Securing Resource Discovery in Content Hosting Networks

Sushama Karumanchi[1(✉)], Jingwei Li[2], and Anna Squicciarini[1]

[1] College of Information Sciences and Technology, Pennsylvania State University, State College, USA
sik5273@psu.edu
[2] College of Computer and Control Engineering, Nankai University, Tianjin, China

Abstract. Secure search query routing is a long-standing problem in distributed networks, which has often been addressed using "all-or-nothing" approaches, that require either full anonymity and encrypted routing or full trust on the routing nodes. An important problem with secure routing is how to guarantee the search query is transmitted in an expected way. In this paper, we tackle the problem of secure routing by considering a generic policy-driven routing approach, and focus on the steps required to verify in a fully distributed manner that a search query is routed in accordance to a requester's preferences and detect cheating nodes. We present an efficient and effective verification method for query routes, that is agnostic to the specific routing algorithm being used and achieves strong security guarantees. We cast our approach in the context of content dissemination networks (CDN) and show through experimental evaluations the performance of our approach.

Keywords: Resource discovery · Query routing · Security · Content dissemination networks · Malicious forwarding

1 Introduction

Content hosting or dissemination networks are those networks that store content in a distributed manner. Today, such networks are gaining popularity. Content providers such as Netflix and Youtube utilize content distribution networks to store their data [15]. Most of the current Internet activities are based on content retrieval than point-to-point communications [13]. Resources in content sharing and dissemination networks (CDN) are discovered through search queries, disseminated along the network using a routing protocol, raising potential security and privacy concerns against the query and the search route.

In these networks, user information privacy and security are considered important issues [15], as content providers, in addition to their own information, store their client's information in the CDNs. In order to sustain their businesses, clients' information should be handled very carefully.

J. Tian et al. (Eds.): SecureComm 2014, Part I, LNICST 152, pp. 153–173, 2015.
DOI: 10.1007/978-3-319-23829-6_12

One of the privacy and security problems of these environments is associated with the propagation path of search query, which may be very sensitive, and may ideally be handled only by trusted peers, due to the content of the query (and possible business interests associated with them). The query owner or requester might want to forward the query only to selected nodes in the network, according to the company data management policies of the requester. For instance, a user might request an album stored in his Flickr account, and Flickr uses Yahoo's cloud to store photos. The search query for the album might traverse the Internet over random routing nodes and the user might not prefer such a random routing path taken by his or her query and the content. Also, Flickr, in order to protect the customer's privacy might want the query to be routed only through specific nodes that satisfy certain user or company requirements. As highlighted by this example, we urge a practical method to detect cheating nodes in the query propagation path, that do not comply with the user or company requirements.

In this paper, we assume the existence of a policy based routing protocol in place (e.g., [11,19]), wherein the routing preferences of a node requesting a resource through a distributed search are expressed by means of a set of policy conditions. Our main goal is to detect nodes that tamper with such routing protocols by (i) forwarding the query to policy non-satisfying nodes and (ii) dropping the query even though there are policy satisfying nodes present. It is worth noting that our focus on *policy-compliant distributed search* is different from the problem of *protecting the content of search query*, which just aims at preventing other (policy-non-compliant) nodes from learning the content of search query, and can be easily achieved using one-to-many encryption [3,10, 18]. Here, we consider a more challenging issue, i.e., guaranteeing the query is transmitted in *correct path*, which not only implies *protecting the content of search query*, but also *limits unnecessary access of the query over the network*.

Toward developing solutions for ensuring policy-compliant distributed search, we design a two-phased routing compliance verification mechanism in the context of content dissemination networks. Our proposed scheme works by firstly identifying the correct path of search query propagation, and then checking the policy satisfiability of all the nodes in the path. Our scheme is secure, in terms of verifiability and non-repudiable search compliance, if the path is correctly identified in the first phase. We also consider practical methods to further improve the efficiency and enhance security of our proposed scheme. Note that our verification method does not presume a specific policy routing method. Rather, given a generic policy routing search wherein queries are routed across nodes based on conditions of the relaying nodes, we wish to ensure that it has been forwarded correctly, that is, as intended by the requester. We conduct an experimental analysis of our approach, and obtain an estimate of the computational overhead our approach generates. Our results show that our approach is efficient, even in case of large networks.

The rest of the paper is organized as follows. Section 2 overviews related work. Section 3 reviews the main cryptographic notions adopted by our scheme. Section 4 discusses threat model and main assumptions. We define the policy

compliant search in Sect. 5. In Sect. 6, we present our approach to detecting malicious nodes. Section 7 discusses the security aspects of our approach. In Sect. 8, we present our experimental analysis. We conclude in Sect. 9.

2 Related Work

Trust establishment is a well-known challenge in distributed networks. Malicious nodes can abuse the data or the established search query forwarding protocols in a number of ways. To address these issues, a large body of work exists on secure distributed networks, tackling sybil attacks, denial of service, free riders and cheat detection [4,9,15–17]. In the context of content dissemination networks, researchers have focused primarily on the issues of denial of service attacks [4,7,9], privacy and security of the content propagated within the content centric networks [23], and sybil attacks [15]. Some recent work has also explored issues related to access control in ad-hoc networks [12]. In addition, issues related to secure search query propagation are very crucial to content dissemination networks as searching is the main purpose of content dissemination networks, and hence the protection of search query propagation through the network is very important. In this work, we aim to tackle the security issues of search query propagation in distributed networks.

In this space, recent work has focused on efficient query processing. For instance, Durr et al. [5] analyzed different query forwarding strategies in privacy preserving social networks. Also, many have investigated intelligent query processing methods [2,14,20,22]. However, unlike in our work, intelligent processing methods do not consider the security aspects of the query forwarding process itself.

Zhang et al. [23] propose a mechanism to protect the confidentiality of data by encrypting them with identity based cryptography in content-centric networks. While it is sensible to utilize identity based cryptography to protect the confidentiality of the data propagated and selectively disseminate data, it is also important to detect malicious nodes in the network which propagate the data to false nodes. In this work, we employ efficient identity based signature schemes and attribute based encryption schemes to establish the integrity of the path taken by the search query and detect the malicious nodes that abuse the query forwarding algorithms. On a similar note, Padmanabhan and Simon [17], propose a mechanism to identify offending routers in a network and securely trace the path of the traffic. Their approach requires each node in a path to respond to the requester with an OK response that it received a packet. In contrast, we propose an efficient approach in which we use aggregated signatures to ensure the integrity of the path taken by a search query. Our protocol does not require a message from every node that the traffic or the query passes through. Mirzak et al. [16] also propose an approach to detect malicious routers, based only on the traffic information that each node has. Our approach is different from theirs in that it detects the malicious nodes mainly based on the attributes or properties of the nodes in the network, by making use of policies.

