EventGuard: A System Architecture for Securing Publish-Subscribe Networks

Mudhakar Srivatsa[†], Ling Liu[‡] and Arun Iyengar[†]
IBM T.J. Watson Research Center, Yorktown Heights, NY 10562[†]
College of Computing, Georgia Institute of Technology, Atlanta, GA 30332[‡]
{msrivats, aruni}@us.ibm.com, lingliu@cc.gatech.edu

[1] Publish-subscribe (pub-sub) is an emerging paradigm for building a large number of distributed systems. A wide area pub-sub system is usually implemented on an overlay network infrastructure to enable information dissemination from publishers to subscribers. Using an open overlay network raises several security concerns such as: confidentiality and integrity, authentication, authorization and denial-of-service (DoS) attacks. In this paper we present EventGuard − a framework for building secure wide area pub-sub systems. The EventGuard architecture is comprised of three key components: (1) a suite of security guards that can be seamlessly plugged-into a content-based pub-sub system, (2) a scalable key management algorithm to enforce access control on subscribers, and (3) a resilient pub-sub network design that is capable of scalable routing, handling message dropping-based DoS attacks and node failures. The design of EventGuard mechanisms aims at providing security guarantees while maintaining the system's overall simplicity, scalability and performance metrics. We describe an implementation of the EventGuard pub-sub system to show that EventGuard is easily stackable on any content-based pub-sub core. We present detailed experimental results that quantify the overhead of the EventGuard pub-sub system and demonstrate its resilience against various attacks.

Categories and Subject Descriptors: C.2.4 [**Distributed Systems**]: Distributed Applications—*Security and Performance*

General Terms: Security

Additional Key Words and Phrases: Publish-Subscribe System, Access Control, DoS Attacks, Resilient Overlay Network, Performance and Scalability

1. INTRODUCTION

An increasingly large number of Internet applications require information dissemination across different organizational boundaries, heterogeneous platforms, and a large, dynamic population of publishers and subscribers. A publish-subscribe overlay service (hereafter refer to as pub-sub) is a wide-area communication infrastructure that enables data dissemination across potentially unlimited numbers of publishers and subscribers, scattered geographically across the wired and wireless Internet [Carzaniga et al. 2001]. In such an environment, publishers publish information in the form of event notifications and subscribers have the ability to express their interests in an event or a pattern of events by sending subscriptions to the pub-sub overlay network. The pub-sub overlay network uses content-based routing schemes to dynamically match each publication to all the active subscriptions, and notifies the subscribers of any publication that matches their registered interest, ensuring that subscribers only receive notifications of those events that match their interests.

An important characteristic of pub-sub overlay services is the decoupling of publishers and subscribers combined with content-based routing protocols, enabling a many-to-many communication model. Such a model presents many inherent benefits as well as potential risks. On one hand, by offloading the task of identifying destination addresses of publications to the pub-sub overlay network, it not only allows message routing to be handled in a way that avoids unnecessary message replications but also enables dynamic and fine-grained subscriptions. As a result, pub-sub overlay services have proven to be scalable and effective for wide-area information dissemination. On the other hand, many security concerns exist in such an environment regarding authenticity, confidentiality, integrity and availability of publications and subscriptions: such as the confidentiality, integrity and authenticity of subscriptions and publications.

Most research and development of pub-sub systems to date have been largely dedicated to the performance and scalability of pub-sub networks as well as the expressiveness of event publication and subscription models nodes [Carzaniga et al. 2001][Banavar et al. 1999][Datta et al. 2003]. Only recently, a few researchers have studied specific security requirements of pub-sub networks [Wang et al. 2002], pointing out attacks threatening message integrity (unauthorized writes) and authenticity (fake origins) in addition to message confidentiality (unauthorized reads), and the risks of bogus publications and fake subscriptions. Unfortunately, most of the existing secure event distribution protocols proposed so far focus only on the content confidentiality risks in pub-sub networks [Raiciu and Rosenblum 2006][Opyrchal and Prakash 2001]. The lack of a more holistic security framework has been a major hurdle in deploying pub-sub systems for mission-critical applications that could greatly benefit from them.

In this paper, we present *EventGuard* − a framework for securing a pub-sub overlay service. EventGuard simultaneously supports in-network matching and secure content-based routing, but makes careful design choices to tradeoff performance with security. This is achieved by separating event attributes into two types: routable attributes (that are used for in-network matching) and secret attributes (whose confidentiality needs to be guaranteed). For example, the secret attribute `patientRecord` in an event $e = \langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`age`, 25$\rangle$, $\langle$`patientRecord`, *record*$\rangle\rangle$ should be intelligible to only a subscriber S who has subscribed for $f = \langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $>$, 20$\rangle\rangle$, but not to a subscriber S' who has subscribed for $f' = \langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $>$, 30$\rangle\rangle$. The pub-sub network nodes should be capable of matching the routable attributes in an event e (`topic` and `age` in the above example) against the constraints in a subscription filter f without obtaining any information about the secret attribute `patientRecord`.

While past work on secure content-based routing have suggested using group key management algorithms, the need to keep a publisher informed of such groups of subscribers breaks the decoupling between publishers and subscribers, thereby, consequently weakening the flexibility, performance and scalability of the pub-sub system. In contrast, EventGuard proposes to decouple key management between publishers and subscribers as follows: we associate an authorization key $K(f)$ with a subscription filter f and an encryption key $K(e)$ with an event e. The publisher uses the encryption key $K(e)$ to *encrypt* the secret attributes in an event e; and

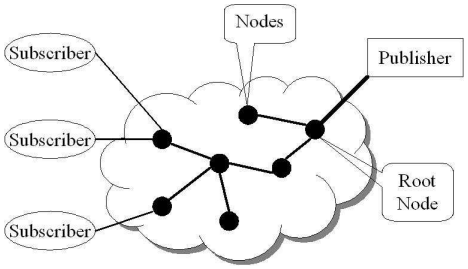

Fig. 1. Basic Pub-Sub System

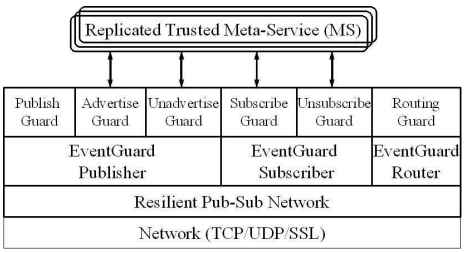

Fig. 2. EventGuard Architecture

the subscriber uses the authorization key $K(f)$ to *decrypt* the secret attributes in a matching event e. We use hierarchical key derivation algorithms [Wong et al. 2000] to map the authorization keys and the encryption keys into a common key space. The mapping ensures that a subscriber can efficiently derive an encryption key $K(e)$ for an event e using an authorization key $K(f)$ for the subscription filter f if and only if the event e matches the subscription filter f. As shown in this paper, disassociating the publisher's encryption key ($K(f)$) with subscriber groups offers significant benefits in terms of both flexibility and scalability.

EventGuard supports token-based subscription matching in the pub-sub network. For content-based subscriptions, events may be safely filtered at the subscriber nodes by deriving the decryption key for the event from the authorization key corresponding to the subscription filter. In order to reduce the risk of targeted selective message dropping attack through passive logging, we develop a probabilistic multi-path routing scheme to minimize the amount of information (about the routable attributes) that can be inferred by the routing nodes.

Besides a decoupled key management scheme and a probabilistic content-based routing scheme, EventGuard comprises of suite of security guards to protect a pub-sub overlay service from various vulnerabilities and threats and ensuring authenticity, availability, confidentiality, and integrity of publications, subscriptions, and pub-sub overlay routing. We present a prototype implementation of EventGuard on top of Siena [Carzaniga et al. 2001] to show that EventGuard is easily stackable on any content-based pub-sub core. With this prototype, we have conducted experimental evaluation of the overhead added by EventGuard to the pub-sub system by comparing EventGuard with Siena. Our experimental results show that EventGuard can secure a pub-sub network with minimal performance penalty.

The rest of this paper is organized as follows. We first present a formal pub-sub system model and a threat model, which serve as the basic system model for the design of EventGuard in Section 2. Section 3 details the design of our security guards, Section 4 presents our scalable key management algorithm and Section 5 describes EventGuard's resilient network design. We present an implementation of EventGuard and evaluate it in Section 6. Section 7 discusses some related work followed by the conclusion in Section 8.

2. PRELIMINARIES

2.1 Reference Pub-Sub Model

In content-based pub-sub systems, publishers publish their contents in terms of event notifications. An event notification is a set of attributes where an attribute is defined by its name, type, and value [Carzaniga et al. 2001]. Subscribers have the ability to express their interest in an event by sending a subscription to the pub-sub overlay network infrastructure. The subscription is a predicate containing one or more constraints (filters). The infrastructure notifies the subscribers of any published notification that matches their subscribed interests.

Pub-sub systems typically support two levels of event matching – *topic-based* and *content-based*. In a topic-based matching scheme [Aguilera and Strom 2000], every event is marked with a topic. A topic could be a simple keyword or any unique numeric identifier. A subscriber subscribes to one or more topics, and receives all the events published under these topics. The pub-sub network routes events based on simple topic matching. Content-based matching schemes [Carzaniga et al. 2001][Aguilera et al. 1999][Banavar et al. 1999] are layered on top of topic-based matching schemes and allow more sophisticated event matching and filtering. For example, a subscriber may specify a *condition* on the event (say, `stock price` > 100) as a part of its subscription.

A typical pub-sub system implements five important primitives: *subscribe*, *advertise*, *publish*, *unsubscribe* and *unadvertise*. Subscribers specify the events in which they are interested using the *subscribe* function. Publishers advertise the type of events they would publish using *advertise*. Publishers publish events via the *publish* function. A subscription is repeatedly matched until it is canceled by a call to *unsubscribe*. An advertisement remains in effect until it is canceled by an *unadvertise*. We use the term messages to loosely denote all traffic on the pub-sub network, including publications, subscriptions, advertisements, unsubscriptions and unadvertisements.

Publications are specified in terms of events and subscriptions are expressed in terms of predicate filters. Formally, an event $e = \langle \alpha \rangle^* = \langle name, type, value \rangle^*$, where α is some attribute of the form $\langle name, type, value \rangle$, *name* refers to some attribute name, *type* refers to the data type of the attribute, *value* corresponds to its published value, and the notation * indicates that an event may comprise one or more attributes. A *filter* selects events by specifying a set of attributes and constraints on the values of those attributes. Formally, filter $f = \langle \phi \rangle^* = \langle name, operator, value \rangle^*$, where ϕ is some constraint of the form $\langle name, operator, value \rangle$, *name* refers to some attribute name, *value* specifies an attribute value, *operator* refers to a binary operator, and the notation * indicates that a filter may be comprised of one or more constraints in a conjunctive form. Operators typically include common equality and ordering relations ($=$, $<$, $>$, etc) for numeric types; and substring, prefix, suffix operators for strings.

An attribute $\alpha = \langle\ name_\alpha, type_\alpha, value_\alpha\ \rangle$ satisfies a constraint $\phi = \langle\ name_\phi, operator_\phi, value_\phi\ \rangle$ if and only if $name_\alpha = name_\phi$, $value_\phi$ is of $type_\alpha$, and $operator_\phi(value_\alpha, value_\phi)$ is true. When an attribute α satisfies a constraint ϕ, we say that α *matches* ϕ. Equivalently, when α matches ϕ, we say that ϕ *covers* α. For example, an attribute $\alpha = \langle price, 120 \rangle$ matches the constraint $\phi = \langle price, \geq, 100 \rangle$.

An event e matches a subscription filter f if for all constraints ϕ in f, there exists some attribute α in e that matches ϕ. When a filter is used in an advertisement, it defines the set of all possible notifications that can be generated by the publisher. An event e matches an advertisement filter f if for all attributes α in e, there exists some constraint ϕ in f that covers α. The notion of *covers* can be extended in a straightforward manner to two subscription filters, or two advertisement filters or a subscription filter and an advertisement filter.

Unsubscriptions and unadvertisements serve to cancel previous subscriptions and advertisements respectively. Given an unsubscription $unsubscribe(X, f)$, where X is the identity of the subscriber and f is a filter, the pub-sub system cancels all subscriptions $subscribe(X, g)$ submitted by the subscriber X with subscription filter g covered by f. Similarly, an unadvertisement message $unadvertise(Y, f)$ cancels all advertisements $advertise(Y, g)$ submitted by the publisher Y with advertisement filter g covered by f.

As illustrated in Figure 1, in a wide-area pub-sub system, publishers and subscribers are usually outside the pub-sub network (though not required). Typically, we have a relatively small set of known and trusted publishers and a much larger set of unknown subscribers. A natural choice for the topology of a pub-sub network is a hierarchical topology (see Figure 1). Other plausible topologies include peer-to-peer and mixed topologies like super-peer topologies [Carzaniga et al. 2001]. For the sake of simplicity, in this paper we assume a hierarchical topology for the pub-sub network. When a node n receives a subscription request $subscribe(m, f)$ from node m, it registers filter f with identity m. If filter f is not covered by any previously subscribed filters at node n then node n forwards $subscribe(n, f)$ to its parent node. Note that node m could be a subscriber or simply another node in the pub-sub overlay network that forwarded a subscription request to node n.

Effectively, for every publisher, a pub-sub dissemination tree is constructed with the publisher as the root, the subscribers as the leaves and the pub-sub routing nodes as the intermediate nodes of the tree. The publications flow from the root (publisher) to the leaves (subscribers) of the tree. Similarly, advertisements, unsubscriptions and unadvertisements are propagated from the root to the leaves of the tree. Note that a node n in the pub-sub network may belong to one or more pub-sub dissemination trees (or so called pub-sub network channels), and each corresponds to a publisher and a topic of events that the publisher publishes through this channel. When a node n receives a publication notification $publish(Y, e)$ from Y to publish the event e, it uses the pub-sub dissemination tree to which it belongs to identify all active subscriptions whose filters $\{f_1, f_2, \cdots, f_p\}$ are matched by the event e. Then, node n identifies and forwards event e to those of its children nodes $\{X_1, X_2, \cdots, X_q\}$ that have subscriptions with subscription filters covered (matched) by a subset of filter f_i $(1 \leq i \leq p)$.

2.2 Threat Model

The pub-sub overlay service model is comprised of three entities: publishers, subscribers and routing nodes. In this section, we present our threat model for all these entities.

Publishers. EventGuard assumes that authorized publishers are honest. All pub-

lications by authorized publishers are assumed to be valid and correct. However, one could build a feedback mechanism wherein the subscribers rate the publishers periodically [Srivatsa et al. 2005][Xiong and Liu 2004]. Over a period of time, subscribers would subscribe only to high quality publishers, and the low quality publishers would eventually run out of business. However, unauthorized publishers may *masquerade* as authorized publishers and *flood* the network and consequently the subscribers, with incorrect or duplicate publications, advertisements or unadvertisements.

EventGuard assumes that event attributes can be partitioned into routable attributes (that are used for in-network routing) and secret attributes (whose confidentiality needs to be preserved from unauthorized entities). This partition is publisher specific and applies to all subscribers subscribing to that publisher. On one hand, this restriction allows EventGuard to be highly scalable while retaining attractive security properties. On the other hand, this restriction limits the class of events that can be protected by EventGuard. For example, in location-based events (e.g., ⟨name, building, room, time⟩) the choice of secret attributes is application-specific. In such scenarios, one may use a common minimum set of routable attributes; however, we note that reducing the number of routable attributes reduces the overall performance of a pub-sub system.

Subscribers. EventGuard assumes that authorized subscribers may be *partially dishonest*. Concretely, we assume that an authorized subscriber does not reveal publications to other unauthorized subscribers (otherwise, this would be equivalent to solving the digital copyrights problem). However, *unauthorized subscribers* may be curious to obtain information about publications to which they have not subscribed. Also, subscribers may attempt to *spam* or *flood* the pub-sub network with duplicate or fake subscriptions and unsubscriptions.

Routing nodes. EventGuard assumes that some of the nodes on the pub-sub network may exhibit dishonest behavior. However, we also assume that a significant fraction of the pub-sub nodes are non-malicious so as to ensure that the network is *alive*. A pub-sub network is alive if it can route messages and maintain its connectivity despite the presence of malicious nodes. Malicious nodes may *eavesdrop* or *corrupt* pub-sub messages routed through them. Malicious nodes may also attempt to selectively (say based on `topic = stockQuote`) or randomly drop pub-sub messages. Further, malicious nodes may attempt to *flood* other nodes and subscribers.

