P^*\| \cdot \|T\| \cdot \|A_S^*\| \cdot \|I\| + \|C\|)$ |

content takes 0.87 ms (i.e., an average of 0.22 ms per KB). 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 of PIDGIN. The symbols used are described in Table 1.

## 8. RELATED WORK

The problem of encrypted matching in opportunistic networks is an instance of the larger problem of search over encrypted data. Song *et al.* [22] propose a search scheme over encrypted data based on symmetric keys. The symmetric nature of the scheme rules out its applicability when mobile nodes communicate with each other without any prior contact. The PEKS scheme [5] supports search of encrypted data in the public key setting. PIDGIN uses 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 to content while providing collusion-resistant decryption keys.

ABE schemes can regulate access to content while guaranteeing collusion resistance. However, both the CP-ABE [4] and KP-ABE [9] variants do not protect the policies and attributes associated with content, respectively. PIDGIN uses CP-ABE [4] as a building block after we protect the policies (since the original CP-ABE scheme does not protect them). The complimentary KP-ABE [9] scheme does not protect attributes. While Goyal *et al.* leave the problem of encrypted attributes as an open question [9], this challenging issue is addressed in this paper.

ESPOON [2] can protect security policies in outsourced environments. Asghar *et al.* [3] propose ESPOON$_{ERBAC}$ to extend ESPOON with Encrypted Role-Based Access Control (ERBAC) in outsourced environments. However, these solutions [2, 3, 12] assume that there is no collusion between a user and a server. Thus, none of these solutions [2, 3, 12] are applicable to opportunistic networks where each node can serve in all three roles – that is, publisher, broker, and subscriber.

There are schemes that protect policies [13, 14, 16, 20] by assuming that the policy is evaluated at the receiver's end. Furthermore, schemes offering hidden credentials [11] and hidden policies [8] assume direct interaction between the sender and the receiving parties. Unfortunately, such schemes cannot work in opportunistic networks where policy enforcement is delegated to untrusted brokers.

Shikfa *et al.* [21] propose a method that provides privacy and confidentiality in the content-based forwarding setting. However, their method is orthogonal to our work. 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 the opportunity for a node to subscribe. Moreover, our proposed scheme regulates access to content while offering more expressive and fine-grained policies when compared to theirs [21]. Similarly, schemes [14, 16] hiding CP-ABE policies assume end-to-end communication while we address the case where brokers in the middle must make access decisions before forwarding content in the network.

Nabeel *et al.* [15] 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 register with publishers prior to any communication and that publishers share symmetric keys with subscribers. This solution cannot work in opportunistic network settings where loosely-coupled publishers and subscribers may not have a chance to pre-arrange key sharing with each other.

In the context of publish-subscribe systems, there are many solutions that address privacy and security issues [6, 19, 23]. 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 publisher, broker, and subscriber.

## 9. CONCLUSION

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 for cryptographic operations 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. Future work could investigate possible optimisations and the use of more efficient pairing implementations [10].

## 10. ACKNOWLEDGEMENTS

The authors would like to thank Daniele Miorandi for providing his valuable feedback on the initial draft, as well as the anonymous reviewers for their comments and suggestions to improve the quality of this work. The work of the first and third authors has been partially supported by EIT ICT Labs and the TENACE PRIN project (Grant no. 20103P34XC) funded by the Italian MIUR, respectively.

## 11. REFERENCES

[1] Haggle: An EU funded project. `http://www.haggleproject.org/`, June 2010. Last accessed: August 7, 2013.

[2] M. R. Asghar, M. Ion, G. Russello, and B. Crispo. ESPOON: Enforcing Encrypted Security Policies in Outsourced Environments. In *The Sixth International Conference on Availability, Reliability and Security*, ARES'11, pages 99–108. IEEE Computer Society, August 2011.

[3] M. R. Asghar, M. Ion, G. Russello, and B. Crispo. ESPOON$_{ERBAC}$: Enforcing security policies in outsourced environments. *Elsevier Computers & Security (COSE)*, 35:2–24, 2013. Special Issue of the International Conference on Availability, Reliability and Security (ARES).

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

[5] D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer Berlin Heidelberg, 2004.

[6] S. Choi, G. Ghinita, and E. Bertino. A privacy-enhancing content-based publish/subscribe system using scalar product preserving transformations. In P. G. Bringas, A. Hameurlain, and G. Quirchmayr, editors, *Database and Expert Systems Applications*, volume 6261 of *Lecture Notes in Computer Science*, pages 368–384. Springer Berlin Heidelberg, 2010.

[7] 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.

[8] K. Frikken, M. Atallah, and J. Li. Attribute-based access control with hidden policies and hidden credentials. *Computers, IEEE Transactions on*, 55(10):1259–1270, 2006.

[9] 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*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM.

[10] G. Grewal, R. Azarderakhsh, P. Longa, S. Hu, and D. Jao. Efficient implementation of bilinear pairings on ARM processors. In L. Knudsen and H. Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 149–165. Springer Berlin Heidelberg, 2013.

[11] 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*, WPES '03, pages 1–8, New York, NY, USA, 2003. ACM.

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

[13] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In N. Smart, editor, *Advances in Cryptology EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer Berlin Heidelberg, 2008.

[14] J. Lai, R. Deng, and Y. Li. Fully secure cipertext-policy hiding CP-ABE. In F. Bao and J. Weng, editors, *Information Security Practice and Experience*, volume 6672 of *Lecture Notes in Computer Science*, pages 24–39. Springer Berlin Heidelberg, 2011.

[15] 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*, SACMAT '12, pages 133–144, New York, NY, USA, 2012. ACM.

[16] T. Nishide, K. Yoneyama, and K. Ohta. Attribute-based encryption with partially hidden encryptor-specified access structures. In S. Bellovin, R. Gennaro, A. Keromytis, and M. Yung, editors, *Applied Cryptography and Network Security*, volume 5037 of *Lecture Notes in Computer Science*, pages 111–129. Springer Berlin Heidelberg, 2008.

[17] E. Nordström, P. Gunningberg, and C. Rohner. A search-based network architecture for mobile devices. Technical report, Department of Information Technology, Uppsala University, 2009.

[18] L. Pelusi, A. Passarella, and M. Conti. Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *Communications Magazine, IEEE*, 44(11):134–141, 2006.

[19] 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*, pages 944–955, 2010.

[20] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In O. Reingold, editor, *Theory of Cryptography*, volume 5444 of *Lecture Notes in Computer Science*, pages 457–473. Springer Berlin Heidelberg, 2009.

[21] A. Shikfa, M. Önen, and R. Molva. Privacy and confidentiality in context-based and epidemic forwarding. *Computer Communications*, 33(13):1493 – 1504, 2010.

[22] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

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