In summary, while several interesting works exist on policy compliance routing (e.g. [11,19]), we are not aware of any work on detection of malicious nodes that do not comply with the query forwarding protocol established for the network. Rather, previous works focus on policy specification, and assume that the nodes are honest. In this work, we detect malicious nodes that do not comply with such query routing protocols. Since search query forwarding is an important phase of the dissemination of content in content dissemination and peer to peer networks, we aim to provide a solution to efficiently and effectively support verifiable query forwarding in these networks.

3 Cryptographic Background

3.1 Attribute-Based Encryption

Attribute-based encryption (ABE) has been widely applied to impose fine-grained access control on encrypted data [18]. Two kinds of ABE have been proposed so far: key-policy attribute-based encryption (KP-ABE) [10] and ciphertext-policy attribute-based encryption (CP-ABE) [3]. In KP-ABE, each ciphertext is labeled with a set of descriptive attributes, and each private key is associated with an access policy that specifies which type of ciphertexts the key can decrypt. In CP-ABE, the access policy is specified in ciphertext and the private key is associated with a set of attributes. In this paper, we will utilize CP-ABE for policy-compliance checking, and thus introduce its main primitives below.

- $\mathsf{Setup}(\lambda)$: The setup algorithm takes as input a security parameter λ, and outputs (pk, msk), where pk denotes the public key and msk denotes the master secret key of ABE system.
- $\mathsf{KeyGen}(\omega, msk)$: The key generation algorithm takes as input an attribute set ω and the master secret key msk, and outputs the decryption key dk_ω.
- $\mathsf{Enc}(m, \mathcal{P})$: The encryption algorithm takes as input a message m and the policy $\mathcal{P}$, and outputs the ciphertext $[ct]_\mathcal{P}$ with respect to access policy $\mathcal{P}$.
- $\mathsf{Dec}([ct]_\mathcal{P}, dk_\omega)$: The decryption algorithm takes as input a ciphertext $[ct]_\mathcal{P}$ which was assumed to be encrypted under a policy $\mathcal{P}$ and the decryption key dk_ω for attribute set ω, and outputs the original message m if and only if ω satisfies $\mathcal{P}$.

3.2 Identity-Based Aggregate Signature

An aggregate signature is a single short string that convinces a verifier that a set of n messages are signed by n distinct signers [8]. In this paper, we will utilize a special line of aggregate signature, namely identity-based aggregate signature, in which users' identities (e.g., email address) are used as their public keys, and thus the verifier only needs a description of who signed what for verification. The algorithms of identity-based aggregated signature are described as follows.

- $\mathsf{Setup}(\lambda)$: The setup algorithm takes as input a security parameter λ, and outputs (pk, msk), where pk denotes the public key and msk denotes the master secret key of identity-based aggregate signature.
- $\mathsf{KeyGen}(id, msk)$: The key generation algorithm takes as input a descriptive identity id and the master secret key msk, and outputs the signing key sk_{id}.
- $\mathsf{Sign}(m, sk_{id})$: The signing algorithm takes as input a message m and the signing key sk_{id}, and outputs the signature $[\sigma]_{id}$.
- $\mathsf{Agg}([\sigma]_{S_1}, S_1, [\sigma]_{S_2}, S_2)$: The aggregate algorithm takes as input two sets of identity-message pairs S_1 and S_2, and two identity-based (aggregate) signatures $[\sigma]_{S_1}$ and $[\sigma]_{S_2}$ on the identity-message pairs contained in sets S_1 and S_2 respectively; if $\mathsf{Ver}([\sigma]_{S_1}, S_1) = 1$ and $\mathsf{Ver}([\sigma]_{S_2}, S_2) = 1$, this algorithm outputs the signature $[\sigma]_{S_1 \cup S_2}$ on the identity-message pairs in $S_1 \cup S_2$.
- $\mathsf{Ver}([\sigma]_S, S)$: The verification algorithm takes as input - the (aggregate) signature $[\sigma]_S$ and a description of the identity-message pairs in S, and outputs 1 if and only if $[\sigma]_S$ could be a valid signature output from Sign or Agg for S.

4 Design Goals and Threat Model

Our overarching goal is to guarantee policy compliant search, where policies can be specified by means of a set of conditions against the relaying nodes. Our specific objectives to accomplish this goal are outlined as follows.

(1) *Verifiable Search Compliance*: The main design goal of this work is to provide a mechanism to *verify* that a search query in a CDN is forwarded in compliance with the requester's preferences. These routing preferences are defined over the nodes' attributes by means of policies, similar to conventional policy-based routing. Note that we do not aim to define a new way of performing policy-routing. We assume the existence of a policy-compliant routing scheme such as [11]. We aim to provide an effective mechanism to verify that policy routing is carried out correctly. (2) *Non-repudiable Search Compliance*: we would like to ensure that if a node is involved in a search query, it cannot deny having received the query. (3) *Cost-effective*: the modifications and overhead for providing verifiable policy routing should not represent a major additional cost to conventional routing, nor should they alter the way either routing or caching operate.

Our approach to meet these objectives is based on the following threat model and assumptions. We assume that the network is static. Nodes have knowledge of their direct neighbors, but may not know any peers beyond their first degree neighbors. Each node in the network is globally identifiable, and initially assigned with its identity-based secret key sk_{id} and attribute-based decryption key dk_{prof}. Nodes find resources by forwarding requests through distributed search protocols [1], wherein a resource request is evaluated by a receiving node and either satisfied or relayed to the neighbor node in search of a node able to provide the requested resource. Precisely, we assume that only nodes with certain properties, indicated in a policy by the node originating the request, are asked and allowed to forward the resource requests. We assume that the majority of the nodes are semi-honest. That is, the nodes keep their individual identifiable information

(e.g., sk_{id} and dk_{prof}) away from other nodes to avoid the leakage of private information. Malicious nodes may not adhere to the policy-compliant search protocol, and may send the search query to nodes which do not satisfy the requester's policy.

5 Search Query and Policy Compliant Search

5.1 Bloom Filters of the Nodes and Search Queries

The resources available in the peer network $\mathcal{G}$ are described by means of attributes storing their main features, and are categorized based on their content type (e.g. media files, services, etc.).[1]

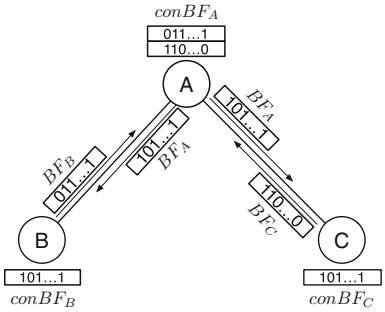


Fig. 1. Creation of concatenated Bloom Filter ($conBFs$)

We assume that resources and attributes of each node are encoded using bloom filter (BF) by the node itself. More precisely for each attribute (say fee) its corresponding value is encoded by means of a single order preserving hash function [6,21] h agreed upon by the network. The hash function generates an index value for the attribute value, where a 1 is placed in the corresponding position in the filter. For instance, assuming $fee = 100$ and $h(100) = 7$, the corresponding attribute filter should have the 7th element set to 1, whereas the remaining elements are set to 0. Since the profile of a node consists of a set of attributes, a bloom filter associated with a profile is generated as the concatenation of attribute filters encoding all attribute values, in a known order. Note that by using an order preserving hash function, there is no need of using multiple hash functions to represent the attributes as there would not be any collisions among the index values.