2.3 EventGuard Overview

EventGuard is designed to be completely modular and operates entirely above a content-based pub-sub *core*. EventGuard requires minimal coupling with the pub-sub core and hence can be easily ported from one pub-sub core to another. Figure 2 shows EventGuard's architecture. EventGuard is comprised of three components. The first component is a suite of security guards that guard the pub-sub system from various security threats discussed in Section 2.2. The second component is a light-weight key management service to provide identification and authorization control for advertisements and subscriptions in the pub-sub system. The third component is a resilient pub-sub network that is capable of handling node failures and selective and random dropping-based DoS attacks.

Security Guards. EventGuard comprises of six guards, securing six critical pub-sub operations: subscribe guard, advertise guard, publish guard, unsubscribe guard, unadvertise guard and routing guard. These guards are built on top of content-based routing primitive available in a pub-sub network with the goal of protecting these operations from various attacks discussed in Section 2.2.

Key Management Service. EventGuard relies on a thin *trusted* meta-service (MS) to create keys (that are used for confidentiality and access control in the pub-sub network) and signatures (that are used to ensure the authenticity of control activities such as subscribe, unsubscribe, advertise and unadvertise). The MS also supports a periodic rekeying operation to efficiently handle unsubscriptions in a large pub-sub system. The MS may also include an access control engine that determines the set of filters that a subscriber is authorized to subscribe for and the set of filters that a publisher is authorized to publish under. As described later in this paper, access control is implicitly enforced by issuing the right set of encryption and decryption keys to the authorized publishers and authorized subscribers respectively.

Resilient pub-sub network. EventGuard achieves resilience to node failures and message dropping based attacks by constructing a network topology that is richer than the popular tree-based event dissemination topology. Although a strict tree-based topology minimizes the communication cost in the pub-sub content routing network, it is not robust for handling node failures and message dropping attacks [Srivatsa et al. 2006]. We improve the resilience of the pub-sub network by modifying the tree structure to incorporate multiple independent paths [Srivatsa and Liu 2004] from the publisher to subscribers.

3. EVENTGUARD: BASIC SECURITY GUARDS

In this section we present a high level functional overview of EventGuard. We first introduce the three building blocks used by EventGuard: tokens, keys and signatures. Then we describe how EventGuard uses these primitives to develop six safeguards for securing the six important pub-sub operations: subscribe, advertise, publish, unsubscribe, unadvertise and routing. In this section, we first describe EventGuard mechanisms in the context of a topic-based pub-sub system. Then, we present EventGuard mechanisms to handle more complex subscriptions.

Signatures play a fundamental role in achieving message authentication and protecting the pub-sub services from flooding-based DoS attacks. EventGuard uses a probabilistic signature algorithm for achieving authenticity. A signature scheme is probabilistic if there are many possible valid signatures for each message and the verification algorithm accepts any of the valid signatures as authentic. In the first prototype of EventGuard, we use ElGamal [ElGamal 1985] as the probabilistic signature algorithm. A signature on any message M using ElGamal yields a tuple $\langle r, s \rangle$. The r-component of the signature is guaranteed to be unique (with high probability). Further, if the same message M is signed twice by the same entity x, we get two different, but valid ElGamal signatures of M. All messages originating at entity x are signed using its private key $rk(x)$; and all its signatures are verified using its corresponding public key $pk(x)$. EventGuard uses the trusted meta-service MS to create signatures for advertisements and subscriptions. Subscriptions and

advertisements are authenticated using signatures, ensuring that malicious nodes cannot flood the pub-sub network with bogus publications or phony subscriptions. EventGuard requires the ability to generate public/private keys and certificates for the MS and the publishers in the pub-sub network (using OpenSSL [OpenSSL]). EventGuard uses in-built mechanisms for distributing certificates and public keys. As described later in this paper, publishers and subscribers receive MS's public key (with certificate) when they send their first advertisement or a subscription request to the MS. Subscribers also receive a publisher's public key (with certificate) from advertisements disseminated through the pub-sub network by the publishers.

We have introduced tokens, keys and signatures as fundamental building blocks of EventGuard. The next challenge is to design and construct the six concrete safeguards for the following six essential operations: subscribe, advertise, publish, unsubscribe, unadvertise and routing.

3.1 Subscribe Guard

Subscribe guard is designed for achieving subscription authentication, subscription confidentiality and subscription integrity, and preventing DoS attacks based on spurious subscriptions. Suppose that a subscriber S wishes to subscribe for a topic w. In EventGuard, subscriber S sends the topic w to the EventGuard trusted meta service MS indicating that it wishes to subscribe for topic w. At this point, the MS may act as the authority for implementing a cost model for the pub-sub system. For example, the MS may collect a subscription fee for every subscription; the subscription fee may be dependent on the topic w. Let $\phi'(w)$ be the original subscription filter for topic w sent to MS by the subscriber S, $sb(w)$ denote the subscription permit issued by MS upon receiving a subscription $\phi'(w)$ from subscriber S, and $\phi(w)$ denote the legal subscription transformed from $\phi'(w)$ by MS in two steps: (1) replacing topic w with token $T(w)$ and (2) signing the subscription with the subscription signature provided by MS. Both are included in the subscription permit $sb(w)$ generated by MS. They are defined as follows:

$$\begin{aligned}
\phi'(w) &= \langle \texttt{topic}, EQ, w \rangle \\
sb(w) &= \langle K(w), T(w), sig^{S}_{MS}(T(w)), UST^{S}(w) \rangle \\
\phi(w) &= \langle \texttt{topic}, EQ, T(w) \rangle, \langle sig, ANY, sig^{S}_{MS}(T(w)) \rangle
\end{aligned}$$

The MS verifies access rights for a subscriber (if such access control rules are mandated by the publisher) and sends a subscription permit $sb(w)$ to the subscriber S. The key $K(w)$ for topic w is derived as $K(w) = KH_{rk(MS)}(w)$, where $rk(MS)$ denotes the MS's private key and $KH_K(w)$ denotes a keyed hash of string w using a keyed-hash function KH (say HMAC-SHA1 [Krawczyk et al.]) and a secret key K. EventGuard supports periodic (epoch based) rekeying to ensure that subscribers cannot read events past their subscription epoch. The token $T(w)$ for topic w is derived as $T(w) = H(K(w))$, where $H(x)$ denotes a hash of string x using a one-way hash function H (say, MD5 [Rivest 1992] or SHA1 [Eastlake and Jones 2001]). $UST^S(w)$ is a special token given to the subscriber to enable safe unsubscription (discussed later under unsubscribe guard). Observe that if any two subscribers subscribe for topic w, they get the same encryption key $K(w)$ and the same token $T(w)$.

The signature $sig^S_{MS}(T(w))$ is an ElGamal signature by the MS on the token $T(w)$ in the subscription permit $sb(w)$ provided to subscriber S. The signature has two parts $sig^S_{MS}\ (T(w)) = \langle r,\ s\rangle$. Note that the r-component of the signature is always unique. Therefore, we use r-component of the signature as the subscription identifier. This signature serves us three purposes. First, it enables nodes in the pub-sub network to check the validity of a subscription. Second, we use the subscription identifier (the r-component of the signature) to detect and curb DoS attacks based on subscription flooding. Note that even if two subscribers S and S' subscribe for the same topic w, $sig^S_{MS}(T(w)) \neq sig^{S'}_{MS}(T(w))$ (discussed later under routing guard). Third, it is used to construct the special token $UST^S(w)$ $= KH_{rk(MS)}(r)$ where r denotes the r-component of the MS's signature. We use $UST^S(w)$ to prevent DoS attacks based on fake unsubscription (discussed later under unsubscribe guard).

Upon receiving a subscription permit $sb(w)$ from the MS, subscriber S transforms its original subscription filter $\phi'(w)$ to a legal subscription filter $\phi(w)$. The subscriber S could then submit and deploy the signed subscription $\phi(w)$ on the pub-sub network. Consequently, any publication that includes the token $T(w)$ is routed to S. Routing nodes on the pub-sub network are not able to perform unauthorized reads or writes on the content of any subscription message, thus guaranteeing subscription confidentiality and integrity. Further, nodes compromised due to DoS attacks, even though malicious, are not able to attack the pub-sub network by flooding fake subscriptions.

A subscriber S may restrict the number of publications it would like to receive. For example, a subscriber may use $sb(w_1)$ and $sb(w_2)$ to construct a subscription filter that is a conjunction of filters $f(w_1)$ and $f(w_2)$. In general a subscription filter $f = \langle \phi(w_1),\ \phi(w_2),\ \cdots,\ \phi(w_m)\rangle$, where $\phi(w)$ described above.

3.2 Publish Guard

Publish guard is designed to safeguard publication confidentiality, publication authenticity, and DoS attacks based on bogus publications and spam. Suppose a publisher P wishes to publish a publication *pbl* under topics $w_1,\ w_2,\ \cdots,\ w_m$. The topics are used to categorize the content *pbl*. The content *pbl* could be any arbitrary sequence of bytes including text, multimedia, and so on. For each topic w_i, the publisher fetches the topic's token $T(w_i)$ and its encryption key $K(w_i)$ from the MS. A publication event e is constructed as follows. Let e' denote the original publication message and e denote a legal event publication transformed from e' using tokens and content encryption of publication messages. We formally define them as follows:

$$
\begin{aligned}
e' &= \langle\langle \texttt{publisher}, P\rangle, \langle \texttt{content}, pbl\rangle, \langle \texttt{topic}, w_1\rangle, \cdots, \langle \texttt{topic}, w_m\rangle\rangle \\
e &= \langle\langle \texttt{publisher}, P\rangle, \langle \texttt{content}, E_{K_r}(pbl)\rangle, \langle \texttt{topic}, T(w_1)\rangle, \langle T(w_1), E_{K(w_1)}(K_r)\rangle, \cdots, \\
&\quad \langle \texttt{topic}, T(w_m)\rangle, \langle T(w_m), E_{K(w_m)}(K_r)\rangle\rangle
\end{aligned}
$$

The key K_r is a random encryption key generated each time a publisher needs to publish an event. P sends the event e along with its signature, namely, $sig_P(e)$; we note that the certificate for a publisher's public key is distributed to the subscribers using the advertisement message described in the following section. Observe that

any subscriber for topic w_i possesses the key $K(w_i)$. An authorized subscriber uses the key $K(w_i)$ to decrypt the random key K_r, and uses the random key K_r to decrypt the publication *pbl*.

The publisher uses an ElGamal signature to sign its publications. The first component of the signature is used as the publication identifier. The signature serves two purposes. First, it enables nodes in the pub-sub network to check the validity of a publication (a publisher's public key is distributed to pub-sub network nodes and subscribers via advertisements as discussed later under advertise guard). Second, we use the publication identifier (the r-component of the signature) to detect and curb a DoS attack based on publication flooding (discussed later under routing guard).

When multiple publishers publish on a common topic, it might be essential to ensure that the publications from a publisher P are not readable by another publisher P'. EventGuard handles this problem using a small modification to the authorization key $K(w)$ for topic w. Instead of having a topic key shared across all users the MS can generate a per publisher authorization key for topic w as $K^P(w) = KH_{rk(MS)}(P \parallel w)$. The MS distributes $K^P(w)$ to a publisher. The MS uses $K^P(w)$ to derive authorization keys for subscribers that subscribe to a topic w from publisher P. This incurs almost no additional key generation cost. On the other hand, the subscriber group approach has to maintain separate groups for every publisher P.

3.3 Advertise Guard

Advertise guard is designed for achieving advertisement authentication, advertisement confidentiality and integrity, and preventing DoS attacks based on bogus advertisements. Suppose a publisher P wishes to publish events under topic w. Publisher P sends w and its public-key $pk(P)$ to the MS. At this point the MS may charge a publication fee to the publisher that is some arbitrary function of w. $\phi'(w)$ is the original advertisement filter for topic w.

$$\begin{aligned}
\phi'(w) &= \langle \texttt{publisher}, EQ, P\rangle, \langle \texttt{topic}, EQ, w\rangle \\
ad(w) &= \langle K(w), T(w), sig^P_{MS}(T(w) \parallel P \parallel pk(P)), UAT^P(w)\rangle \\
\phi(w) &= \langle \texttt{publisher}, EQ, P\rangle, \langle \texttt{topic}, EQ, T(w)\rangle, \langle sig, ANY, sig^P_{MS}(T(w) \parallel P \parallel pk(P))\rangle
\end{aligned}$$

The MS sends an advertisement permit $ad(w)$ to the publisher P. The key $K(w)$, the token $T(w)$ and the signature $sig^P_{MS}(T(w) \parallel P \parallel pk(P))$ are computed in the same manner as that for subscriptions. The special token $UAT^P(w)$ is used for unadvertisements (discussed in unadvertise). The publisher then constructs the advertisement filter ϕ and propagates it to the pub-sub network. Note that the public-key $pk(P)$ is essential for the pub-sub nodes and the subscribers to verify the authenticity of publications.

3.4 Unsubscribe Guard

Unsubscribe guard is designed to prevent unauthorized unsubscribe messages, flooding of unsubscribe messages, and spam. When a subscriber S wishes to unsubscribe from a topic w, S sends $\langle T(w),\ sig^S_{MS}(T(w)),\ UST^S(w)\rangle$ to the MS. The MS checks if $sig^S_{MS}(T(w))$ is a valid signature on $T(w)$. The MS uses the special token

$UST^S(w)$ to ensure protection from DoS attacks based on fake unsubscription. The MS checks if $UST^S(w)$ is indeed equal to $KH_{rk(MS)}(sbId)$, where $sbId$ denotes the subscription identifier, namely, the r-component of the signature $sig^S_{MS}(T(w))$. Note that the signature $sig^S_{MS}(T(w))$ and the token $T(w)$ are sent to the pub-sub network nodes when the subscriber S subscribes for the topic w. However, the subscriber S is never required to reveal the special token $UST^S(w)$ to the pub-sub network. Hence, no malicious node in the pub-sub network would be able to fake an unsubscribe request. Moreover, the use of $UST^S(w)$ prevents some subscriber S' ($\neq S$) who has subscribed for topic w (and thus possesses signature $sig^{S'}_{MS}(T(w))$, token $T(w)$ and key $K(w)$) from unsubscribing subscriber S from topic w. We use $\phi'(w)$ to denote the original unsubscription message for topic w.

$$\begin{aligned} \phi'(w) &= \langle \texttt{topic}, EQ, w \rangle \\ usb(w) &= \langle sig_{MS}(T(w) \parallel sbId) \rangle \\ \phi(w) &= \langle \texttt{topic}, EQ, T(w) \rangle, \langle sig, ANY, sig_{MS}(T(w) \parallel sbId) \rangle \end{aligned}$$

The MS sends an unsubscription permit $usb(w)$ to the subscriber S. Note that the signature includes the token $T(w)$ and the original subscription's identifier $sbId$. Subscriber S would unsubscribe from topic w by sending $\phi(w)$ to the pub-sub network. Nodes in the network use the MS's signature to check the validity of an unsubscription and use the unsubscription identifier $usbId$ (the r component of signature $sig_{MS}(T(w) \parallel sbId)$) to detect and curb DoS attacks based on unsubscription flooding.

In EventGuard, an authorization key $K(f)$ act like a capability issued to authorize subscribers to read all events e that match the filter f. As described in our subscription model (see Section 2.1), all subscriptions are accompanied by a payment and are valid for one time epoch. We use a rekeying algorithm that is similar to the lazy revocation (epoch based periodic rekeying) algorithms used in several group key management protocols [Yang et al. 2001]. At the beginning of a new epoch, if the subscribers need to refresh their subscriptions, then they must obtain new authorization keys from the MS. To avoid flash crowds attempting to subscribe at the beginning of a new epoch, we evenly space out the epoch intervals on a per-topic basis. We also adaptively vary the length of the epoch on a per-topic basis using the subscription history. Detailed discussion on choosing the per-topic epoch length is outside the scope of this paper.