We also assume that all the nodes are aware of the neighbor nodes' attributes. Initially, all nodes send their local BF to their neighbor nodes, such that each

[1] Examples of categories in T are Standard Industrial Classification (SIC), or the North American Industry Classification System (NAICS).

node is able to maintain a concatenated bloom filter ($conBF$) to keep track of its neighbor's information. A $conBF$ consists of several layers, each of which is a BF corresponding to a neighbor node, and the BF consists of the attribute information of is neighbors. Figure 1 shows an example of how a concatenated filter, $conBF$, may be created. The two pairs of nodes A and B, A and C exchange their local attribute BF with each other, and finally three $conBFs$ are respectively built at these nodes.

Bloom filters aid in routing a query in a policy-compliant manner, which will be discussed in the next section. Precisely, a search query is specified in terms of the requested resource categories, and possible attribute conditions against the service attributes.

Definition 1 (Search Query). *A Search Query (SQ) is an expression of the form:* $SQ = (\{c_1, \ldots, c_n\}; \{a_1 \Theta_1 v_1, \ldots, a_m \Theta_m v_m\})$, *where* $\{c_1, \ldots, c_n\} \in T$ *is a set of resource categories, and* $\{a_1, \ldots, a_m\}$ *is a set of resource attributes in A,* $\Theta_j \in \{<, >, =, \geq, \leq\}$, *and* $\{v_1, \ldots, v_m\}$ *is the set of attribute values.*

Example 1. Let the requester specify the following search query, $SQ = (\{Weather\}, \{Fee < \$100, ExecTime < 20s\})$. The requester is looking for a resource belonging to the $Weather$ category, and the querying of this resource should charge the requester less than 100 dollars fee and should execute in less than 20 s.

As introduced in Sect. 4, given a service search query SQ, we aim to verify that it is routed in the peer network only through *compliant* nodes. Compliant nodes are the nodes that meet the conditions of the search policy (denoted as SP), and therefore are involved in the resource discovery process. For the purpose of this work we consider search policies defined against possible attributes of the routing nodes themselves (e.g. support for certain services, domain, etc.), rather than on the conditions for routing itself (e.g. minimal search path etc.). To verify compliance, we model the profile $prof$ of each node by a set of attributes describing its features, related to security and privacy, routing.

For simplicity, an SP is defined as a combination of atomic Boolean conditions (or Node Criteria), although a more sophisticated definition could also be supported. Before formally introducing SP, we define *node criteria*, as follows.

Definition 2 (Node Criteria (NCriteria)). *A* node criteria *is defined as a combination of clauses in disjunctive normal form* $c_1 \vee \cdots \vee c_n$, *where each* c_j *is an atomic clause, denoting a single or a conjunction of conditions* $c_j = cond_1 \wedge .. \wedge cond_j$, *and each* $cond_i, i \in [1, j]$ *is of the form:* `ATT OP value` *where: (1)* `ATT` *is an attribute, (2)* `OP` *(e.g., =,≥, ≤) is a matching operator; (3)* `value` *is the node preferred value for* `ATT`, *and can be a constant or a variable.*

5.2 Search Policies and Policy Compliant Search

Having defined search queries, we are now ready to formalize search policy.

Definition 3 (Search Policy). *Given a search query SQ, a search policy SP is defined as a couple* $(NCriteria, NHop)$*, where: NCriteria is the node criteria specified according to Definition 2; NHop can take either a value* $n, n \geq 0$*, or it can be set to* $*$*. NHop denotes the maximum number of intermediate nodes, that SQ will be allowed to traverse per a possible path, whose profile satisfies NCriteria.* $NHop = *$ *denotes that no restrictions are placed on the hop count.*

Given a node n and a search policy $SP = (NCriteria, NHop)$ specified by a requester r, a node is compliant if $NCriteria$ is satisfied by the profile $(prof_n)$ of n and the number of hops transmitted from r to n is not more than $NHop$.[2]

A *policy-compliant distributed search* is simply defined as a list of connected nodes satisfying SP.

Definition 4 (Policy-Compliant Distributed Search). *Let* $G =< N, E >$ *be a network, and* (SP, SQ) *be a pair of search policy and query specified by a requester node r. Suppose there is a (cycle-free) sequence of connected nodes* $\overline{Path} = \{r, n_1, \ldots, n_k = d\}$ *in G connecting r with a node d able to resolve the query SQ. If every node* $n_i \in \overline{Path}$ *satisfies the search policy SP, then* $\overline{Path}$ *distributed search is policy-compliant with respect to SP.*

Note that in the definition above we essentially request a sequence of nodes in the network graph where each node satisfies the policy and that leads to the successful resolution of query. We do not impose any condition against how this path is found or against any other properties of the path itself (if it is an optimal path or if it is minimal). Several path finding algorithms could be used, with no impact on our problem statement.

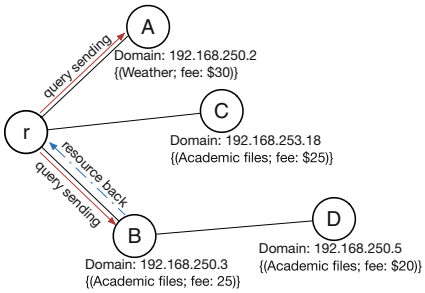


Fig. 2. An example for policy-compliant distributed search

A weaker notion of the above definition, which will be useful for our verification algorithms is defined as α compliance.

Definition 5 (α-Policy-Compliant Distributed Search). *Let the pair* (SP, SQ) *be a search policy and query specified by a requester node r. Let the*

[2] If $NHop = *$, we consider it is infinitely large.

set $\overline{Path}$ *contain all the nodes transmitted during a time of distributed search* (SP, SQ)*. If, for any arbitrarily sampled node* $n \in \overline{Path}$*, the probability that* n *satisfies the search policy* SP *is not less than* α *(*$0 < \alpha \leq 1$*), then* $\overline{Path}$ *is* α*-policy-compliant.*

An example of policy-compliant distributed search is given below.

Example 2. Assume a P2P network is organized as in Fig. 2. A requester node (denoted as r in Fig. 2) sends a query asking for academic files. r requests that the files do not cost more than \$25. Accordingly, the search query is formalized as $SQ = (\{Academic files\}, \{fee \leq \$25\})$. Moreover, r requests the search to be carried out only within its local area network and hence, defining the following node criteria or policy: $NCriteria = \{(Domain = 192.168.250.X)\}$. This policy indicates that the search is required to be performed within a subnet, the IP address of which ranges from 191.168.250.1 to 192.168.250.255, and also restricts the search zone to be its direct neighbor nodes ($NHop$ is set to be 1). The corresponding distributed search for this request is shown in Fig. 2. It is clear that this query should not be transmitted to node C or node D, because the former is in a different domain from the one specified in $NCriteria$, while the latter violates the $NHop$ restriction.

6 Malicious Node Detection

We now describe our routing compliance verification mechanism. Our solution includes two main phases: *resource discovery*, and *compliance checking* phase. During the resource discovery phase, the requester propagates a pair (SP, SQ) of search policy and query to discover the resources satisfying SQ, while restricting the query routing only through the nodes satisfying SP. Upon resolving the query, the discovered resource R, as well as a path proof PF, are returned back to requester. In the compliance checking phase, the requester takes as input PF and verifies the policy-compliance of the returned search path.

6.1 Resource Discovery

We now describe the resource discovery phase of our policy verification mechanism.

Suppose a requester issues a search query $SQ = (\{c_1, \ldots, c_n\}; \{a_1 \Theta_1 v_1, \ldots, a_m \Theta_m v_m\})$ (see Definition 1), simultaneously restricting the search query propagation path to be controlled by a search policy $SP = (NCriteria, NHop)$ (see Definition 3). The following steps are executed.

Assume an exhaustive search across the network is enforced, where the query is forwarded to all suitable nodes. The requester r firstly evaluates the node criteria on all of its neighbor nodes. Let us consider each individual atomic condition $cond_i$ in search policy SP. Recall that $cond_i$ is in the form of `Att OP Value` (see Definition 2), and h is the hash function used for encoding the attribute `Att` in the bloom filter. The requester computes the index $index_{cond_i}$ of this

atomic condition by hashing $h(\texttt{Value})$. For example, for partly bounded conditions (e.g. $\texttt{A} < 1$), the upper or lower bound of the condition are hashed (e.g. $h(\texttt{10})$). The computed $index_{cond_i}$ is then compared with the positions of the non-full index values of bloom filter in $conBF_r$. This condition $cond_i$ is fully satisfied by a bloom filter BF in $conBF_r$, when (1) BF in the exact position equal to $index_{cond_i}$ have 1, in case of equality conditions ($\texttt{Att} = \texttt{value}$); (2) or BF in the positions anywhere before or after $index_{cond}$ have 1s, in case of partly bounded conditions ($\texttt{Att} \geq / \leq \texttt{value}$).

For every node (we say n_j) satisfying BFs, the requester stores a *hop item* to collect three pieces of information: the identity of the previous node ($\perp$ for requester), id_{prev}, the identity of current hop id_{cur} and the identity of next hop id_{nxt}. Each of the hop items is signed under the requester's signing key sk_r, and encapsulated into a hop list L. The hop list is then to be passed along with the search query and updated and signed by each node, such that the requester can finally verify the authenticity of the propagation path.

During resource discovery, every node n_j receives (SQ, SP) as well as $(L = \langle e_{r\cdot}, \ldots, e_{ij} \rangle, \sigma)$, where L is the hop list consisting of the hop items the search query has transmitted by and σ is the signature aggregated on these hop items. More precisely, suppose n_j receives this data from a previous node n_i (n_i could be the requester n_r). n_j firstly verifies the authenticity of L. This is achieved by completing two operations.

1. n_j picks out the last item $e_{ij} = (\cdot, id'_i, id'_j)$ of L and checks whether $id'_i = id_i$ and $id'_j = id_j$, to guarantee that the search query propagates in a authentic way *at this hop from* n_i *to* n_j;
2. n_j checks the validity of signature σ on messages $e_{r\cdot}, \ldots, e_{ij}$ and identities $id_r, \ldots, id_i$ to guarantee this search query propagated correctly *in all of the previous hops.*

Operation (1) guarantees that n_i honestly sends query following the hop information recorded in L, while (2) guarantees that none of the faked hop items exists in previous propagation path. If either of the checks fails, an error is reported and the search for resource is aborted.

Node n_j then checks the satisfaction of local resources and neighbor node criteria based on bloom filter, using the same approach described above for node criteria evaluation. One of the following two cases could arise:

- Case 1. If a query-satisfying resource is found locally by n_j or none of the neighbor nodes satisfies the search policy, the search is over. A new hop item $e_{j\perp} = (id_i, id_j, \perp)$ is generated to indicate "end hop", and signed using n_j's signing key sk_j. The signature (on $e_{j\perp}$) is then aggregated with the previous aggregated signature σ to generate a new version of σ. Finally, after appending $e_{j\perp}$ with L, the authenticated path (L, σ) is sent back to the requester (either traversing backward through the whole path or directly, depending on the specific query resolution algorithm being adopted).
- Case 2. Otherwise, there must exist at least one neighboring node satisfying SP. For every satisfying neighbor node (we say n_k), (L, σ) is replicated, and

another hop item $e_{jk} = (id_i, id_j, id_k)$ is generated and appended with the new copy of L, to indicate that next hop is n_k. Similar to the first case, after signing e_{jk} and aggregating the new signature into (the copy) σ, the updated (L, σ) is then sent to node n_k, along with the query-policy pair (SQ, SP).

Note that, although we present it for the case of exhaustive search, our scheme can be easily adapted to support any routing protocol (e.g. random walk). As compared to existing protocols, in our scheme, the requester is able to restrict the query to be forwarded only through certain nodes by defining a policy over the query.

Example 3. Figure 3 shows a toy example for the process of resource discovery. Two neighbor nodes (n_1 and n_3) are respectively sent the resource query from n_r. For the node n_3, a satisfying resource is found locally, and returned back to requester along with the authenticated path. For the node n_1, it forwards the query to a next policy satisfying node n_2, which does not have any policy satisfying neighbor nodes. So, another path of authenticated nodes is sent back to the requester following this path: $n_2 \rightarrow n_1 \rightarrow n_r$.

6.2 Compliance Checking

Upon receiving the authenticated path, the requester starts to check whether it is also policy compliant. Verifying policy compliance is a two-step process. The first step consists of checking path authenticity. The requester examines the hop list L, in specific, whether the concatenation of nodes is correct. For example, for any two continuous hop items (we say e_{ij} and e_{jk}), the requester checks whether id_{cur} in e_{ij} equals id_{prev} in e_{jk} and whether id_{nxt} in e_{ij} equals id_{cur}. Then, it verifies the validity of the aggregated signature σ using all the identities stored in the current node entry of hop items in L. This is achieved by examining whether σ is a valid aggregated signature on a series of messages $e_{r\cdot}, \ldots, e_{jk}, e_{k\perp}$ by the public identities $id_r, \ldots, id_j, id_k$, where id_i $(i = r, \ldots, j, k)$ is the identity of node generating and signing the hop item $e_{i\cdot}$. If either of the verification steps fails, an error is reported.