3.5 Unadvertise Guard

Unadvertise guard is designed to prevent the pub-sub network from unadvertisement flooding. When a publisher P wishes to unadvertise for a topic w, P sends $\langle T(w),\ sig^P_{MS}(T(w) \parallel P \parallel adId),\ UAT^P(w) \rangle$ to the MS. Similar to those illustrated in unsubscribe guard, the special token $UAT^P(w)$ is computed as follows: $UAT^P(w) = KH_{rk(MS)}\ (adId)$, where $adId$ denotes the advertisement identifier, namely, the r-component of the signature $sig^P_{MS}(T(w))$. Note that the use of $UAT^P(w)$ ensures DoS attacks based on phony unadvertisements. Let $\phi'(w)$ de-

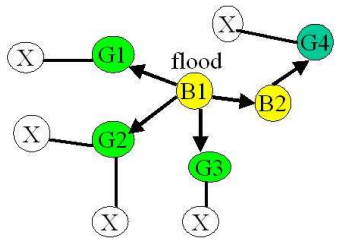

Fig. 3. Handling Flooding based DoS attacks in EventGuard

note the original unadvertisement message for topic w.

$$\begin{aligned}\phi'(w) &= \langle \texttt{publisher}, EQ, P\rangle, \langle \texttt{topic}, EQ, w\rangle \\ uad(w) &= \langle sig_{MS}(T(w) \parallel P \parallel adId)\rangle \\ \phi(w) &= \langle \texttt{publisher}, EQ, P\rangle, \langle \texttt{topic}, EQ, T(w)\rangle, \langle sig, ANY, sig_{MS}(T(w) \parallel P \parallel adId)\rangle\end{aligned}$$

Upon receiving an unadvertise request from publisher P, the MS generates an unadvertisement permit $uad(w)$ and send it back to the publisher P. The publisher P uses the advertisement signature $sig^{P}_{MS}(T(w) \parallel P \parallel adId)$ included in the permit to create a legal unadvertise request and submit it to the pub-sub overlay network. This signature (similar to unsubscription) is used by the routing nodes to check its authenticity and detect DoS attacks based on unadvertisement flooding.

3.6 Routing Guard

The pub-sub network nodes route messages based on tokens − the pseudonym for topics. Besides performing the functionality of a regular pub-sub node, we require the nodes to perform additional checks to ensure safety from DoS attacks. Now, we discuss the checks implemented by nodes to protect the pub-sub network from flooding-based DoS attacks.

EventGuard requires nodes on the pub-sub network to perform two security checks. The first check is based on signatures for maintaining sender authenticity and the second check is based on detecting duplicate messages. Subscriptions, unsubscriptions, advertisements and unadvertisements are verified for the MS's signature. The publications are verified for its publisher's signature. Duplicates are checked using the r-component of the signature. Recall that we designate the r-component of the ElGamal signature as the message's identifier. When a node receives two subscriptions with the same identifier, it blocks the later one. With the guarantee of sender authenticity and the prevention of duplicate messages, no flooding attack could propagate beyond one good pub-sub node. Figure 3 illustrates this point. In Figure 3, a malicious (bad) node $B1$ attempts a flooding based DoS attack to all its neighbor nodes. Observe that no invalid message (incorrect signatures) and no duplicate message from node $B1$ would propagate beyond the non-malicious (good) nodes $G1$, $G2$, $G3$ and $G4$. More importantly, none of the nodes marked X would be hit by this DoS attack. Thus, by deploying routing guards in the pub-sub network, EventGuard can effectively isolate the effect of flooding attacks.

We implement the routing guard (i.e., the two security checks on each routing

node) in three steps. First, we require a node to maintain the history of identifiers previously seen by it. Second, we augment each EventGuard message with a timestamp that is signed by the MS (for advertisement, subscription, unadvertisement and unsubscription) or signed by the publisher (for a publication). Third, a non-malicious node blocks any message if the condition $|ct - ts| > max\_delay$ is met, where ct is the current time, ts is the timestamp on a message, and max_delay is a system defined parameter. Nodes only need to maintain a history of identifiers for a time duration of max_delay. Note that max_delay must account for time skew between nodes and routing and communication delays on the pub-sub network.

4. EVENTGUARD: KEY MANAGEMENT

We have so far described EventGuard mechanisms for a simple topic-based subscription models. In this section, we extend EventGuard mechanisms to handle more sophisticated content-based matching operators (see Section 2.1 for the definition of topic-based and content-based matching operators). In section 3 we used a per-topic key to enforce event confidentiality from routing nodes and unauthorized subscribers. However, content-based pub-sub networks may use more sophisticated matching operators, such as numeric attribute based matching operators ($>$, $<$). In this section, we present secure and scalable key management algorithms to enforce event confidentiality for content-based matching operators.

4.1 Overview

Secure event dissemination with content-based matching operators refers to preserving the confidentiality of secret attributes in an event from unauthorized subscribers and the routing nodes in the pub-sub network. Most existing key management solutions for pub-sub networks use group key management protocols to manage subscribers grouped together based on their subscriptions. However, given a flexible subscription filter based authorization model, every event can potentially go to a different subset of subscribers. In the worst case, for NS subscribers, there are 2^{NS} subgroups, thereby making it infeasible to set up static groups for every possible subgroup. Although some optimizations have been proposed for dynamic groups such as key caching [Opyrchal and Prakash 2001], the worst case key management cost remains at $O(2^{NS})$ due to its inherent design.

EventGuard improves past solutions to the key management problem using a completely different design philosophy. Our key management algorithms disassociate keys from subscriber groups and ensure that the key management cost is *independent* of the total number of the subscribers (NS) in the pub-sub system. This is achieved by associating a subscription key $K(f)$ with a filter f and an encryption key $K(e)$ with an event e such that it is computationally feasible to derive $K(e)$ from $K(f)$ (using routable event attributes) if and only if e matches f. While this offers complete confidentiality to secret attributes in an event, the routable attributes may be vulnerable to some inference attacks by the pub-sub network nodes. EventGuard uses a resilient network (see Section 5) to support probabilistic multi-path event routing to allow scalable content-based routing, while minimizing the amount of information (about the routable attributes) that can be inferred by the routing nodes. The primary idea here is to route events from a publisher to its subscribers probabilistically using multiple independent paths such that the fre-

quency of all tokens (routing labels on an event) appears (nearly) indistinguishable for all the routing nodes in the pub-sub network.

4.2 Key Management Algorithms

In EventGuard, event confidentiality is implemented using authorization keys and encryption keys. These keys serve complementary purposes. An encryption key is used to encrypt an event so as to maintain its confidentiality from the routing nodes and the subscribers who have not subscribed to that event. An authorization key is used as an authorization permit for subscribers to decrypt an event. We embed encryption and authorization keys into a common *key space* using hierarchical key derivation algorithms [Wong et al. 2000] such that a subscriber can use its authorization keys to efficiently derive the encryption keys only for those events that match their subscriptions. In this section we describe our key management algorithm and present a detailed quantitative analysis that highlights the benefits of our approach against the group key management approach.

We use authorization keys and encryption keys to support access control on pub-sub systems. These keys serve complementary purposes. An encryption key is used to maintain the confidentiality of an event from subscribers who have not subscribed to that event. An authorization key is designed to encode content-based matching semantics into a key derivation algorithm such that an authorized subscriber can efficiently derive the encryption keys for those events that match their subscriptions. In this paper, we demonstrate our approach using four different types of publication-subscription matching: topic or keyword based matching, numeric attribute based matching, category based matching, and string based suffix/prefix matching.

For topic or keyword based matching, an authorization key $K(f)$ associated with a filter $f = \langle \texttt{topic}, EQ, \texttt{cancerTrail} \rangle$ must be capable of decrypting the message msg in event $e = \langle \langle \texttt{topic}, \texttt{cancerTrail} \rangle, \langle \texttt{message}, msg \rangle \rangle$. On the other hand, a key $K(f')$ associated with filter $f' = \langle \texttt{topic}, EQ, \texttt{humanGenome} \rangle$ should not be able to decrypt msg in event e. For numeric attribute based matching, a key $K(f_1)$ used for the filter $f_1 = \langle \langle \texttt{topic}, EQ, \texttt{cancerTrail} \rangle, \langle \texttt{age}, >, 20 \rangle \rangle$ and a key $K(f_1')$ used for the filter $f_1' = \langle \langle \texttt{topic}, EQ, \texttt{cancerTrail} \rangle, \langle \texttt{age}, >, 30 \rangle \rangle$ must be capable of decrypting the message msg in event $e_1 = \langle \langle \texttt{topic}, \texttt{cancerTrail} \rangle, \langle \texttt{age}, 35 \rangle, \langle \texttt{message}, msg \rangle \rangle$. On the other hand, key $K(f_1)$ should be capable of decrypting the message msg in event $e_1' = \langle \langle \texttt{topic}, \texttt{cancerTrail} \rangle, \langle \texttt{age}, 25 \rangle, \langle \texttt{message}, msg \rangle \rangle$, but not the key $K(f_1')$. For category based matching, a key $K(f_2)$ used for filter $f_2 = \langle \langle \texttt{topic}, EQ, \texttt{cancerTrail} \rangle, \langle \texttt{news}, \ni, \texttt{unclassifiedNews} \rangle \rangle$, a key $K(f_2')$ used for $f_2' = \langle \langle \texttt{topic}, EQ, \texttt{cancerTrail} \rangle, \langle \texttt{news}, \ni, \texttt{classifiedNews} \rangle \rangle$, and a key $K(f_2'')$ used for $f_2'' = \langle \langle \texttt{topic}, EQ, \texttt{cancerTrail} \rangle, \langle \texttt{news}, \ni, \texttt{secretNews} \rangle \rangle$ must be capable of decrypting the event $e_2 = \langle \langle \texttt{topic}, \texttt{cancerTrail} \rangle, \langle \texttt{news}, \texttt{unclassifiedNews} \rangle, \langle \texttt{message}, msg \rangle \rangle$. On the other hand, only $K(f_2'')$ should be capable of decrypting the message msg in $e_2' = \langle \langle \texttt{topic}, \texttt{cancerTrail} \rangle, \langle \texttt{news}, \texttt{secretNews} \rangle, \langle \texttt{message}, msg \rangle \rangle$, but not the keys $K(f_2)$ and $K(f_2')$. For string based prefix/suffix matching, a key $K(f_3)$ used for filter $f_3 = \langle \langle \texttt{topic}, EQ, \texttt{cancerTrial} \rangle, \langle \texttt{name}, \text{PF}, \texttt{a} \rangle \rangle$ and a key $K(f_3')$ used for the filter $f_3' = \langle \langle \texttt{topic}, EQ, \texttt{cancerTrial} \rangle, \langle \texttt{name}, \text{PF}, \texttt{an} \rangle \rangle$ should be capable of decrypting the message msg in event $e_3 = \langle \langle \texttt{topic}, \texttt{cancerTrial} \rangle, \langle \texttt{name}, \texttt{andy} \rangle, \langle \texttt{message}, msg \rangle \rangle$. On the other hand, only $K(f_3)$ must be capable of decrypting the message msg in $e_3' = \langle \langle \texttt{topic}, \texttt{cancerTrial} \rangle, \langle \texttt{name}, \texttt{alex} \rangle,$

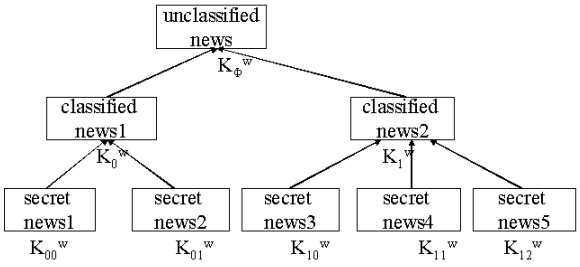

Fig. 4. Key Tree: Category Hierarchy

$\langle$`message`, $msg\rangle\rangle$, but not the key $K(f_3')$.

In the following sections, we first describe our techniques to handle simple subscriptions that consists of a topic and at most one constraint, say, $f = \langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $>$, $15\rangle\rangle$. A complex subscription could consist of constraints combined using the $\wedge$ and $\vee$ Boolean operators. We have described algorithms to handle numeric attribute based in-network matching in [Srivatsa and Liu 2007]. In this paper, we describe our key management algorithms for category based matching (Section 4.3), string based prefix/suffix matching (Section 4.4) followed by techniques to handle complex subscriptions in Section 4.5.

4.3 Category Based Matching

In this section, we present techniques for access control on named categories that are typically arranged as a category tree. In a category tree the children of a node represent more detailed information about the same topic than its parent and thus may be considered more confidential. An example category hierarchy that is applicable in a military scenario is shown in Figure 4. A subscriber who subscribes for `secretNews1` is implicitly entitled to receive all publications published under categories `classifiedNews1` and `unclassifiedNews`; however, the converse is not true. Additionally, a subscriber who subscribes for `secretNews1` is not permitted to read events categorized under `classifiedNews2`. In general, we use a category matching operation $\ni$ such that an event $e = \langle name_{\alpha}, value_{\alpha}\rangle$ matches a filter $f = \langle name_{\phi}, \ni, value_{\phi}\rangle$ if and only if $value_{\phi}$ is an ancestor of $value_{\alpha}$ in a category tree named $name$.

Our key derivation algorithm supports category based subscriptions. Given a category, say `news`, we can construct a subscription filter $f = \langle$`news`, $\ni$, $cat\rangle$. The filter f matches any event $e = \langle$`news`, $v\rangle$ if and only if v is an ancestor of cat on the category tree. We associate an authorization key $K(f)$ with every subscription filter f and an encryption key $K(e)$ with every event e. The authorization keys and the encryption keys satisfy the following properties:

—Given $K(f)$ it should be computationally easy to derive a key $K(e)$, if $v \in ancestor(cat)$.

—Given $K(f)$ it should be computationally infeasible to derive a key $K(e)$, if $v \notin ancestor(cat)$.

We construct keys that satisfy the above mentioned properties as follows. We map

the authorization keys and encryptions into a common key space constructed using a category attribute key tree (CAKT for short). Given a subscription filter $\langle$`news`, $\ni$, *cat*$\rangle$, we use the CAKT to derive an authorization key $K(f)$ that corresponds to the category *cat*, denoted by K^{ont}_{cat}, where *ont* denotes the name of the ontology (in this example, *ont* = `news`). The CAKT enables one to easily (computationally) derive a key $K^{ont}_{cat'}$ from K^{ont}_{cat} if and only if $cat' \in ancestor(cat)$ in the category tree corresponding to the ontology *ont*. For any event $e = \langle$`news`, $v\rangle$, we encrypt the message with the encryption key $K(e) = K^{ont}_{v}$. By the construction of the category key tree it follows that $K(e)$ is easily derivable from $K(f)$ if and only if $v \in ancestor(cat)$.

Preliminaries. Given a category ontology, we map each category to a key tree identifier *ktid*. The root of the tree is assigned *ktid* = $\emptyset$. We label the i^{th} child of a specialization with *ktid* = ξ as $\xi \parallel i$. For the sake of simplicity we assume that CAKT is a binary category key tree such that each specialization has exactly two or zero children. However, the techniques discussed in this paper can be extended in a straightforward fashion to handle a case where different specializations in the category tree have different numbers of children. In our experimental section, we use an ontology tree wherein the number of children per tree node was randomly chosen between 2 to 4. Now we describe a technique to construct a CAKT, and then present algorithms for constructing subscriptions and publications using the CAKT.

Category Key Tree (CAKT). We derive a parent element ξ's key from its children elements ($\xi \parallel 0$) and ($\xi \parallel 1$) as follows: $K^{ont}_{\xi} = mix(blind(K^{ont}_{\xi\parallel 0}, K^{ont}_{\xi\parallel 1}))$ $= mix(blind(K^{ont}_{\xi\parallel 1}, K^{ont}_{\xi\parallel 0}))$. There are several options for functions *mix* and *blind*. The function *blind* is chosen such that given $blind(x)$ it is very hard to guess x. The function *mix* is chosen such that $mix(blind(x), y) = mix(blind(y), x)$.

The functions *blind* and *mix* are defined based on Diffie-Hellman logical key hierarchy (DH-LKH) [Rafaeli and Hutchison 2003] as follows: $blind(x) = g^{H(x)}$ *mod* p and $mix(g^{H(x)}$ *mod* p, $y) = g^{H(x)H(y)}$ *mod* p. The parameter p is a large prime such that discrete log problem in the field Z_p is computationally intractable. The parameter g is a generator in field Z_p. Prime p and generator g are assumed to be system wide known parameters. Observe that $mix(g^{H(y)}$ *mod* p, $x) = g^{H(y)H(x)}$ *mod* $p = mix(g^{H(x)}$ *mod* p, $y)$. Hence, K^{ont}_{ξ} is derived as $K^{ont}_{\xi} = g^{H(K^{ont}_{\xi\parallel 0})H(K^{ont}_{\xi\parallel 1})}$ *mod* p for some $\xi \in (0+1)^*$. We use the least significant 128 bits of the result as the actual key. For example, $K^{\texttt{news}}_{0} = g^{H(K^{\texttt{news}}_{00})H(K^{\texttt{news}}_{01})}$ *mod* p.