The second step is to check whether the authentic path is policy compliant. The step is of course necessary as some nodes may have passed the message along without meeting the policy conditions. Generally, our algorithm is based

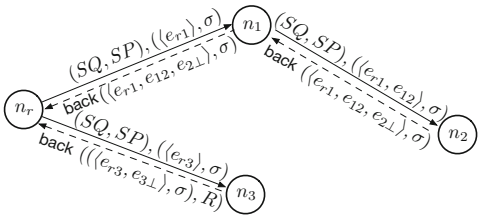


Fig. 3. Resource discovery

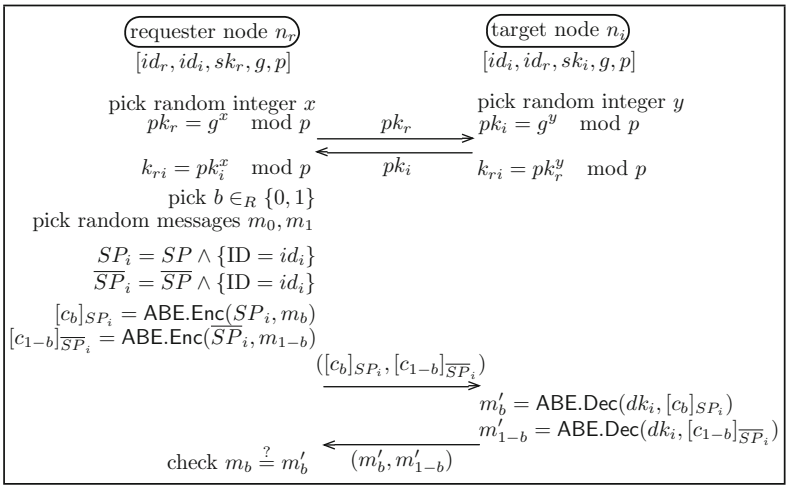


Fig. 4. Protocol for checking policy satisfaction of node in iterative model (the group generator g and big prime p are predefined as system parameters)

on examining the policy satisfiability of nodes in the propagation path using attribute-based encryption, and follows either the iterative model or the non-iterative model. Suppose the requester wants to check the satisfiability of node n_i.

Iterative Model. The algorithm in iterative model shown in Fig. 4 is executed. Firstly, the requester is to establish a secure communication channel with the target node n_i following the well known Diffie-Hellman key exchange protocol. Notice that to avoid the man-in-the-middle attacks, the exchange step needs to be cryptographically bounded. To this end, rather than trivially exchange DH public keys, both nodes append their identity-based signatures along with their DH public keys (omitted in Fig. 4 for simplicity), therefore preventing adversaries from faking public keys and eavesdropping the shared session key k_{ri}.

After establishing a secure channel with n_i, the requester generates two random messages m_0 and m_1, and picks a random bit $b \in_R \{0, 1\}$ for encrypting m_0, m_1 respectively, under the hybrid policy SP_i and its complementary $\overline{SP_i}$ using attribute-based encryption. Specifically, after building $SP_i = SP \wedge \{\text{ID} = id_i\}$, where $\wedge$ is an AND gate connecting SP and an atomic policy $\{\text{ID} = id_i\}$, the requester encrypts m_b under SP for obtaining $[c_b]_{SP_i}$, while m_{1-b} under $\overline{SP_i}$ for obtaining $[c_{1-b}]_{\overline{SP_i}}$. $[c_b]_{SP_i}$ and $[c_{1-b}]_{\overline{SP_i}}$ are then sent to n_i for decryption. The target node n_i tries to decrypt *both* the ciphertexts using its decryption key dk_i and feeds back the messages (m'_b, m'_{1-b}). The requester checks whether $m_b = m'_b$, if not, a policy-violating routing is reported. The reason to build a hybrid policy SP_i (binding the target node's identity with test policy) is to prevent collusion among nodes, in which a policy satisfying node could lend its decryption key to the target node to help it pass the test. In other words, under the hybrid policy, even if other satisfying nodes share their decryption keys with the target node, the checking cannot be passed, because the lent decryption keys

from other nodes do not satisfy the binding policy ID = id_i, and fail to decrypt the test ciphertext.

Non-Iterative Model. In the verification algorithm above, the requester is able to detect non-compliant nodes in an adaptive manner. In other words, since the requester runs the proposed protocol with the target node one by one, to check the policy compliance of a full path, it is able to know the intermediate checking results at each node, and decide the node to be checked for the next time. For higher efficiency, the requester may prefer to check nodes in a non-iterative manner. Suppose $S = \{id_{i_k}\}_{k=1}^{|S|}$ is the set consisting of all the target nodes to be checked. For each node id_{i_j} $(j = 1, \ldots, n)$ in S, two random messages $(m_0^{(i_j)}, m_1^{(i_j)})$ are generated and encrypted in the same way for m_0 and m_1 as the algorithm of Fig. 4. All the test messages $\{(m_0^{(x)}, m_1^{(x)})\}_{id_x \in S}$ are then propagated along with the set S in the checking path. Each node tries to decrypt the pair of encrypted messages if its identity is in S. Finally, the decrypted messages are sent back to the requester for final decision. Each decrypted message can be optionally signed by the nodes in the path to avoid modifications to the decrypted messages by other nodes in the path. Through this model, the requester is only required to be online when preparing test messages and when checking the decrypted results, reducing the computational overhead significantly.

Example 4. Let us re-consider Example 3. Suppose the path delivered back to requester is $\{n_r, n_1, n_2\}$, and the requester n_r tries to verify the policy compliance of n_1 and n_2. To this end, n_r generates two pairs of random messages $(m_0^{(1)}, m_1^{(1)})$ and $(m_0^{(2)}, m_1^{(2)})$, and for each pair $(m_0^{(i)}, m_1^{(i)})$ (where $i = 1, 2$) respectively encrypts $m_0^{(i)}$ and $m_1^{(i)}$ under the policy $SP \wedge \{\text{ID} = n_i\}$ and $\overline{SP} \wedge \{\text{ID} = n_i\}$. In the iterative model of the compliance checking protocol, requester sends $C_1 = ([c_0^{(1)}]_{SP \wedge \{\text{ID}=n_1\}}, [c_1^{(1)}]_{\overline{SP} \wedge \{\text{ID}=n_1\}})$ to n_1 for decryption, and if the decrypted ciphertext $m_1^{(1)\prime}$ does not equal $m_1^{(1)}$, an unsatisfying node is reported; otherwise requester continues to test n_2 using $C_2 = ([c_0^{(2)}]_{SP \wedge \{\text{ID}=n_2\}}, [c_1^{(2)}]_{\overline{SP} \wedge \{\text{ID}=n_2\}}))$ in the same way. In the non-iterative model mode of this protocol, the requester sends the two pairs (i.e., C_1 and C_2) of ciphertext to n_1 at once, which tries to decrypt the pair (i.e., C_1) of ciphertext intended for it and forwards the other pair (i.e., C_2) to n_2 for decryption. The response (i.e., decryptions of C_2 and C_1) is then sent back to requester for final decision along the path from n_2, n_1 to n_r.