Analogous to the category key tree, the pub-sub system also generates a *blinded key tree*. The *MS* is responsible for generating a blinded key tree $BK^{ont}_{\xi} = blind(K^{ont}_{\xi})$, for all key tree identifiers ξ in the CAKT. The blinded keys are required for a subscriber to generate the authorization keys for all the specializations that it is authorized for. Also, by the hardness of the discrete log problem in the field Z_p it is computationally infeasible to derive a key K^{ont}_{ξ} using only its blinded key BK^{ont}_{ξ} or from its children blind keys $BK^{ont}_{\xi\parallel 0}$ and $BK^{ont}_{\xi\parallel 1}$.

For every leaf element with key tree identifier equal to *ktid*, the *MS* generates key $K^{ont}_{ktid} = KH_{K(w)}$ ($ont \parallel ktid$), where $K(w) = KH_{rk(MS)}(w)$ is the authorization key for the topic w, and $rk(MS)$ denotes the meta-service secret key. For example,

the key for leaf element `secretNews1` (with $ktid = 00$) under ontology $ont =$ `news` and topic $w =$ `cancerTrail` is derived as $K_{00}^{\texttt{news}} = KH_{K(\texttt{cancerTrail})}(\texttt{news} \parallel 00)$, where $K(\texttt{cancerTrail}) = KH_{rk(MS)}(\texttt{cancerTrail})$. For any non-leaf element on the key tree, the *MS* derives its key using the publicly available blinded key tree for the topic w and ontology *ont*. Note that the *MS* does not have to store any extra confidential information to handle subscriptions; the key that corresponds to a given *ktid* under a topic w can be efficiently computed on the fly. The *MS* spends only a one-time effort to generate a blinded key tree for every topic w and ontology *ont*.

Publication. The encryption keys for an event are constructed as follows:

$$\begin{aligned} e &= \langle\langle \texttt{topic}, w\rangle, \langle ont, cat\rangle, \langle \texttt{message}, msg\rangle\rangle \\ K(e) &= K_{ktid(cat)}^{ont} \end{aligned}$$

For example, a publication $e = \langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`news`, `classifiedNews1`$\rangle$, $\langle$`message`, $msg\rangle\rangle$ is encrypted as follows. P identifies that the element `unclassifiedNews1` in the key tree has an identifier 0 (see Figure 4). P generates the encryption key $K(e) = K_0^{\texttt{news}}$.

Subscription. The authorization keys for a subscription filter are constructed as follows:

$$\begin{aligned} f &= \langle\langle \texttt{topic}, EQ, w\rangle, \langle ont, \ni, cat\rangle\rangle \\ K(f) &= K_{ktid(cat)}^{ont} \end{aligned}$$

Given a publication with msg with `ktid` $= ktid_\alpha$, a subscriber who has subscribed for `ktid` $= ktid_f$ does the following. The subscriber checks if $ktid_\alpha$ is a prefix of $ktid_f$. The subscriber uses this information to extract the suffix $b_0 b_1 \cdots b_{m-1}$ and derives the key $K_{ktid_\alpha}^{ont}$ from the key $K_{ktid_\phi}^{ont}$. Observe that any subscriber who possesses the key that corresponds to some element of the key tree can efficiently derive the keys for all its ancestors recursively as $K_\xi^{ont} = (BK_{\xi \parallel b}^{ont})^{H(K_{\xi \parallel \overline{b}}^{ont})} \ mod\ p$ for some $\xi \in (0+1)^*$, $b = 0$ or 1 and $\overline{b}$ denotes the bit complement of b. However, it is computationally infeasible for a subscriber to derive the keys corresponding to its children or siblings.

For example, given a publication with message msg encrypted with the key $K_0^{\texttt{news}}$ a subscriber S who possesses the key $K_{00}^{\texttt{news}}$ does the following. The subscriber extracts the publication's key tree identifier $ktid_\phi = 0$ and its subscription's key tree identifier $ktid_\alpha = 00$. Then, S identifies that element 0 is an ancestor of element 00. Then, S derives $K_0^{\texttt{news}} = (BK_{01}^{\texttt{news}})^{H(K_{00}^{\texttt{news}})} \ mod\ p = g^{H(K_{00}^{\texttt{news}})H(K_{01}^{\texttt{news}})} \ mod\ p$ (since, $BK_{01}^{\texttt{news}} = g^{H(K_0^{\texttt{news}})} \ mod\ p$). Recall that the blinded key tree $BK_{ktid}^{\texttt{news}}$ is made publicly available by the pub-sub system. Now, S can use $K_0^{\texttt{news}}$ to decrypt the message msg in the publication.

4.4 String Based Suffix and Prefix Matching

In this section, we present our key derivation algorithm for string based suffix/prefix matching. Given a string attribute, say `name`, we can construct a subscription filter $f = \langle$`name`, PF, $u\rangle$. The filter f matches any event $e = \langle$`name`, $v\rangle$ if and only if string u is a prefix of string v. We associate an authorization key $K(f)$ with

every subscription filter f and an encryption key $K(e)$ with every event e. The authorization keys and the encryption keys satisfy the following properties:

—Given $K(f)$ it should be computationally easy to derive a key $K(e)$ if u PF v, that is, u is a prefix of v.

—Given $K(f)$ it should be computationally hard to derive a key $K(e)$ if u is not a prefix of v.

We construct keys that satisfy the above mentioned properties as follows. We map the authorization keys and encryption keys to the common key space using a string attribute key tree (SAKT for short). Given a subscription filter $\langle str, PF, u\rangle$, we use the SAKT to derive an authorization key that corresponds to the string u, denoted by K_u^{str}, where str denotes the name of the string attribute. The SAKT enables one to easily (computationally) derive a key K_v^{str} from K_u^{str} if and only if u PF v. For any event $e = \langle str, v\rangle$, we encrypt the message with $K(e) = K_v^{str}$. By the construction of the string attribute key tree it follows that $K(e)$ is easily derivable from $K(f)$ if and only if u PF v. Our construction for string suffix matching is very similar to string prefix matching and will not be discussed separately.

String Attribute Key Tree (SAKT). We first present a technique to construct the SAKT. Given a string value $u = u_0u_1\cdots u_{k-1}$, where u_i denotes a character in the string u, we construct a key K_u^{str} recursively as follows: $K_{u_0u_1\cdots u_i}^{str} = KH_{K_{u_0u_1\cdots u_{i-1}}^{str}}(u_i)$, where KH denotes a keyed hash function. Let the symbol Ø denote the null string. We derive the authorization key for the null string corresponding to the key tree as $K_{\emptyset}^{str} = KH_{K(w)}(str)$, where $K(w) = KH_{rk(MS)}(w)$ is the authorization key for the topic w, and $rk(MS)$ denotes the MS's secret key. An example topic would be $w =$ `cancerTrail` and string attribute $str =$ `name`. For example, $K_{\mathtt{a}}^{str}$ is derived as $K_a^{str} = KH_{K_{\emptyset}^{str}}(\mathtt{a})$ and $K_{\mathtt{an}}^{str}$ is derived as $K_{\mathtt{an}}^{str} = KH_{K_{\mathtt{a}}^{str}}(\mathtt{b})$.

Publication. A publisher P constructs the encryption key for a string attribute event e as follows:

$$\begin{aligned} e &= \langle\langle \mathtt{publisher}, P\rangle, \langle \mathtt{topic}, w\rangle, \langle str, v\rangle, \langle \mathtt{message}, msg\rangle\rangle \\ K(e) &= K_v^{str} \end{aligned}$$

For example, given a publication $e = \langle\langle$`publisher`, $P\rangle$, $\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`name`, `andy`$\rangle$, $\langle$`message`, $msg\rangle\rangle$ we construct $K(e) = K_{\mathtt{andy}}^{\mathtt{name}}$.

Subscription. We construct an authorization key for a subscription as follows.

$$\begin{aligned} f &= \langle\langle \mathtt{topic}, EQ, w\rangle, \langle str, PF, u\rangle\rangle \\ K(f) &= K_u^{str} \end{aligned}$$

Given a publication with string v, a subscriber who has subscribed for a string u does the following. The subscriber checks if u is a prefix of v. If so, the subscriber derives the encryption key K_v^{str} from the authorization key K_u^{str}. Note that the generation of children keys from parent keys is computationally efficient because such computations use a fast one-way hash function. However, it is computationally infeasible for a subscriber to derive the keys corresponding to its ancestors or its

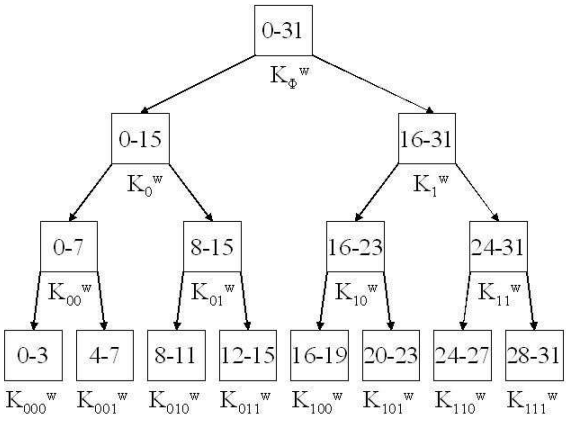

Fig. 5. Key Tree: Numeric Attributes − Range Queries

siblings. For example, given a publication with v = `andy`, a subscriber who has subscribed for u = `a` decrypts the message msg in a publication as follows. Given v = `andy` and u = `a`, the subscriber first extracts the suffix `ndy`. Then, S derives $K^{str}_{\texttt{an}} = KH_{K^{str}_{\texttt{a}}}(\texttt{n})$, $K^{str}_{\texttt{and}} = KH_{K^{str}_{\texttt{an}}}(\texttt{d})$, and $K^{str}_{\texttt{andy}} = KH_{K^{str}_{\texttt{and}}}(\texttt{y})$.

4.5 Complex Subscriptions

We have so far presented EventGuard techniques to handle category based subscriptions and string based prefix/suffix matching. We note that numeric attribute based matching can be achieved in a way that is analogous to string based prefix/suffix matching as shown in a preliminary version of our paper [Srivatsa and Liu 2007] (see Figure 5). However, we have so far dealt with subscriptions that consist of a topic and at most one constraint, say, $f = \langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $\in$, (0, 15)$\rangle\rangle$. A complex subscription could consist of constraints combined using the $\wedge$ and $\vee$ Boolean operators. In general a complex filter is represented as a complex subscription $f = \langle\langle$`topic`, EQ, $w\rangle$, $B(sf_1, sf_2, \cdots, sf_l)\rangle$ where each sf_i is a simple filter (only one constraint) and B is a monotone Boolean expression. An example of a complex filter could be f=$\langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, ($\langle$`age`, $\in$, (0, 15)$\rangle$ $\wedge$ $\langle$`gender`, EQ, `F`$\rangle$ $\wedge$ ($\langle$`news`, $\ni$, `secretNews1`$\rangle$ $\vee$ $\langle$`news`, $\ni$, `secretNews5`$\rangle$)$\rangle$. A matching event for the example subscription shown above could be $e = \langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`age`, 9$\rangle$, $\langle$`gender`, `F`$\rangle$, $\langle$`news`, `classifiedNews1`$\rangle$, $\langle$`message`, $msg\rangle\rangle$.

Preliminaries. Given a complex subscription $f = \langle\langle$`topic`, EQ, $w\rangle$, $B(sf_1, sf_2, \cdots, sf_l)\rangle$, we express B in disjunctive normal form (DNF) [Mathpages] as $B = \bigvee_{i=1}^{nd} D_i$, where $D_i = \bigwedge_{j=1}^{nsf} sf_j$. We then divide f into nd complex filters $\{f_1, f_2, \cdots, f_{nd}\}$, where $f_i = \langle\langle$`topic`, EQ, $w\rangle$, $D_i(sf_1, sf_2, \cdots, sf_l)\rangle$. The subscriber now subscribes independently for each of these nd subscription filters. Note that this is equivalent to the original subscription on the filter f since, $B = \bigvee_{j=1}^{nd} D_j$. For example, we divide a complex filter $f = \langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, ($\langle$`age`, $\in$, (0, 15)$\rangle$ $\wedge$ $\langle$`gender`, EQ, `F`$\rangle$ $\wedge$ ($\langle$`news`, $\ni$, `secretNews1`$\rangle$ $\vee$ $\langle$`news`, $\ni$, `secretNews5`$\rangle$)$\rangle$ into two filters $f_1 = \langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, ($\langle$`age`, $\in$, (0, 15)$\rangle$ $\wedge$ $\langle$`gender`, EQ,

F⟩ ∧ ⟨news, ∋, secretNews1⟩)⟩ and f_2 = ⟨⟨topic, EQ, cancerTrail⟩, (⟨age, ∈, (0, 15)⟩ ∧ ⟨gender, EQ, F⟩ ∧ ⟨news, ∋, secretNews5⟩)⟩. Note that the subscriber can subscribe for the filters f_1 and f_2 independently and receive all events e that match the filter $f = f_1 \vee f_2$. In the following portions of this section we describe techniques to derive keys for complex filters whose constraints include only the ∧ operator. The concrete technique for constructing subscriptions and publications using these derived keys is very similar to that discussed in numeric attribute based matching and category based matching and will be omitted in this section.

Publication. The encryption key for a complex event e_i = ⟨⟨topic, w⟩, ⟨$name_1$, $value_1$⟩, ⋯, ⟨$name_l$, $value_l$⟩⟩ is constructed as follows: $K(e_i) = H(\bigoplus_{j=1}^{l} K(se_j))$, where se_j = ⟨⟨topic, w⟩, ⟨$name_j$, $value_j$⟩⟩. For example, given an event e = ⟨⟨topic, cancerTrail⟩, ⟨age, 9⟩, ⟨gender, F⟩, ⟨news, classifiedNews1⟩, ⟨message, msg⟩⟩, the key $K(e)$ is computed as follows. We break up e_i into three simple events se_1 = ⟨⟨topic, cancerTrail⟩, ⟨age, 9⟩⟩, se_2 = ⟨⟨topic, cancerTrail⟩, ⟨gender, F⟩⟩ and se_3 = ⟨⟨topic, cancerTrail⟩, ⟨news, classifiedNews1⟩⟩. Then, we compute the encryption keys for the simple events using the techniques described in earlier sections: $K(se_1) = K_{010}^{\texttt{age}}$ (since $ktid(9) = 010$), $K(se_2) = K_F^{\texttt{gender}}$ and $K(se_3) = K_0^{\texttt{news}}$ (since $ktid(\texttt{classifiedNews1}) = 0$). Finally, we derive $K(e) = H(K_{010}^{\texttt{age}} \oplus K_F^{\texttt{gender}} \oplus K_0^{\texttt{news}})$.

Subscription. Now we describe how an authorization key is constructed for a filter f_i = ⟨⟨topic, EQ, w⟩, $D_i(sf_1, sf_2, \cdots, sf_l)$⟩. We divide the subscription filter f_i into l simple subscriptions of the form: f_{ij} = ⟨⟨topic, EQ, w⟩, sf_j⟩. The set of authorization keys associated with f_i is $\{K(sf_1), K(sf_2), \cdots, K(sf_l)\}$, where $K(sf_j)$ is the authorization key for a subscription filter f_{ij} using the techniques described in earlier sections. For example, given a subscription filter f_1 = ⟨⟨topic, EQ, cancerTrail⟩, (⟨age, ∈, (0, 15)⟩ ∧ ⟨gender, EQ, F⟩ ∧ ⟨news, ∋, secretNews1⟩)⟩, we split into three simple filters: f_{11} = ⟨⟨topic, EQ, cancerTrail⟩, ⟨age, ∈, (0, 15)⟩⟩, f_{12} = ⟨⟨topic, EQ, cancerTrail⟩, ⟨gender, EQ, F⟩⟩ and f_{13} = ⟨⟨topic, EQ, cancerTrail⟩, ⟨news, ∋, secretNews1⟩⟩. We then associate the following authorization keys with filter f: $K(f_{11}) = K_0^{\texttt{age}}$ (since $ktid(0, 15) = 0$), $K(f_{12}) = K_F^{\texttt{gender}}$, and $K(f_{13}) = K_{00}^{\texttt{news}}$ (since $ktid(\texttt{secretNews1}) = 00$). Given an event e = ⟨⟨topic, cancerTrail⟩, ⟨age, 9⟩, ⟨gender, F⟩, ⟨news, classifiedNews1⟩, ⟨message, $E_{K(e)}(msg)$⟩⟩, a subscriber derives key $K(e)$ as follows. The subscriber computes $K_{010}^{\texttt{age}}$ (since $ktid(9) = 010$) from $K_0^{\texttt{age}}$ (since $ktid(0, 15) = 0$) using the numeric attribute key tree for age, $K_0^{\texttt{news}}$ (since $ktid(\texttt{classifiedNews1}) = 0$) from $K_{00}^{\texttt{news}}$ (since $ktid(\texttt{secretNews1}) = 00$) using the category attribute key tree for news. The subscriber uses the authorization key $K_F^{\texttt{gender}}$ with the derived keys $K_{010}^{\texttt{age}}$ and $K_0^{\texttt{news}}$ to compute $K(e) = H(K_{010}^{\texttt{age}} \oplus K_F^{\texttt{gender}} \oplus K_0^{\texttt{news}})$. One can show that a subscriber can derive $K(e)$ from $K(f)$ if and only if the complex event e matches the complex filter f by our composing arguments presented in earlier sections.

4.6 Performance Enhancements

In this section we present two key caching mechanisms to enhance the performance of our key derivation algorithms: temporal key cache and semantic key cache. In a temporal cache, a key is cached with a hope that it is reused in the near future. A

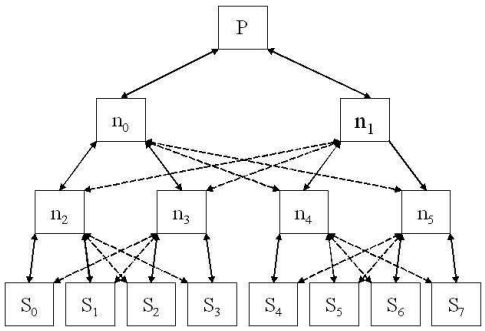

Fig. 6. Constructing Resilient Networks: Thick lines represent links in the binary tree network and the dashed lines represent additional links added to binary tree network to make its $ind = 2$

semantic key cache extends the temporal key cache by exploiting the functioning of our key derivation algorithms to enhance the system's performance.

Temporal Key Cache. A temporal key cache exploits the temporal locality in the events received by a subscriber. Caching the encryption key $K(e)$ saves the cost of computing $K(e)$ from $K(f)$. We use a simple least recently used (LRU) based cache replacement policy to maintain the temporal key cache.

Semantic Key Cache. The key idea behind the semantic key cache is to extend a temporal key cache using specific properties of our key derivation algorithm. Given an event e, the semantic key cache selects a cached authorization key $K(f_{opt})$ such that it is most efficient to derive $K(e)$ from $K(f_{opt})$. The optimal authorization key $K(f_{opt})$ for deriving the encryption key $K(e)$ is determined as follows. Given a filter f and an event e we define a distance between them as $dist(f, e)$. If the event e does not match the filter f, then $dist(f, e) = \infty$. If the event e matches the filter f, then we define $dist(f, e)$ as the computational cost incurred in deriving $K(e)$ from $K(f)$. We compute the optimal filter f_{opt} as $f_{opt} = argmin_{f \in C}\ dist(f, e)$, where C denotes the temporal key cache and $f \in C$ denotes a subscription filter f whose key $K(f)$ is cached in the temporal key cache C.

For example, for some numeric attribute num, let us suppose that the cache C consists of three keys $K_{\emptyset}^{num}$, K_0^{num} and K_1^{num}. Now we choose $K(f_{opt})$ to derive $K(e) = K_{000}^{num}$ as follows. First we observe that K_{000}^{num} can be derived only from keys $K_{\emptyset}^{num}$ and K_0^{num}. Computing K_{000}^{num} from $K_{\emptyset}^{num}$ requires three hash computations, while that from K_0^{num} requires two hash computations. Hence, we choose $K(f_{opt})$ $= K_0^w$.

5. EVENTGUARD: RESILIENT NETWORK GUARD

The six security guards discussed so far can achieve message authenticity, confidentiality, integrity, and protect the pub-sub network from flooding-based DoS attacks. In addition, per topic token helps to alleviate selective message dropping attacks. However, they are incapable of handling random message dropping based attacks and frequency based inference attacks on per topic tokens. In this section, we present techniques to restructure the pub-sub network in way that can effectively handle these attacks using multi-path event routing.

There are three important design goals in constructing a resilient pub-sub network: (i) the pub-sub network must be resilient to message dropping attacks, (ii) the pub-sub network must be resilient to inference attacks on routing tokens, and (iii) the communication cost should be minimal. We first discuss two network topologies that represent extremities of the spectrum and then describe the EventGuard solution. The first network topology is an $a-$ary tree topology. The second network topology mirrors the propagation scheme used in Byzantine fault tolerant information dissemination [Malkhi et al. 2001]. The $a-$ary tree topology incurs minimum communication cost but is not strongly resilient to message dropping attacks. The BFT propagation algorithm incurs very high communication cost and is highly resilient to message dropping attacks.

In this section, we proceed in three steps. First, we compare the communication cost between these two pub-sub network architectures. Second, we study the resilience of $a-$ary trees towards message dropping attacks. Third, we propose EventGuard network architectures that strike a trade-off between resilience to message dropping attacks and the communication cost.

5.1 Overview: Pub-Sub Network Architectures

Let NS denote the number of subscribers in the system and $N(w)$ denote the number of subscribers who have subscribed to topic w. In an $a-$ary tree network, we assume that each publisher corresponds to one $a-$ary tree and a publisher is the root of the tree, the subscribers who have matching subscriptions are the leaves of the tree and the pub-sub nodes are intermediate elements of the tree. The height h of the tree is given by $\lceil \log_a NS \rceil$. For simplicity we compute h by $h = \log_a NS$ and assume that h is rounded up to an integer. Let $M^{tree}(w)$ denote the communication cost (in terms of the number of messages) of propagating a publication on topic w from the publisher to the subscribers. Since the cost of sending the publication to any individual subscriber is lesser than or equal to h, the total cost $M^{tree}(w) \leq hN(w)$. Also, the publication message is never required to traverse any link of the tree more than once. Hence, $M^{tree}(w) \leq \Sigma_{i=1}^{h} a^i = \frac{a}{a-1}(NS-1)$. Combining the two constraints, we have $M^{tree}(w) \leq min(hN(w), \frac{a}{a-1}(NS-1))$. Observe that the maximum communication cost for an $a-$ary tree occurs when $N(w) = NS$ and $M_{max}^{tree}(w) = \frac{a}{a-1}(NS-1)$. $M_{max}^{tree}(w)$ is minimized when $a = NS$, that is, the publisher is directly connected to all the subscribers. In general, as the parameter a increases the expected communication cost decreases. However, the communication load on the publisher and pub-sub nodes increases with a since the publisher and the pub-sub nodes may have to forward an event to a children nodes.

The BFT propagation algorithm assumes that the number of malicious nodes (m) is known. A non-malicious node accepts an event e as an authentic event if and only if it receives $m+1$ identical copies of e from distinct $m+1$ nodes. The key idea is that in any set of $m+1$ nodes there is at least one non-malicious node, and thus if $m+1$ distinct nodes report an event e then e has to be authentic. In a BFT propagation scheme, each subscriber has to minimally receive $m+1$ identical publication messages, irrespective of the network topology (grid, tree) used for propagation. Hence the communication cost, denoted by $M^{bft}(w)$, satisfies the following condition: $M^{bft}(w) \geq (m+1)N(w)$. Assuming that $NS = 1000$, about

10% of the nodes are malicious, and $m = 100$, we have $M^{tree}(w) \leq min(5N(w), 1332)$ (assuming a 4−ary tree: $a = 4$ and $h = \log_4 1000 \approx 5$ and $\frac{a}{a-1}NS = 1332$) and $M^{bft}(w) \geq 101N(w)$. This implies that the communication cost in any BFT dissemination algorithm would be at least 20 times ($\approx \frac{101N(w)}{5N(w)}$) the a−ary tree based algorithm. However, one should note that the BFT dissemination is completely resilient to message dropping attacks and is *unconditionally secure* (requires no digital signatures). In a wide-area network with node-to-node latency on the order of 70ms [Zegura et al. 1996], it might be advisable to limit the communication cost while incurring additional signature verification cost (1-2ms per verification).

5.1.1 *Resilience to Message Dropping Attacks.* We have discussed the BFT propagation scheme and its complete resilience to message dropping attacks. In comparison, a simple a−ary tree-based network is vulnerable to a message dropping attack. A publication from the publisher successfully reaches a subscriber only if all the nodes on the routing path from the publisher to the subscriber are non-malicious. Let p denote the fraction of nodes that are malicious. Assuming that malicious nodes are randomly distributed on the network, the probability that a publication reaches a subscriber is $Pr(succ) = (1 - p)^h$. Even when $p = 10\%$, with $h = 5$ we find that the probability of a successful delivery of a publication is only 0.59. This implies that 10% malicious nodes are able to harm about 41% of the subscribers. One way to increase $Pr(succ)$ is to increase a (consequently decrease h). However, as we have pointed out earlier, as a increases the load on the publisher and the nodes on the pub-sub network increases, thereby harming the scalability of the system.

The key problem with the tree-based topology is that there is only one *independent path* from a publisher to a subscriber [Srivatsa and Liu 2004]. Informally, two paths Q_1 and Q_2 are independent if they share no node other than their source and their destination node. If we have *ind* independent paths between a publisher P and a subscriber S, then *ind* malicious nodes (one per independent path) could completely block any communication between P and S. The BFT propagation scheme uses $m + 1$ independent paths to propagate the publication thereby ensuring that at least one independent path is devoid of malicious nodes. Note that using an arbitrary peer-to-peer topology for the pub-sub network does not directly entail the existence of multiple independent paths [Srivatsa and Liu 2004].

5.2 Low Cost Resilient Pub-Sub Network

In pub-sub systems one may not require absolute guarantee of message delivery at all time. This permits us to tradeoff resilience with communication cost. We modify an a−ary tree such that it has *ind* independent paths while increasing the communication cost by not more than a factor of *ind* ($ind \leq a$). For simplicity, we illustrate our technique by modifying a binary tree network to yield a network with $ind = 2$.

Figure 6 shows the key idea behind constructing a resilient event dissemination networks G^2. Note that d refers to the depth of a node, with root (publisher) at depth 0 and the leaves (subscribers) at depth h. For any node n, let $parent(n)$ denote the parent of node n and $sibling(n)$ denote an immediate left or right sibling of node n. EventGuard's resilient network adds one additional edge to every

subscriber and every node in the system. Concretely, for every node n we add an additional edge from n to $sibling(parent(n))$. We now claim that the resilient network G^2 has the following property.

CLAIM 5.1. *The resilient network G^2 has $ind = 2$ independent paths from the publisher P to every subscriber in the system.*

We prove Claim 5.1 using Theorem 5.2 which explicitly constructs two independent paths from the publisher (root) to any subscriber (leaf) on the resilient network.

THEOREM 5.2. *Let $Q = \langle P, n_1, n_2, \cdots, n_d, S\rangle$ denote a path from the publisher P to some subscriber S in the original tree based network. Then, $Q_1 = Q$ and $Q_2 = \langle P, sibling(n_1), sibling(n_2), \cdots, sibling(n_d), S\rangle$ are two independent paths from P to S in the resilient network.*

PROOF. First, we show that the path Q_2 exists (path $Q_1 = Q$ exists trivially). We show that for any $1 \leq i \leq d$, there exists an edge from $sibling(n_i)$ to $sibling(n_{i+1})$. From path Q_1 we know that n_i is the parent of node n_{i+1}. Hence, n_i is the parent of node $sibling(n_{i+1})$. By the construction of our resilient network, we add an edge from any node n to $sibling(parent(n))$. Hence, $sibling(n_{i+1})$ is connected to $sibling(n_i)$ (since, $n_i = parent(sibling(n_{i+1}))$).

Second, we show that $\{n_1, n_2, \cdots, n_d\} \cap \{sibling(n_1), sibling(n_2), \cdots, sibling(n_d)\} = \emptyset$. First, for any $1 \leq i \leq d$, $n_i \neq sibling(n_i)$. Second, for any two nodes n_i and n_j $1 \leq i, j \leq d$ such that $i \neq j$, $n_i \neq n_j$ since the node n_i is at depth i from the root, while n_j is at depth j from the root ($i \neq j$). Hence, the paths Q_1 and Q_2 are independent. □

One can easily extend this network construction scheme for any $ind \leq a$. Construct a resilient nework G^{ind} by connecting any node n to $parent(n)$ and $ind-1$ distinct siblings of $parent(n)$ (these siblings indeed exist since $ind \leq a$).

CLAIM 5.3. *The resilient network G^{ind} has ind independent paths from the publisher P to every subscriber in the system.*

PROOF. The proof for Claim 5.3 follows the same lines as that for Claim 5.1. □

CLAIM 5.4. *The resilient network G^{ind} incurs ind times the communication cost of G^1.*

PROOF. The proof follows from the construction of independent paths in Theorem 5.2. □

As we increase ind, the communication cost increases by a factor ind. However, we believe that $ind = 2$ would suffice for most practical pub-sub networks. Assuming that the malicious nodes are randomly distributed on the network, the probability that publication reaches a subscriber is $Pr(succ) = 1 - (1 - (1 - p)^h)^{ind}$. With $p = 0.1$ and $ind = 2$ we would require $h \leq 3.66$ for $Pr(succ) \geq 0.9$. For a pub-sub network with each publisher having 1000 subscribers as the upper bound, this would translate to $a = 7$ (7-ary tree). On the other hand, achieving the same level of resilience with $ind = 1$ would require $h \leq 1$ and thus $a = NS$. Recall that as a increases, the load on the publisher and the nodes on the pub-sub routing path increases and affects the system's scalability. We demonstrate in the experiments

sections that our technique can be employed to construct a γ-resilient network with $Pr(succ) = \gamma$ by carefully choosing ind and a.

5.3 Secure Routing

In this section, we describe techniques for secure content-based routing using the multi-path routing infrastructure. We achieve secure content-based routing in two steps: tokenization and probabilistic multi-path event routing.

5.3.1 *Tokenization.* We first describe our techniques for tokenizing the routable `topic` attribute in an event so as to support content-based routing. We use the solution proposed by Song et al. [Song et al. 2000] for searches on encrypted data to construct our algorithm. Let us consider a topic w. The MS generates a token $T(w)$ for the topic w using a keyed hash function KH as $T(w) = KH_{rk(MS)}(w)$, where $rk(MS)$ is the MS's master key. The subscriber subscribes for a topic w using an authorization filter $S = \langle$`topic`, EQ, $T(w)\rangle$. When a publisher wishes to publish an event under topic w, it constructs a routable attribute for the event as: $\langle r,\ KH_{T(w)}(r)\rangle$, where r is a randomly chosen nonce. A node matches an event e with a routable attribute $\langle r, match\rangle$ against a subscription filter f with a tokenized constraint $\langle$`topic`, EQ, $tok\rangle$ by checking if $KH_{tok}(r) = match$. A proof of correctness is contained in [Song et al. 2000].

While tokenization allows content-based event routing, curious routing nodes may attempt to break the confidentiality of routable attributes in a pub-sub message using a *frequency inference attack*. For example, a routing node may observe the frequency of the events that match a given subscription filter. Using a priori knowledge about the frequency distributions of different events, a curious routing node can guess the topic embedded in an event. One should note that this attack applies only to the routable attributes and not the secret attributes in an event (since the secret attributes are encrypted). In the following section, we present probabilistic multi-path event routing as an effective technique to support secure content-based routing while minimizing the amount of information that can be inferred by the routing nodes.

5.3.2 *Probabilistic Multi-Path Routing.* One way to thwart the frequency inference attack is to route events from a publisher to its subscribers probabilistically using multiple independent paths such that the frequency of all tokens appear (nearly) indistinguishable for all the routing nodes in the pub-sub network. Two paths from a publisher P to a subscriber S are independent if they share no common node other than their end points (namely, P and S).