7 Practical and Security Considerations

Recall our proposed two-phased method works by identifying the correct path of search query propagation and then checking the policy satisfiability of all the nodes in the path. The method is secure if the path is correctly identified at first. In spite of this, our approach suffers from two shortcomings. First, it places a computational burden on the requester for testing the policy satisfiability of

the nodes in the path. Second, our method relies on the correctness of the query propagation path, resulting that there might exist a few potential attacks aiming at breaking our method through faking a cheating path. In this section, we consider some practical methods to address both shortcomings.

7.1 Determining the Number of Nodes to Verify

In this subsection, we analyze the number of nodes the requester should check to achieve α-compliance (i.e., at least a percentage α of the nodes in path satisfies the policy), for both cases.

We model our problem as follows. Suppose a query SQ has been resolved by a given *Path*, where $|\texttt{Path}| = n$ indicates that n unique nodes were involved during this search. Assume that an arbitrary number of nodes m ($m < n$) in the path does not satisfy our policy requirement. Our aim is to estimate the number of nodes to be checked to detect this dishonest behavior with a confidence greater than α[3]. Note that our detection model follows the "once for all" philosophy. That is, if only one non-compliant node is found, we consider the full search dishonest.

Non-iterative Probabilistic Model. In the first verification model, discussed in Sect. 6.2, we assume that the requester generates all the target nodes (constitute the target set `target`) to check at once. The requester does not obtain any feedback about the intermediate checking results before it generates all the target nodes. Suppose the number of nodes to be checked is x (i.e., $|\texttt{target}| = x$ in this case)[4].

Given known values of n and m, we can compute the minimum value of x by resolving the following inequality which contains only x as an unknown value.

$$1 - \frac{\binom{n-m}{x}}{\binom{n}{x}} \geq \alpha \Rightarrow 1 - \frac{(n-m)!(n-x)!}{n!(n-m-x)!} \geq \alpha \tag{1}$$

The equation is easily understood. Our problem consists of selecting x nodes at once from n path nodes to be checked, with $\binom{n}{x}$ possibilities. Assume the x target nodes to check are all selected from the $n-m$ satisfying nodes, which has $\binom{n-m}{x}$ possibilities.

Then, we can compute the probability of not detecting a dishonest node by randomly checking x nodes as $\frac{\binom{n-m}{x}}{\binom{n}{x}}$. Thus, $1 - \frac{\binom{n-m}{x}}{\binom{n}{x}}$ is the probability of detecting any non-compliant node by checking x nodes.

Iterative Probabilistic Model. In the iterative probabilistic model presented in Sect. 6.2, we assume that the requester is able to adaptively generate the target node to be checked. Since in this scheme, the requester knows the intermediate results obtained from previous checks, it can decide accordingly which nodes are

[3] The symbol α is abused here to denote the confidence threshold in dishonesty detection.

[4] We need to restrict that $x \leq n - m$ in our models. This is because, we can always detect non-compliant nodes if we test more than $n - m$ nodes.

to be checked. Suppose the number of nodes to be checked is x. Suppose A_k is the probability for detecting dishonesty by checking k nodes in `Path`. It is clear that $A_1 = \frac{m}{n}$ and

$$A_k = (1 - \sum_{i=1}^{k-1} A_i)\frac{m}{n-k+1} \tag{2}$$

In what follows we explain the above equation. Since the requester can adaptively generate the target node, the probability of selecting the satisfying node is not identical each time a check is performed. For example, if the requester successfully selects a non-compliant node the first time, and detects a dishonest node, then $A_1 = \frac{m}{n}$. The next time, in the adaptive case, the probability of catching a non-compliant node becomes $\frac{m}{n-1}$, because one satisfying node has been verified already, and should be removed for all the subsequent selections. Thus, the probability $\frac{m}{n-1}$ holds in the case that the non-compliant node is not caught in the first time, having probability $1 - A_1$.

Accordingly, the probability of catching non-compliant nodes in the second check is computed as $A_2 = (1 - A_1)\frac{m}{n-1}$. Recursively, for A_k, $(1 - \sum_{i=1}^{k-1} A_i)$ is the probability that any non-compliant node is not caught in the first $k-1$ times of checking. At the kth round, $k-1$ satisfying nodes are removed due to the inability of catching non-compliant nodes in the first $k-1$ times, and thus the probability of catching a dishonest node for the k th time is $\frac{m}{n-k+1}$. Finally, we can get the probability $A_k = (1 - \sum_{i=1}^{k-1} A_i)\frac{m}{n-k+1}$.

We are to solve the following inequality with respect to unknown x:

$$\sum_{i=1}^{x} A_i \geq \alpha \tag{3}$$

Interestingly, although our proposed non-iterative and iterative models work in a different manner, they achieve the same probability of catching dishonest nodes, assuming they check the same number of nodes. This finding can be demonstrated by solving the general formula (2) and comparing the result $\sum_{i=1}^{x} A_i$ with the probability of non-iterative model (i.e., the left part of inequality (1)). In what follows, we provide a detailed proof that the left part of Eq. (3) equals to the left part of Eq. (1). That is, for a fixed value of x, $\sum_{i=1}^{x} A_i = 1 - \frac{(n-m)!(n-x)}{n!(n-m-x)!}$ where $A_k = (1 - \sum_{i=1}^{k-1} A_i)\frac{m}{n-k+1}$ for $k = 2, 3, \ldots, x$.

Without loss of generality, we denote $prob_{noniter a}(k) = 1 - \frac{(n-m)!(n-k)!}{n!(n-m-k)!}$ indicating the probability of catching dishonest nodes in the non-iterative model when checking x nodes. Similarly, $prob_{itera}(x) = \sum_{i=1}^{x} A_i$ is the probability of catching dishonest nodes in the interactive model. It is clear that in the iterative model $A_x = prob_{itera}(x) - prob_{itera}(x-1)$, and we substitute this expression into Eq. (2) to obtain

$$\begin{aligned} prob_{itera}(k) - prob_{itera}(k-1) &= (1 - prob_{itera}(k-1))\frac{m}{n-k+1} \\ 1 - prob_{itera}(k) &= (1 - \frac{m}{n-k+1})(1 - prob_{itera}(k-1)) \end{aligned} \tag{4}$$

Then, our aim is to recursively solve the Eq. (4) to obtain $prob_{itera}(k)$, with the condition that $prob_{itera}(1) = A_1 = \frac{m}{n}$. To this end, we iterate the variable k in Eq. (4) from k down to 2 to get a series of $k-1$ equations as follows.

$$1 - prob_{itera}(k) = (1 - \frac{m}{n-k+1})(1 - prob_{itera}(k-1))$$
$$\dots \qquad \dots$$
$$1 - prob_{itera}(2) = (1 - \frac{m}{n-2+1})(1 - prob_{itera}(1))$$

We then multiply these $k-1$ equations together to get

$$1 - prob_{itera}(k) = (1 - prob_{itera}(1)) \times \frac{\prod_{i=2}^{k}(n-i+1-m)}{\prod_{i=2}^{k}(n-i+1)} \tag{5}$$

It is clear that $\prod_{i=2}^{k}(n-i+1-m) = (n-m-k+1)\dots(n-m-1) = \frac{(n-m-1)!}{(n-m-k)!}$ and $\prod_{i=2}^{k}(n-i+1) = (n-k+1)\dots(n-1) = \frac{(n-1)!}{(n-k)!}$. We further substitute both equations as well as $prob_{itera}(1) = \frac{m}{n}$ into Eq. (5).

$$1 - prob_{itera}(k) = \frac{n-m}{n}\frac{(n-m-1)!(n-k)!}{(n-1)!(n-m-k)!}$$
$$prob_{itera}(k) = 1 - \frac{(n-m)!(n-k)!}{n!(n-m-k)!} = prob_{nonitera}(k)$$

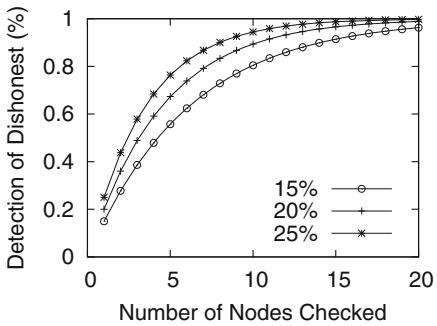


Fig. 5. Probability estimation of dishonest detection in non-iterative and iterative models

In Fig. 5, we provide some numerical examples about probability of detecting dishonest nodes proportional to the number of nodes to be verified. In this numerical example, the path consists of 100 loop-free nodes. We present three cases wherein we assume there are respectively 15 %, 20 % and 25 % nodes in the path that do not satisfy the requester's policy, and show the confidence

of detecting dishonest nodes. It is clear from Fig. 5 that, in order to achieve a detection confidence of 0.9, we only need to check a small set of nodes in the path. Precisely, the requester will need to check 8, 11 and 15 nodes in the 15 %, 20 % and 25 % case for ensuring 0.9 confidence. Only a subset of the nodes in the path are to be verified for high confidence results, and we can check only part of the nodes in the path to save computing and networking resources.

7.2 Attacks and Countermeasures

There are a number of potential attacks against our scheme. In this subsection, we outline two of the most common attacks, along with some potential countermeasures.

- In the first potential attack, since the metadata (i.e., the hop list L and aggregated signature σ) in resource discovery phase would be transferred back to the requester, a malicious node could record this information, and use it for launching replay attack in the future. For example, suppose a requester requests for resources, with the same policy twice. Since the policy is the same, it would follow the same path for the both times of search. A malicious node could record the hop list as well as the aggregated signature returned back in the first time, and use it for cheating the next time of search. Specifically, in the second time of search, even if the malicious node does not forward the query to the policy satisfying neighbor node, it can send back the recorded hop list and aggregated signature in the first time to cheat that it has forwarded the query in the correct way. A simple countermeasure to this attack is to append another time entry in the hop item in L and ask each node to sign on the hop item including not only previous, current and next node, but also a time period to distinguish the signatures for two times of search. In this way, during verification, the requester can easily detect the old metadata and catch the dishonest node.
- The second potential attack originates from the fact that a malicious node is lazy, which does not forward the search query to satisfying neighbor node and cheat that none of the neighbor nodes satisfy the policy. Suppose A is a lazy node adjacent to the requester r, and we can detect this lazy node in the following ways. The requester node r compares the policy with the BF received from a neighbor node A, and notes down the value x that lies in the corresponding positions related to the policy. This value gives the number of A's neighbor nodes satisfying the search policy. The requester expects to receive x aggregated signatures from its neighbor A. If it did not receive at least x aggregated signatures (and an exhaustive search was implemented), then it concludes that A is lazy or that it has dropped SQ[5].
- In the third potential attack, a malicious node (say A), upon receiving a search query, could cheat that none of the neighbor nodes satisfies the policy

[5] If a non-exhaustive search algorithm is used, the requestor would expect at least k responses, where k is to be determined according to the routing scheme employed by the network.

and return back the updated (L, σ) to requester, but forwards the search query to a policy unsatisfying neighbor node (say B), which will drop the forwarding of the query and/or does not send the aggregated signature to the requester. We point out that, this attack is challenging to be detected, since the malicious nodes (A and B) are adjacent. In this case, some additional controls are needed, in addition to the scheme discussed in this paper. A simple approach to fully prevent the policy unsatisfying nodes from accessing the search query, is for the requester to encrypt the content of the query using attribute-based encryption, such that only the policy satisfying nodes are able to decrypt and access the query. In this way, even if a policy unsatisfying node receives the search query, it is not able to learn the content of query.

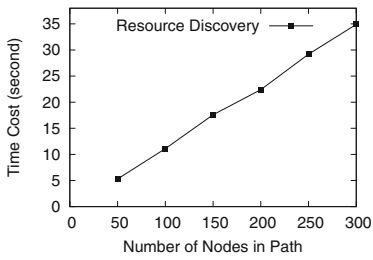
8 Experimental Analysis

We conducted our experiments on an Intel core i7 CPU @2.00 GHz, 8 GB RAM, Ubuntu machine. In these experiments, we are mainly concerned with the computational times of our protocols rather than the network delays involved. Hence, our experiments do not reflect network communication delays among the nodes in the network. We conduct our experiment on a peer to peer network topology whose structure is obtained from http://snap.stanford.edu/data/. The network consists of 10,000 + nodes.

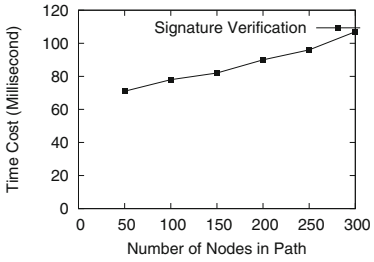
Our first experiment involves testing for the computational times of the first step of our protocol, that is, secure proof of identities of the path of the search query (see Sect. 6.1). This experiment has two parts to it. The first part measures the computational times for the aggregated signature and search query traversal through the network. The second part measures the computational times for the verification by the requester, of the aggregated signatures of the paths that the search query had taken.

First, we vary the path length traversed by the search query and observe the respective computational times. Path length is the number of nodes traversed by the search query in a path. From the graph in Fig. 6(a), we observe that as the path length increases, the time for computing the aggregated signatures increases. Next, in the second part, we vary the path length and observe the respective computational times. Interestingly, from the graph in Fig. 6(b), we observe that even though as the number of nodes in a path increases, the time to verify the aggregated signatures increases very negligibly, in the order of milliseconds. This confirms that using aggregated signatures for secure proof of identities of a path is efficient when compared to sending individual signatures by each node in the path to the requester. This is because as the number of nodes increases in a path, the number of individual signatures to verify will increase for the verifier. Hence, receiving individual signatures from every node would drastically increase the communication overhead of the protocol.