Let λ_t denote the actual frequency of a token t. We assume that the routing nodes can deduce λ_t for all tokens t using the underlying domain knowledge. We set the number of independent paths ind_t for routing an event with token t to be proportional to λ_t, say, $ind_t = \tau\lambda_t$ for some constant τ. Now, given an event with token t, the publisher P uniformly and randomly chooses one path amongst the set of ind_t independent paths. Every node in the routing path observes the apparent frequency of the token t $\lambda'_t = \frac{\lambda_t}{ind_t} = \frac{1}{\tau}$. Clearly, the apparent frequency of all tokens in the pub-sub system as observed by the routing nodes is a constant $\frac{1}{\tau}$. However, colluding routing nodes may be able to infer more information, especially

if the colluding nodes are on two or more independent paths from a publisher P to a subscriber S. In particular, if all the routing nodes collude with one another then $\lambda'_t = \lambda_t$. However, if the fraction of colluding nodes is smaller than one, then the apparent frequency as observed by the routing nodes may be sufficiently skewed to drastically constrain a large scale inference attack.

Consistent with other research works in this area, we use entropy as the metric for measuring the amount of leaked information [Perng et al. 2006]. The actual entropy of the system is measured as $S_{act} = -\sum_{t\in\Gamma}\lambda_t \log(\lambda_t)$, where the frequencies of tokens are normalized such that $\sum_{t\in\Gamma}\lambda_t = 1$. The entropy of the system as observed by the routing nodes is $S_{app} = -\sum_{t\in\Gamma}\lambda'_t \log(\lambda'_t)$, where λ'_t is the apparent frequency of tokens as observed by curious routing nodes normalized such that $\sum_{t\in Gamma}\lambda'_t = 1$. Ideally, if $\lambda'_t = \mathrm{c}$ for all $t \in \Gamma$, then S_{app} attains a maximum value $S_{max} = \log(|\Gamma|)$, where $|\Gamma|$ denotes the size of the set Γ. Hence, the lower the entropy S_{app} is, the less is its randomness and the higher the accuracy of an inference attack. Note that the entropy measure is independent of the exact nature of the inference algorithm used by the routing nodes.

6. EVENTGUARD EVALUATION

We have implemented EventGuard on top of an unmodified Siena pub-sub core [Carzaniga et al. 2001]. Siena is a content-based pub-sub system whose working is very similar to our reference model in Section 2.1. A unique feature of our design is that the nodes in the pub-sub network can route messages as if they were *original* Siena messages. Hence, no changes were required to the Siena pub-sub core (e.g., the content-based routing and event matching algorithms). This is because EventGuard uses the same in-network matching operators as those supported by the Siena pub-sub core. We have implemented the meta-service MS as a stand-alone entity. The MS computes the authorization keys on the fly since the key derivation cost is fairly low. For a MS with limited computing power, one could cache the derived keys to tradeoff computing power with main memory utilization. Our prototype implementation uses the following cryptographic algorithms. We use SHA1 for the hash function H, HMAC-SHA1 for the keyed hash function KH, and AES-128-CBC for the encryption algorithm E. For modular exponentiations in field Z_p, we use the standard exponentiation by squaring algorithm that computes the result in $O(\log_2 p)$ time.

We evaluate our EventGuard prototype implemented on the Siena pub-sub core in two steps. We first present some micro-benchmarks to quantify the overhead of EventGuard mechanisms and measure the performance and storage overheads at the MS, a publisher, a subscriber and a node. Then we present macro-benchmarks to quantify the overhead of the entire system including key management costs, MS load and scalability. We also present implementation based measurements on end-to-end throughput and latency of the pub-sub network as an effect of EventGuard mechanisms. We also quantify the effect of EventGuard's resilience to DoS attacks.

In this section, we present performance numbers from simulation based experiments on EventGuard. First, we experimentally measure the computation and communication cost of EventGuard's basic guards. Then, we present the improvements on message confidentiality and integrity through EventGuard. We also demonstrate

the resilience of EventGuard against flooding-based DoS attacks. We also quantify the average load on the MS, the publisher, the subscriber and the nodes as we vary the subscription and publication rate.

Simulation Setup. We used GT-ITM [Zegura et al. 1996] topology generator to generate an Internet topology consisting of 4K nodes. We linked these nodes using open TCP connections to form a binary tree based hierarchical topology. The latencies for links were obtained from the underlying Internet topology generated by GT-ITM. The round trip times on these links varied from 24ms to 184ms with mean 74ms and standard deviation 50ms. We simulated 32 publishers and NS=8K subscribers. The publishers and subscribers were randomly connected to one leaf node in the pub-sub network. We used discrete event simulation [FIPS] to simulate the function of the pub-sub network. All experimental results presented in this section were averaged over 5 independent simulation runs.

We simulated 128 topics, with the popularity of each topic varying according to a Zipf-like distribution [Qin Lv and Shenker 2002]. Each subscriber subscribed for 32 topics chosen from the set of 128 topics using the Zipf distribution. Amongst 128 topics, 32 were numeric attributes, 32 were category attributes, 32 were string attributes and the remaining 32 were simple topics. Numeric attributes had a range of size 256 units and a least count of 4 units; the subscription range was chosen using a Gaussian distribution with mean 128 and a standard deviation 32. Hence, the number of elements in the numeric attribute tree was 127 and the height of the numeric attribute tree was 6. Category trees were for height 4, and the number of children for each non-leaf element was chosen uniformly and randomly between 2 to 4. The average number of elements in a category tree was 82. The length of the string attributes were Zipf distributed between 1 and 8. Each publication message was assumed to be 256 Bytes long.

6.1 Micro-Benchmarks

In this section, we estimate the amount of computational and storage overhead due to EventGuard on the pub-sub system. All our measurements were made on a 900MHz Intel Pentium III running RedHat Linux 9.0 using Sun Java 1.5.0. Table I shows the amount of time it takes for executing different cryptographic primitives used by EventGuard. These times have been measured using the new *nanoTime* method introduced in J2SE 1.5.0. All reported values have been averaged over 1000 measurements. Note that the computation time for hash computation (SHA1 and HMAC-SHA1) depends on the block size. The larger the block size is, the faster is its hash computation rate. For instance, SHA1 hashes can be computed at 2 MB/s when the block size is small (16 Bytes) and about 57 MB/s for large block size (1024 Bytes). Similarly, AES-128-CBC can encrypt data at 10 MB/s, and the ElGamal algorithm can sign 714 16-Byte blocks per second and verify 588 signatures per second.

We experimentally measured the computational time for subscriptions, publications and unsubscriptions. The cost for an advertisement is very similar to that of subscriptions and the cost for unadvertisement is equivalent to that of an unsubscription. We analyzed the cost of these operations at all four entities: a publisher, a subscriber, the MS and a pub-sub node. We also analyzed the messaging and

H	SHA1	2 MB/s-57 MB/s
KH	HMAC-SHA1	1.6 MB/s-51 MB/s
E	AES-128-CBC	10 MB/s
sig	ElGamal-512-sign	911 Sign/s
sig	ElGamal-512-verify	668 Verify/s

Table I. Computation Times for Cryptographic Primitives used by EventGuard

	MS (ms)	publisher (ms)	subscriber (ms)	node (ms)
subscribe	$1.4 + 0.0012 * \|w\|$	-	1.7	1.7
unsubscribe	3.2	-	1.7	1.7
publish	-	$1.4 + (\|pbl\| + 16m) * 1.17 * 10^{-4}$	$1.7 + (\|pbl\| + 16) * 1.17 * 10^{-4}$	1.7
advertise	$1.4 + 0.0012 * \|w\|$	1.7	-	1.7
unadvertise	3.2	1.7	-	1.7

Table II. Computation Overheads for EventGuard Operations: w is some topic, pbl is a publication, and m denotes the number of topics marked on message pbl

subscription (Bytes)	unsubscription (Bytes)	publication (Bytes)	advertisement (Bytes)	unadvertisement (Bytes)
128	128	$128 + 16m$	128	128

Table III. Message Size Overhead due to EventGuard including only those messages sent on the pub-sub network: m denotes the number of topics marked on the publication

storage cost at these four entities. Tables II, III, IV and V summarize the results obtained in this section.

Subscription. The cost of a subscription at the MS includes the computation of key $K(w)$, token $T(w)$, special token $UST(w)$, an ElGamal signature on $T(w)$ and the current timestamp ts. Since, the topic w is typically a short string, the cost of computing the key $K(w)$ (using HMAC-SHA1) is $0.67 * |w|\mu$s. The cost of computing token $T(w)$ from $K(w)$ (using SHA1) is $0.5 * |w|\mu$s. The cost of computing special token $UST(w)$ (using HMAC-SHA1) is $0.67 * |sig_r|\mu\text{s} = 42.9\mu$s, where sig_r denotes the r-component of the MS's ElGamal signature (note that $|sig_r| = 512$ bits = 64 Bytes). The cost of computing an ElGamal signature is 1.4ms. Hence, the total cost per subscription topic (dominated by the signature computation time) is about 1.4ms + $1.2 * |w|\mu$s.

The cost of a subscription at the subscriber includes only the signature verification time. The cost of verifying an ElGamal signature is about 1.7ms. The cost of a subscription at a node in the pub-sub network is the cost required to process this subscription, which equals the sum of the cost of verifying the MS's signature and the cost of detecting duplicate identifiers to protect the pub-sub network from subscription-flooding based DoS attacks. Our experiments show that the cost of verifying duplicates is negligible ($< 10\mu$s) when compared to the signature verification time (1.7ms). Furthermore, our experiments show that the cost of processing a subscription at a node in EventGuard is only marginally higher than basic Siena ($< 50\mu$s). Note that a publisher incurs no direct cost for a subscription.

Unsubscription. The cost of an unsubscription at the MS includes the verification of special token $UST(w)$, the verification of MS's signature on the corre-

MS (Bytes)	publisher (Bytes)	subscriber (Bytes)	node (Bytes)
64	180 per adv + HT_{size}	180 per sub + HT_{size}	HT_{size}

Table IV. Storage Overhead: HT_{size} denotes the total size of the hashtable maintained for detecting flooding based DoS attacks. Our experiments use a hashtable of size 1 MB; a node can handle 1000 messages per second and store message identifiers over the last one minute using a storage space of 1000*60*4 bytes = 240KB

	topic	numeric	string	category
key derivation cost	1.03μs	5.51μs	2.28μs	0.56ms

Table V. Average key derivation overhead for various filters: topic (equality), numeric (range), string (prefix/suffix) and category (hierarchical sets)

sponding subscription token and the generation of an unsubscription permit. The cost of verifying the special token requires the computation of one keyed hash on the r-component of the MS's signature. As shown in the case of subscription, this costs 42.9μs. The cost of verifying an MS's signature adds 1.7ms, and the cost of generating an unsubscription permit adds 1.4ms. Hence, the total cost of an unsubscription at the MS is 3.2ms.

The cost of an unsubscription at a subscriber includes only the signature verification time, which costs 1.7ms. The cost of a unsubscription at a node in the pub-sub network is the cost required to process an unsubscription, which can be computed by the sum of the cost of verifying the MS's signature and the cost of detecting duplicate identifiers to protect the network from DoS attacks based on unsubscription-flooding. Our experiments show that the cost of verifying duplicates is negligible when compared to the signature verification time.

Publication. The cost of a publication at its publisher includes the cost of generating a random symmetric key K_r, the cost of encrypting the publication pbl with some random key K_r and the cost of encrypting K_r with $K(w_i)$ for every topic w_i $(1 \leq i \leq m)$ marked on the publication. The total encryption time is $(|pbl| + m * |K_r|) * 0.1\mu\text{s} = (|pbl| + 16m) * 0.117\mu\text{s}$ (note that $|K_r| = 16$ Bytes). Computing the publisher's signature adds an additional 1.4ms.

The cost of a publication at a subscriber includes the cost of checking the publisher's signature, the cost of decrypting the random key K_r and the cost of decrypting the message using key K_r. The total decryption time is $(|pbl| + |K_r|) * 0.117\mu\text{s}$ $= (|pbl| + 16) * 0.117\mu\text{s}$ (note that $|K_r| = 16$ Bytes). Verifying the publisher's signature adds an additional 1.7ms.

The cost of a publication at a node includes only the signature verification time. Similar to subscription, our experiments show that the cost of processing a publication at a node in EventGuard is only slightly higher than the cost of using basic Siena. Note that the MS is not involved directly in the publication process. This largely reduces the aggregate load on the MS as publications are considered by many applications as the most common operation on a pub-sub network.

Messaging Overhead. We now study the overhead added in terms of the length of a message due to EventGuard. For subscriptions and advertisements, the primary overhead is due to the MS's signature which is about 128 Bytes.

For publications, EventGuard adds the following overheads. First, the publisher's

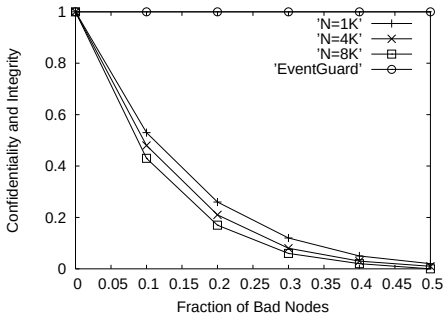

Fig. 7. Confidentiality and Integrity

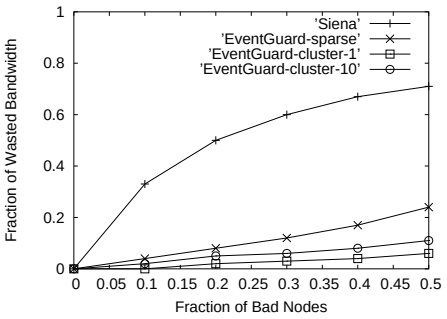

Fig. 8. Flooding-based DoS Attack

signature costs 128 Bytes. Second, the encrypted random key costs 16 Bytes. Third, the random key is encrypted by the topic's encryption key. This adds 16 Bytes for every topic included in the publication. The aggregate publication overhead may turn out to be quite significant if the published message is itself very small. On the other hand even if the published message is of the order of a few KBytes, the relative overhead added due to EventGuard turns out to be extremely small.

Storage Overhead. EventGuard requires publishers, subscribers and the MS to store additional information such as keys and tokens. The MS has the least storage overhead as it is required to store only the secret key $rk(MS)$ (about 64 Bytes).

A subscriber has to store the key $K(w)$ (16 Bytes), the token $T(w)$ (16 Bytes), the special token $UST(w)$ (16 Bytes), subscription time stamp ts (4 Bytes) and the MS's signature (128 Bytes) for each subscription token w. Thus, the total per topic storage overhead is 180 Bytes. Furthermore, the subscriber maintains a hashtable to store the publication identifiers (the r-component of the publisher's signature) in the near past (max_delay) to detect flooding based DoS attacks. The size of this hashtable obviously depends on the number of publications received by the subscriber in the last max_delay time units. Our experiments show that this hashtable is typically small, with its size ranging from 100 Bytes to a few KBs.

The storage overhead at a publisher is very similar to the storage overhead at a subscriber. However, the subscription identifier based hashtable maintained at the publisher is typically much smaller ($<$ 1 KB) than the publication identifier based hashtable at the subscribers, since the number of subscriptions $\ll$ number of publications. A node in the pub-sub network maintains two hash tables, one for subscriptions and one for publications for detecting flooding based DoS attacks. Our experiments show that the size of the subscription identifier based hashtable is usually very small ($<$ 1 KB), and the size of the publication identifier based hashtable is at most a few hundred KBs.

In summary, the performance overhead added by EventGuard is mostly dominated by digital signatures (2ms). However, in a wide-area network where the network latencies are on the order of 70ms [Zegura et al. 1996], the percentile overhead added by EventGuard is significantly smaller.

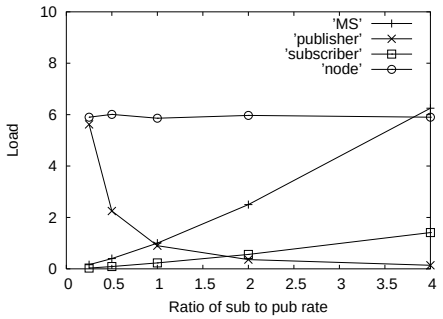

Fig. 9. Load

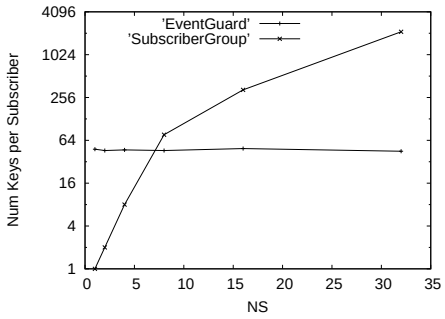

Fig. 10. Num Keys per Subscriber

6.2 EventGuard: Basic Guards

Confidentiality and Integrity. Figure 7 shows the fraction of messages that violate their confidentiality and integrity when in transit between a publisher and its subscribers with different fractions of malicious nodes (p) and different values of NS (number of subscribers). We assume that a message looses its confidentiality and integrity as soon as it transits one bad node in the pub-sub network. Observe that when p is small, even a small increase in p results in a heavy loss of message confidentiality and integrity. Note that as NS increases, the height of the binary tree network increases and so does the probability that at least one bad node appears on a path from a publisher to its subscribers. On the contrary, EventGuard is capable of preserving the confidentiality and integrity of all messages irrespective of the number of malicious nodes in the system.

Flooding-based DoS Attack. Figure 8 shows the fraction of network bandwidth expended on flooded messages as the fraction of malicious nodes (p) varies with NS=8K subscribers. We assume that every malicious node performs a publication flooding-based DoS attack at the rate of 100 messages per unit time. We assume that each publisher publishes at the rate of 25 publications per unit time. We consider two cases: Case one wherein the malicious nodes are uniformly distributed throughout the pub-sub network (EventGuard-sparse in Figure 8); and Case two wherein malicious nodes form k clusters in the pub-sub network (EventGuard-cluster-k in Figure 8). When malicious nodes are clustered together on the pub-sub network, we found that the loss in throughput for EventGuard is much smaller. This is because EventGuard ensures that no flooding attack propagates beyond one non-malicious pub-sub node. Hence, if the malicious nodes are bunched together, they cannot significantly affect other non-malicious nodes in the system. Recall from Figure 3 that no flooding by either of the two malicious nodes B1 or B2 propagates beyond non-malicious nodes G1, G2, G3 and G4. Observe that if B2 were attached to some other part of the pub-sub network, then it could perform a flooding based DoS attack on a different set of non-malicious neighbor nodes that does not overlap with that of B1.

Load. Table VI shows the number of subscription/advertisement requests that can be handled by our implementation of MS. Table VI also shows the ability of EventGuard to scale linearly with the number of MS. We note that in this

Number of MS	1	2	4	8	16	32	64
Subscriptions per second	914	1828	3678	7320	14821	29305	43872
Aggregate network traffic (Mbps)	1	1.98	4	8.03	16.11	33.51	45.20

Table VI. MS Scalability

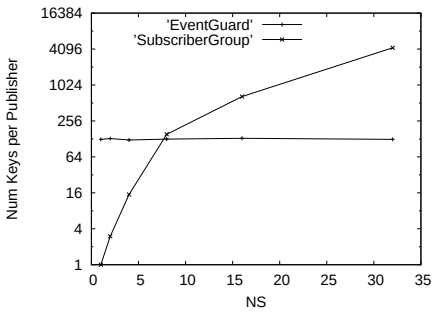

Fig. 11. Num Keys per Publisher

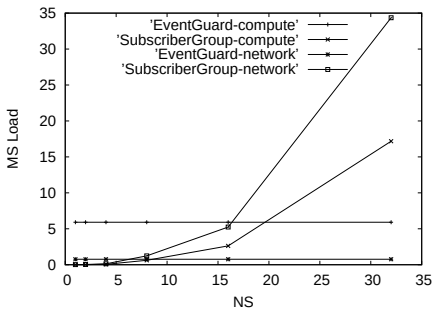

Fig. 12. Key Management Load

experiment all replicas of the MS shared a common 100Mbps wired network. We recall from table III that the size of a subscription/advertisement message is 128 Bytes (excluding network headers). Hence, about 1k subscriptions/advertisements per second would consume 128KBps $\approx$ 1Mbps. Hence, even with 16 replicas of MS, the network traffic does not become the bottleneck. In practice we required 38 replicas of MS before noticing a drop in performance.

Figure 9 shows the relative computational load on the MS, the publisher, the subscriber and a pub-sub node as we vary the rate of subscriptions, unsubscriptions and publications keeping the aggregate rate a constant (we do not consider advertisement and unadvertisement costs in this experiment). We set the subscription rate to be equal to the unsubscription rate so as to ensure that the average number of active subscriptions in the system is almost a constant. Note that only the control operations on subscriptions and unsubscriptions involves the MS. Hence, if a pub-sub network is largely dominated by publications (which is true in most cases) then the relative load on the MS would be very small. If the load on a MS is not acceptable, EventGuard mechanisms easily permit one to add additional meta-servers. The fact that the meta-servers do not have to interact with one another makes it possible for one to build an efficient *load balancing* system to handle the MS load and vary the number of active meta-servers *on-demand*.

Observe that the load on a node remains almost a constant as it depends only on the aggregate rate of subscriptions, unsubscriptions and publications. On the other hand the relative load on a publisher decreases as the publication rate decreases; this is because a publisher is not involved in subscribe and unsubscribe operations. Subscriber load is typically much smaller than the average node load because the number of publications delivered to a subscriber is very small when compared to the total number of publications sent on the pub-sub network. Recall that only those publications that match a subscriber's subscriptions are delivered to the subscriber.

6.3 EventGuard: Key Management

This section compares our key management algorithms with the subscriber group based approach in terms of the number of keys, communication and computation cost.

Number of Keys. Figure 10 shows the average number of keys maintained per subscriber as the number of subscribers NS varies. Recall that the `SubscriberGroup` approach uses group key management techniques on subscriber groups [Opyrchal and Prakash 2001] that require 2^{NS} keys in the worst case. EventGuard requires a small and constant number of keys per subscriber that is independent of NS. Even for 32 subscribers, the number of keys per subscriber using the `SubscriberGroup` approach is about 40 times larger than the EventGuard approach. EventGuard achieves significant reduction in the number of keys, while incurring a computational overhead for running the key derivation algorithms on the publisher and the subscribers. In our later experiments we show that the cost of key derivation is very small compared to wide-area network latencies thereby making it easily affordable. Figure 11 shows the average number of keys maintained per publisher as NS, and the number of subscribers varies. The trends shown in Figure 11 are very similar to that in 10.

Key Management Load. Figure 12 shows the computing and network cost on the key server using the `SubscriberGroup` based approach and `EventGuard`. Computing cost (measured in milliseconds) shows the average cost of group key management in `SubscriberGroup` and the cost of key derivation in `EventGuard` when a new subscriber joins the system. The cost incurred by the `SubscriberGroup` increases dramatically with NS, while that incurred by the `EventGuard` approach is a small constant that is independent of NS. Networking cost (measured in KBytes) shows the average cost of communicating the updated group key in `SubscriberGroup` and the cost of delivering the authorization keys in `EventGuard`. Similar to computing cost, `EventGuard` incurs a small and constant networking cost, while that of `SubscriberGroup` explodes with NS.

Key Cache. From our experiments on throughput and latency, we measured the overhead due to encryption/decryption and key derivation. We observed that the overhead due to encryption/decryption and key derivation for topics and numeric attributes was very low. One could additionally reduce this overhead using key caching on the authorization service MS, the publishers and the subscribers. Figure 13 shows the throughput and latency in a pub-sub network with one publisher, 30 nodes and 32 subscribers for different values of cache size. Observe that when all authorization keys are cached, the encryption/decryption cost becomes the primary overhead for EventGuard. Using the key caching mechanism, the throughput of EventGuard was about 2.2% (as against 10.8% without caching) lower than Siena, and the latency of EventGuard was about 1.5% (as against 5.7% without caching) higher than Siena (using a 64 KB cache).

6.4 EventGuard: Resilient Network

This section presents experimental results on EventGuard's resilient pub-sub network. First, we measure the cost of constructing and routing on EventGuard's resilient network. Second, we measure the efficacy of probabilistic multi-path event

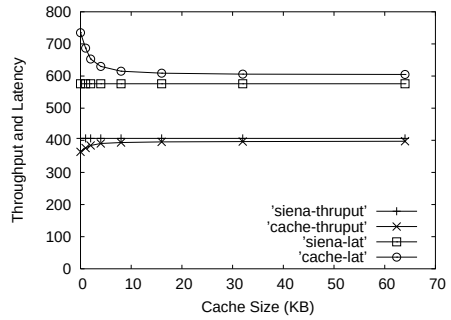

Fig. 13. Key Caching

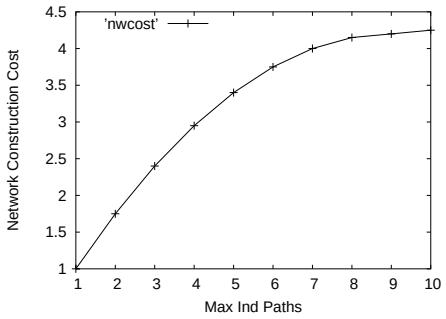

Fig. 14. Cost of Constructing a Multi-Path Event Routing Network

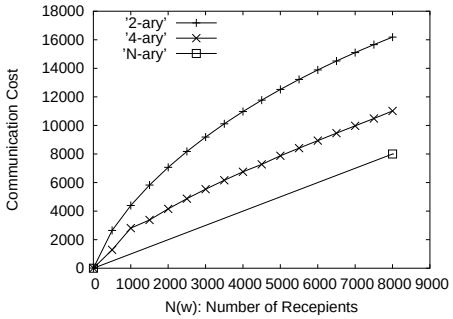

Fig. 15. Communication Cost Vs Number of Recepients $N(w)$

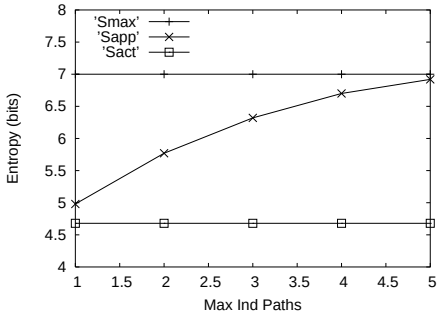

Fig. 16. Secure Content-Based routing under a Non-Collusive Setting

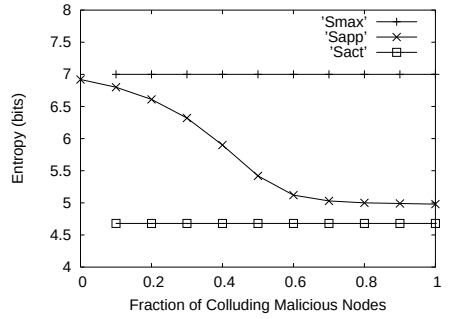

Fig. 17. Secure Content-Based routing under a Collusive Setting

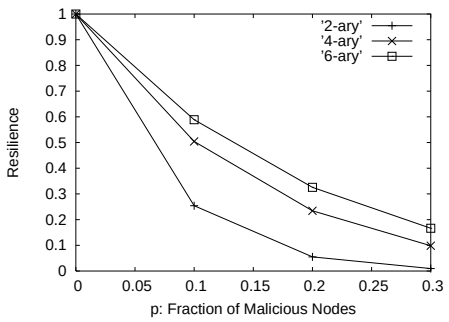

Fig. 18. Resilience Vs a with $ind = 1$

routing in defending against frequency inference attacks. Third, we measure the resilience of multi-path event routing in defending against random and selective dropping attacks.

Multi-Path Network Construction Cost. Figure 14 shows the cost of con-

structing a pub-sub network for different values of maximum number of independent paths (ind_{max}). The values shown in Figure 14 have been normalized against the construction cost for $ind_{max} = 1$. Observe that the construction cost saturates with the maximum number of independent paths. This is because only frequently occurring tokens are routed through a large number of independent paths. Hence, even when ind_{max} is 10, most of the tokens are routed through a smaller number of independent paths; only the most popular 12 tokens (out of 128 tokens) use all 10 independent paths, while 48 tokens (out of 128 tokens) used fewer than two independent paths. Observe from Figure 14 that the cost of constructing a pub-sub network with $ind_{max} = 5$ is about three times the cost of constructing a pub-sub network with $ind_{max} = 1$. Note that while probabilistic multi-path event routing incurs higher construction cost, it incurs no additional overhead for actually routing events on the pub-sub network.

Multi-Path Network Routing Cost. Figure 15 shows the communication cost for publishing an event under topic w versus $N(w)$ for different values of a with $ind = 1$, where $N(w)$ denotes the number of subscribers for topic w. Note that a resilient network constructed by modifying an $a-$ary tree increases the communication cost by a factor ind (for some $1 < ind \leq a$). Hence, the communication cost for an $a-$ary ind independent path network can be obtained by simply multiplying the corresponding cost for an $a-$ary tree network by ind. Observe that the communication cost decreases as a increases. Also note that $a = NS$ minimizes the communication cost but imposes heavy load on the publisher and the pub-sub nodes (load is proportional to a).

Non-Collusive Routing Nodes. We measured the efficacy of probabilistic multi-path event routing in maintaining the confidentiality of routable attributes in an event. Under a non-collusive setting, no two nodes share any inferred information amongst each other. The x-axis in Figure 16 shows the maximum number of independent paths permitted by the pub-sub network topology. Ideally, we want the maximum number of independent paths to be equal to $ind_{max} = \frac{max_{t\in\Gamma}\lambda_t}{min_{t\in\Gamma}\lambda_t}$. Assuming a Zipf distribution over 128 tokens this max-min ratio could be 128. However, the cost of constructing the network topology increases with the number of independent paths. From a more pragmatic standpoint, we limit the maximum number of independent paths between a publisher and its subscribers to five. Increasing the number of independent paths allows us to smooth out the apparent frequency of tokens observed by the routing nodes. The Figure also shows the maximum entropy (`Smax`) and the actual entropy of tokens (`Sact`). Even when $ind = 1$, then the entropy of the apparent frequencies as observed by the routing nodes (`Sapp`) is higher than the actual entropy (`Sact`). By the distributed nature of the pub-sub network, a node on the network may not be able to observe the frequency of all the tokens routed on the network. Hence, even without multiple independent paths, S_{app} is higher than S_{act}. Further, as ind increases, the entropy of information available to routing nodes increases (and thus, the effectiveness of their inference decreases). With $ind_{max} = 5$ independent paths, the apparent entropy S_{app} is within 10% of the maximum entropy S_{max}.

Collusive Routing Nodes. Figure 17 shows the efficacy of probabilistic multi-path event routing against collusive routing nodes. As the fraction of collusive

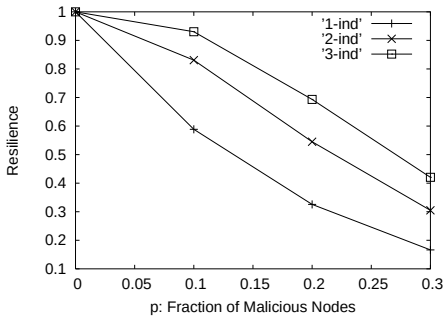

Fig. 19. Resilience Vs *ind* with $a = 6$

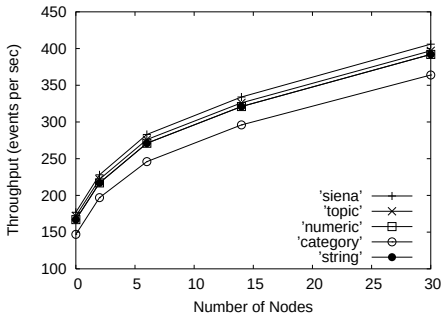

Fig. 20. Throughput

nodes increases, it is more likely that two or more colluding nodes are on two or more independent paths between the publisher and the subscriber. Observe that the entropy decreases as the fraction of collusive nodes increases. In fact, when all the routing nodes collude with one another, the entropy of their observation is equal to the actual entropy of the system (`Sact`). In a more realistic scenario wherein the fraction of colluding nodes is small (10-20%), the apparent entropy (`Sapp`) is significantly higher than the actual entropy (`Sact`), thereby significantly limiting the effectiveness of an inference attack.

Selective and Random Dropping Attack. We now report the experimental results on the effectiveness of using the r-resilient pub-sub networks against message dropping attacks. Our first experiment measures communication cost versus a (for an $a-$ary tree network). The second and third experiments measure the network resilience as a function of p (the fraction of malicious nodes in the network).

Figures 18 and 19 show the resilience of the pub-sub network versus p (fraction of malicious nodes) for different values of a and *ind* respectively. Resilience is measured in terms of the ratio of the number of susbcribers that receive an event on topic w to $N(w)$, averaged over all topics. Observe from Figure 18 that one can improve resilience by increasing a at the cost of publisher load. This is equivalent to decreasing the network's height h, thereby making the network shallow and broad. Figure 19 shows that one can improve resilience by increasing *ind* at the cost of the overall communication cost. A careful selection of parameters *ind* and a is required to strike a balance between resilience, communication cost and publisher load.