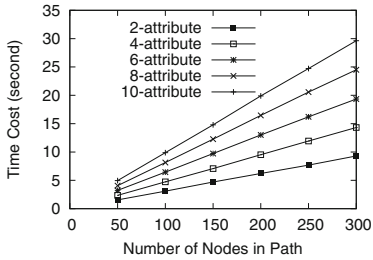
Our second set of experiments test the policy compliance of the nodes in the paths taken by the search query, that is, to test the phase where the requester uses attribute based encryption. First, we compute the computational times



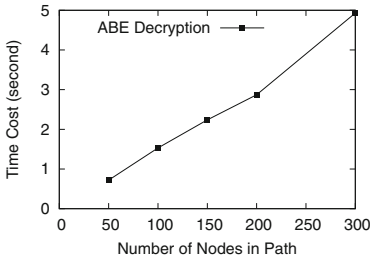
(a) Resource discovery



(b) Verification of correctness of the path taken



(c) Verification of policy compliance: ABE encryption in a path



(d) Verification of policy compliance: ABE decryption in a path

Fig. 6. Computational times of steps

of the encryption of messages performed by the requester or the owner of the query for each node in a path taken by the query. If there are n nodes in a path, then the requester encrypts n messages with the ABE protocol. We compute the computational times by varying the number of nodes in a path, and also we perform the same experiment for different number of attributes in the encryption policy of the requester. From the graph in Fig. 6(c), we observe that as the number of nodes in a path increases, the computational time linearly increases for encrypting the messages with ABE. We also observe that, as the number of attributes in a policy increases, the computational time linearly increases. Next, we also compute the times of the decryption of messages by all the nodes in a path. That is, in this experiment, each node in the path sequentially decrypts the message encrypted by the requester for the node, with the requester's policy. We compute the computational times by varying the number of nodes in a path. From the graph in Fig. 6(d), we observe that the time for all the nodes in a path to decrypt the ABE message linearly increases as the number of nodes in a path increases. In this experiment, the number of attributes in the policy does not affect the computational time for decrypting the message, as for decryption, each node uses its own private key to decrypt the message, and the private key is not associated with the number of attributes in the encryption policy.

9 Conclusion

In this paper, we posit that search queries are critical in such content dissemination networks, as eventually these queries lead to the discovery of the desired content. The importance of search queries, requires us to develop security mechanisms to ensure that the queries are appropriately forwarded based on the needs and policies of the query owner. We propose an effective and efficient protocol to detect malicious nodes that do not comply with the forwarding protocols established in the network. In addition to this, our protocol also aims to protect the integrity of the proof of various paths taken by a search query through the network. In the future, we aim to efficiently address the collusion problem such that the requester is able to verify the policy compliance by just preparing a single ABE encrypted message instead of a multiple encrypted messages equal to the number of nodes in the path.

Acknowledgement. Portion of the work from Dr. Squicciarini was funded under the auspices of National Science Foundation, Grant #1250319.

References

1. Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. ACM Comput. Surv. (CSUR) **36**(4), 335–371 (2004)
2. Arai, B., Das, G., Gunopulos, D., Kalogeraki, V.: Efficient approximate query processing in peer-to-peer networks. IEEE Trans. Knowl. Data Eng. **19**(7), 919–933 (2007)
3. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: SP 2007: Proceedings of the 2007 IEEE Symposium on Security and Privacy, pp. 321–334. IEEE Computer Society (2007)
4. Compagno, A., Conti, M., Gasti, P., Tsudik, G.: Poseidon: mitigating interest flooding ddos attacks in named data networking. In: 2013 IEEE 38th Conference on Local Computer Networks (LCN), pp. 630–638, October 2013
5. Durr, M., Maier, M., Wiesner, K.: An analysis of query forwarding strategies for secure and privacy-preserving social networks. In: 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pp. 535–542, August 2012
6. Fox, E.A., Chen, Q.F., Daoud, A.M., Heath, L.S.: Order preserving minimal perfect hash functions and information retrieval. In: Proceedings of the 13th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 279–311. ACM (1990)
7. Gasti, P., Tsudik, G., Uzun, E., Zhang, L.: Dos and ddos in named data networking. In: 2013 22nd International Conference on Computer Communications and Networks (ICCCN), pp. 1–7, July 2013
8. Gentry, C., Ramzan, Z.: Identity-based aggregate signatures. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 257–273. Springer, Heidelberg (2006)
9. Goergen, D., Cholez, T., Fran, J., Engel, T.: Security monitoring for content-centric networking (2012)

10. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 89–98. ACM (2006)
11. Karumanchi, S., Squicciarini, A.C., Carminati, B.: Policy-compliant search query routing for web service discovery in peer to peer networks. In: International Conference on Web-Services, pp. 387–394 (2013)
12. Karumanchi, S., Squicciarini, A., Lin, D.: Selective and confidential message exchange in vehicular ad hoc networks. In: Xu, L., Bertino, E., Mu, Y. (eds.) NSS 2012. LNCS, vol. 7645, pp. 445–461. Springer, Heidelberg (2012)
13. Khan, S., Cholez, T., Engel, T., Lavagno, L.: A key management scheme for content centric networking. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 828–831, May 2013
14. Li, X., Wu, J.: Cluster-based intelligent searching in unstructured peer-to-peer networks. In: 2005 25th IEEE International Conference on Distributed Computing Systems Workshops, pp. 642–645, June 2005
15. Misra, S., Tourani, R., Majd, N.E.: Secure content delivery in information-centric networks: design, implementation, and analyses. In: Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking, pp. 73–78. ACM (2013)
16. Mizrak, A., Cheng, Y.C., Marzullo, K., Savage, S.: Fatih: detecting and isolating malicious routers. In: 2005 Proceedings of International Conference on Dependable Systems and Networks, DSN 2005, pp. 538–547, June 2005
17. Padmanabhan, V.N., Simon, D.R.: Secure traceroute to detect faulty or malicious routing. SIGCOMM Comput. Commun. Rev. **33**(1), 77–82 (2003)
18. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
19. Salmanian, M., Li, M.: Enabling secure and reliable policy-based routing in manets. In: Military Communications Conference - MILCOM 2012, pp. 1–7 (2012)
20. Vishnu, V., Senthilkumar, N.C.: An intelligent approach to query processing in peer to peer networks. Int. J. Comput. Sci. Issues **9**(3), 1–4 (2012)
21. Wang, J., Wang, J., Yu, N., Li, S.: Order preserving hashing for approximate nearest neighbor search. In: Proceedings of the 21st ACM International Conference on Multimedia, pp. 133–142. ACM (2013)
22. Wang, S., Ooi, B.C., Tung, A., Xu, L.: Efficient skyline query processing on peer-to-peer networks. In: 2007 IEEE 23rd International Conference on Data Engineering, ICDE 2007, pp. 1126–1135, April 2007
23. Zhang, X., Chang, K., Xiong, H., Wen, Y., Shi, G., Wang, G.: Towards name-based trust and security for content-centric network. In: 2011 19th IEEE International Conference on Network Protocols (ICNP), pp. 1–6, October 2011