Comparison with Random Walk and Broadcast based Event Dissemination Schemes. Table VII compares our event dissemination protocol with the random walk and broadcast based event dissemination protocol. Random walk based protocols are ill-suited for event dissemination primarily because the average number of hops required to reach a subscriber from a publisher is $O(N \log N)$, where N is the number of routing nodes in the network. Also, with longer paths, the odds that at least one node in the path is malicious (and thus drops the message) is very high. Table VII shows that one can improve the random walk protocol by using multiple random walkers that simultaneously start from a publisher. On the other end of the spectrum is the broadcast protocol which achieves the lowest latency and maximum resilience to message dropping costs. However, the broadcast based

Event Dissemination	Latency	Resilience	Communication Cost
EventGuard (ind = 1)	532ms	0.6	750
EventGuard (ind = 2)	534ms	0.82	1208
EventGuard (ind = 3)	540ms	0.97	1732
RandomWalk (walkers = 1)	74s	$3.7*10^{-4}$	6415
RandomWalk (walkers = 2)	67s	$7.3*10^{-4}$	12012
RandomWalk (walkers = 3)	59s	$1.1*10^{-3}$	17091
Broadcast	522ms	0.98	12987

Table VII. Comparison of Event Dissemination Protocols: 10% of routing nodes perform message dropping attacks

scheme floods the entire pub-sub network, thereby incurring significantly higher costs than EventGuard. EventGuard can be very close to the broadcast based scheme in terms of latency and resilience, while reducing the communication cost by a factor of at least seven.

6.5 Implementation based Experiments

In this section, we present end-to-end performance measurements from our prototype implementation of EventGuard on Siena pub-sub core. First, we present measurements on the loss in throughput and the increase in latency in publications due to EventGuard. Second, we measure the throughput of EventGuard mechanisms under flooding based DoS attacks.

Experimental Setup. Our implementation of EventGuard is built on top of Siena pub-sub core. We ran this implementation of EventGuard on eight machines each with eight processors (550 MHz Intel Pentium III Xeon processors running RedHat Linux 9.0) connected via a high speed LAN. We simulate the wide-area network delays obtained from the GT-ITM topology generator. We ignored the LAN delays as they measured only a few tenths of a millisecond.

We used GT-ITM [Zegura et al. 1996] topology generator to generate an Internet topology consisting of 63 nodes. The latencies for links were obtained from the underlying Internet topology generated by GT-ITM. The round trip times on these links varied from 24ms to 184ms with mean 74ms and standard deviation 50ms. The tree's root node acts as the publisher, and its leaf nodes act as subscribers for this pub-sub network (32 subscribers and one publisher). We constructed complete binary tree topologies using different numbers of nodes (0, 2, 6, 14, 30) and linked these nodes using open TCP connections to form the pub-sub network. The subscribers were uniformly distributed among all the leaf nodes.

Throughput. We measured the throughput in terms of the maximum number of publications per second that can be handled by the pub-sub system with and without EventGuard (EventGuard-nosig). We measured the maximum throughput as follows. We engineered the publisher to generate publications at the rate of q publications per unit time. In each run of this experiment, the rate q was fixed. We monitored the number of outstanding publications required to be processed at every node. If at any node the number of outstanding publications monotonically increased for five consecutive observations, then we concluded that the node was saturated and the experiment was aborted. We iteratively varied q across different experimental runs to identify the minimum value of $q_{min} = throughput$ such that some node in the pub-sub network was saturated.

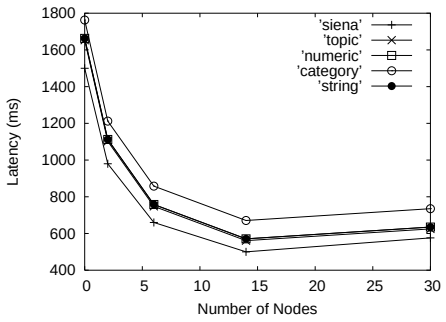

Fig. 21. Latency

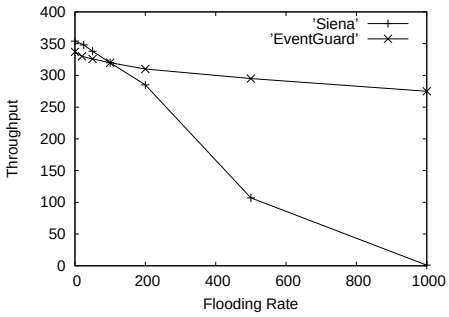

Fig. 22. Resilience to Flooding-based DoS Attacks

Figure 20 shows the maximum throughput versus the number of nodes in the pub-sub network for EventGuard and basic Siena for simple subscriptions. The increase in throughput with the number of nodes shows the scalability of EventGuard. Note that as the number of nodes increases, the number of subscribers connected to one leaf node decreases, thereby increasing the effective throughput. However, as the number of nodes becomes increasingly larger than the number of subscribers, the throughput does not increase any further, since this simply results in underutilized nodes. The main overhead in EventGuard arises due to the verification of ElGamal signatures (1.7ms). We also measured the overhead in the absence of this signature verification at every node in the pub-sub network (EventGuard-nosig in Figure reftab-thruput). We found that the overhead was lesser than 5%. We are currently exploring faster signature algorithms to replace ElGamal.

Latency. We measured latency in terms of the amount of time it takes from the time instant a publication is published till the time it is available to the subscriber (in plain-text). The latency was measured keeping the throughput at its highest (see Figure 20). Figure 21 shows latency versus number of nodes for EventGuard and basic Siena.

Observe that the latency first decreases and then increases. Initially, as the number of nodes increases, the number of subscribers assigned to each leaf node decreases. This consequently decreases the load on a node and thus decreases the latency. However, as the number of nodes increases, so does the height of the dissemination tree. An increase in height by one incurs an additional latency of 70ms (network latency), thereby increasing the overall latency. While the throughput always increases (until it saturates) with the number of nodes, the latency will begin to increase. This requires a careful choice of the number of pub-sub nodes in order to achieve high throughput with acceptable latencies. Observe from Figure 21 that the increase in latency due to EventGuard is very small. This is because the wide-area network latencies are of the order of 70ms while the overhead added at every node by EventGuard is about 2ms. Nevertheless, the maximum increase in latency due to EventGuard is less than 4%.

Flooding-based DoS Attacks. We measured the effect of flooding-based DoS attacks on the throughput of the pub-sub network. We picked one of the leaf nodes

to flood the pub-sub network. We vary fl, the rate that which the malicious node floods messages on the pub-sub network. Figure 22 shows the throughput as fl increases both in the presence and absence of EventGuard mechanisms to guard the system from flooding-based DoS attacks.

Observe from Figure 22 that in the absence of our guards, the pub-sub system deteriorates drastically with the injection of flooding-based DoS attack. In comparison EventGuard shows a more graceful drop in throughput as the flooding rate fl increases. Note that although our guard against flooding-based DoS attacks involves an expensive ElGamal signature check (1.7ms), it restricts the attack into a small neighborhood surrounding the malicious node (see Figure 3). This ensures that the effect of a flooding-based DoS attack is localized and that the rest of the pub-sub network is not affected by it.

7. RELATED WORK

Several pub-sub systems [Carzaniga et al. 2001][Banavar et al. 1999][Datta et al. 2003] have been developed to provide highly scalable and flexible messaging support for distributed systems. Siena [Carzaniga et al. 2001] and Gryphon [Banavar et al. 1999] are large pub-sub system capable of content-aware routing. Scribe [Datta et al. 2003] is an anonymous P2P pub-sub system. Most work on pub-sub systems has focused on performance, scalability and availability. Unfortunately, very little effort has been expended on studying the security aspects of these systems.

A significant amount of work has been done in the field of secure group communication on multicast networks (survey [Rafaeli and Hutchison 2003]). Such systems can leverage secure group-based multicast techniques and group key management techniques to provide forward and backward security, scalability and performance. The key problem in such systems arises due to the fact that IP multicast does not provide any mechanisms for preventing non-group members to have access to group communication. A significant restriction with secure group communication is that the group membership has to be predefined. In contrast, EventGuard permits flexible membership at the granularity of subscriptions. Second, EventGuard uses an overlay network and does not rely on IP multicast technology primarily because there have not been Internet scale deployments of the IP multicast protocol.

Wang et al. [Wang et al. 2002] analyze the security issues and requirements in a content-based pub-sub system. This paper identifies that the general security needs of a pub-sub application include confidentiality, integrity and availability. More specifically they identify authentication of publications, integrity of publications, subscription integrity and service integrity as the key issues. The paper presents a detailed description of these problems in the context of a content-based pub-sub system, but fails to offer any concrete solutions.

Raiciu et. al. [Raiciu and Rosenblum 2006] define security models and develop provable security algorithms for event confidentiality in content-based publish-subscribe networks. The paper describes theoretical limits and tradeoffs between confidentiality and generality of subscriptions and presents constructive schemes for (in)equality and range subscriptions.

Opyrchal and Prakash [Opyrchal and Prakash 2001] analyze secure distribution of events in a content-based pub-sub network from a group key management stand-

point. They show that previous techniques for dynamic group key management fail in a pub-sub scenario since every event can potentially have a different set of interested subscribers. They use a key caching based technique that relies on subscription popularity to reduce the number of encryptions and to increase message throughput. However, their approach requires that the pub-sub network nodes (brokers) are completely trustworthy. EventGuard aims to provide security to the subscribers while maintaining confidentiality even from the pub-sub network nodes.

Perng et al. [Perng et al. 2006] have proposed a mix network based technique to provide publisher/subscriber anonymity against curious routing nodes that have a priori knowledge on event popularity. Note that event popularity is defined as the number of subscribers that are interested in an event. Our secure event routing algorithm complements their proposal by defending against curious routing nodes that have a priori knowledge on the frequency distribution of events. In addition, they do not focus on access control and authorization on the published events.

Several authors have used hierarchical key derivation algorithms [Wong et al. 2000] to develop key management algorithms primarily in the domain of file systems [Atallah et al. 2005][Atallah et al. 2007b][Atallah et al. 2007a]. To the best of our knowledge this is the first paper that applies hierarchical key derivation algorithms to enforce access control in pub-sub systems. However, our solutions do not apply to all pub-sub matching operators, although they cover most of the popular ones [Carzaniga et al. 2001]. One solution is to use computation and communication intensive secure multi-party communication protocols. Nonetheless, scalable access control for arbitrary matching operators remains an open problem. We have used an epoch based subscription model that does not permit revocations within one time epoch. However, this model is very realistic in several payment based pub-sub services that charge some subscription fee per epoch.

8. CONCLUSION

We have presented *EventGuard*, a dependable system architecture for protecting pub-sub services from various attacks. EventGuard offers security features that are critical to pub-sub overlay services, such as authenticity, confidentiality, integrity, and resilience to flooding based DoS attacks. We have described the two key components of EventGuard. The first component is a suite of security guards that secure the basic publish and subscribe operations from DoS attacks and unauthorized reads and writes. These guards can be plugged-into a wide-area content-based pub-sub system in a seamless manner. The second component is a resilient pub-sub network design that is capable of providing secure and yet scalable message routing, countering message dropping-based DoS attacks. A unique feature of EventGuard is its unified security framework that meets both security goals for safeguarding the pub-sub overlay services from various vulnerabilities and threats and performance goals for maintaining the simplicity and scalability of the overall system while providing security guarantees. We have reported a series of experimental evaluations, showing that EventGuard can secure a pub-sub overlay service with minimal performance penalty. Our prototype implementation on top of Siena also demonstrates that EventGuard is easily stackable on any content-based pub-sub core.

Acknowledgements

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

Aguilera, K. and Strom, R. 2000. Efficient atomic broadcast using deterministic merge. In *Proceedings of the 19th ACM PODC*.

Aguilera, M., Strom, R., Sturman, D., Astley, M., and Chandra, T. 1999. Matching events in a content-based subscription system. In *Proceedings of the 18th ACM PODC*.

Atallah, M., Frikken, K., and Blanton, M. 2005. Dynamic and efficient key management for access hierarchies. In *Proceedings of ACM CCS*.

Atallah, M. J., Blanton, M., and Frikken, K. B. 2007a. Efficient techniques for realizing geo-spatial access control. In *Asia CCS*.

Atallah, M. J., Blanton, M., and Frikken, K. B. 2007b. Incorporating temporal capabilities in existing key management schemes. In *ESORICS*.

Banavar, G., Chandra, T., Mukherjee, B., and Nagarajarao, J. 1999. An efficient multicast protocol for content-based publish subscribe systems. In *Proceedings of the 19th ICDCS*.

Carzaniga, A., Rosenblum, D. S., and Wolf, A. L. 2001. Design and evaluation of a wide-area event notification service. In *ACM Transactions on Computer System, 19(3):332-383*.

Datta, A. K., Gradinariu, M., Raynal, M., and Simon, G. 2003. Anonymous publish/subscribe in P2P networks. In *Proceedings of IPDPS*.

Eastlake, D. and Jones, P. 2001. US secure hash algorithm 1. http://www.ietf.org/rfc/rfc3174.txt.

ElGamal, T. 1985. A public key cryptosystem and a signature scheme based on discrete logarithm. In *IEEE transactions on information theory, 31(4): 469-472*.

FIPS. Data encryption standard (DES). http://www.itl.nist.gov/fipspubs/fip46-2.htm.

Krawczyk, H., Bellare, M., and Canetti, R. HMAC: Keyed-hashing for message authentication. http://www.faqs.org/rfcs/rfc2104.html.

Malkhi, D., Rodeh, O., and Reiter, M. 2001. Efficient update diffusion in byzantine environments. In *Proceedings of 20th IEEE SRDS*.

Mathpages. Generating monotone boolean functions. http://www.mathpages.com/home/kmath094.htm.

OpenSSL. Openssl. http://www.openssl.org/.

Opyrchal, L. and Prakash, A. 2001. Secure distribution of events in content-based publish subscribe system. In *Proceedings of the 10th USENIX Security Symposium*.

Perng, G., Reiter, M. K., and Wang, C. 2006. M2: Multicasting mixes for efficient and anonymous communication. In *Proceedings of IEEE ICDCS*.

Qin Lv, S. R. and Shenker, S. 2002. Can heterogeneity make gnutella scalable? In *Proceedings of the first International Workshop on Peer-to-Peer Systems*.

Rafaeli, S. and Hutchison, D. 2003. A survey of key management for secure group communication. In *Journal of the ACM Computing Surveys, Vol 35, Issue 3*.

Raiciu, C. and Rosenblum, D. S. 2006. Enabling confidentiality in content-based publish/subscribe infrastructures. In *Proceedings of IEEE SecureComm*.

Rivest, R. 1992. The MD5 message-digest algorithm. http://www.ietf.org/rfc/rfc1321.txt.

Song, D., Wagner, D., and Perrig, A. 2000. Practical techniques for searches over encrypted data. In *IEEE S & P Symposium*.

Srivatsa, M., Gedik, B., and Liu, L. 2006. Scaling unstructured peer-to-peer networks with multi-tier capability aware topologies. In *Proceedings of IEEE Transactions on Parallel and Distributed Systems (TPDS), Vol. 17, No. 10*.

Srivatsa, M. and Liu, L. 2004. Vulnerabilities and security issues in structured overlay networks: A quantitative analysis. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.

Srivatsa, M. and Liu, L. 2005. Secure event notification architecture for publish-subscribe networks. In *ACM CCS*.

Srivatsa, M. and Liu, L. 2007. Secure event dissemination in content-based publish-subscribe networks. In *IEEE ICDCS*.

Srivatsa, M., Xiong, L., and Liu, L. 2005. Trustguard: Countering vulnerabilities in reputation management for decentralized overlay networks. In *Proceedings of the World Wide Web Conference (WWW)*.

Wang, C., Carzaniga, A., Evans, D., and Wolf, A. L. 2002. Security issues and requirements for internet-scale publish-subscribe systems. In *Proceedings of the 35th Hawaii International Conference on System Sciences*.

Wong, C. K., Gouda, M. G., and Lam, S. S. 2000. Secure group communications using key graphs. In *IEEE/ACM Transactions on Networking: 8, 1(Feb), 16-30*.

Xiong, L. and Liu, L. 2004. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. In *Proceedings of IEEE TKDE, Vol. 16, No. 7*.

Yang, Y. R., Li, X. S., Zhang, X. B., and Lam, S. S. 2001. Reliable group rekeying: A performance analysis. In *Proceedings of ACM SIGCOMM*.

Zegura, E. W., Calvert, K., and Bhattacharjee, S. 1996. How to model an internetwork. In *Proceedings of IEEE Infocom*.