Securing Publish-Subscribe Overlay Services with EventGuard

Abstract.

A publish-subscribe overlay service is a wide-area communication infrastructure that enables information dissemination across geographically scattered and potentially unlimited number of publishers and subscribers. A wide-area publish-subscribe (pub-sub) system is often implemented as a collection of spatially disparate nodes communicating on top of a peer to peer overlay network. Such a model presents many inherent benefits such as scalability and performance, as well as potential challenges such as: (i) confidentiality & integrity, (ii) authentication, and (iii) denial-of-service (DoS) attacks. In this paper we present EventGuard for securing pub-sub overlay services. EventGuard comprises of two components. The first component is a suite of security guards that can be seamlessly plugged-into a content-based pub-sub system. The second component is a resilient pub-sub network design that is capable of scalable routing, handling message dropping-based DoS attacks and node failures. EventGuard mechanisms aim at providing security guarantees while maintaining the system's overall simplicity, scalability and performance metrics. We present an implementation which shows that EventGuard is easily stackable on any content-based pub-sub core. Finally, our experimental results show that EventGuard can secure a pub-sub system with minimal performance penalty.

1 Introduction

Emerging number 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 is a wide-area communication infrastructure that enables information dissemination across geographically scattered and potentially unlimited number of publishers and subscribers [5]. A wide-area publish-subscribe (pub-sub) system is often implemented as a collection of spatially disparate nodes communicating on top of a peer to peer overlay network [5]. In such an environment, publishers publish information in the form of events 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 against all active subscriptions, and notifies the subscribers of an event if and only if the event matches their registered interest.

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, offloading the information dissemination task to the pub-sub network not only improves the scalability and the effectiveness of the pub-sub system, but also permits dynamic and fine-grained subscriptions.

On the other hand, many security concerns exist in such an environment regarding authenticity, confidentiality, integrity and availability. For example, how can we guarantee that only the genuine publications are delivered to the subscribers (publication authenticity) and only the subscribers who subscribe (e.g., have paid) to the service will receive publications matching their interest (subscription authentication)? How do we prevent unauthorized modifications of pub-sub messages (publication and subscription integrity)? How do we perform content-based routing without the publishers trusting the pub-sub network (publication confidentiality)? Can subscribers receive publications without revealing their subscriptions to the pub-sub network (subscription confidentiality)? And how do we defend the pub-sub services from publication spamming and flooding attacks, selective and random message dropping attacks, and other Denial of Service (DoS) attacks?

Most of the existing research on pub-sub systems have been largely dedicated to the performance and scalability of pub-sub networks, as well as, the expressiveness of subscription models [5, 3, 7]. Only recently, a few researchers have studied specific security requirements of pub-sub networks [23], pointing out attacks threatening message integrity (unauthorized writes) and authenticity (fake origins) in addition to message confidentiality (unauthorized reads). Unfortunately, most of the existing secure event distribution protocols focus only on content confidentiality [17, 14]. Very few have devoted to developing a coherent security framework that can guard the pub-sub system from multiple security problems inherent in them.

In this paper, we present *EventGuard* – a dependable framework and a set of defense mechanisms for securing a pub-sub overlay service. EventGuard comprises of two components. The first component is a suite of security guards that can be seamlessly plugged-into a wide-area content-based pub-sub system. The second component

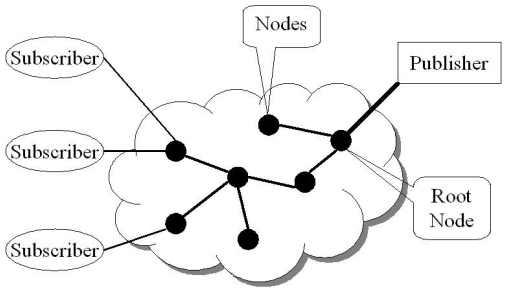

Figure 1: Basic Pub-Sub System

is a resilient pub-sub network design that is capable of secure and yet scalable message routing, countering message dropping-based DoS attacks and node failures. We also present a prototype implementation of EventGuard on top of Siena [5] to show that EventGuard is easily stackable on any content-based pub-sub core. Our experimental results show that EventGuard can secure a pub-sub overlay service with minimal performance penalty.

2 Preliminaries

2.1 Reference Pub-Sub Model

This section presents our reference pub-sub model that is very similar to that used in a content-based pub-sub system like Siena [5]. A pub-sub system implements five important primitives: *subscribe*, *advertise*, *publish*, *unsubscribe* and *unadvertise*. Subscribers specify the events in which they are interested using *subscribe*. Publishers advertise the type of events they would publish using *advertise*. Publishers publish events via *publish*. A subscription remains in effect until it is canceled by a call to *unsubscribe*. An advertisement remains in effect until it is canceled by an *unadvertise*.

Consider the stock quote dissemination where an example event consists of the following *attributes*: e = $\langle\langle$ `exchange`, NYSE$\rangle$, $\langle$`symbol`, IBM$\rangle$, $\langle$`price`, 122$\rangle$, $\langle$`volume`, 2500$\rangle\rangle$. An example subscription could consist of the following *constraints*: f = $\langle\langle$`symbol`, EQ, IBM$\rangle$, $\langle$`exchange`, EQ, NYSE$\rangle$, $\langle$`price`, GT, 100$\rangle\rangle$, where EQ denotes the equality operator and GT denotes the greater than operator.

As illustrated in Figure 1, in a wide-area pub-sub system, publishers and subscribers are usually outside the pub-sub network. Typically, we have a relatively small set of known and trusted publishers and a much larger set of subscribers. A natural choice for the topology of a pub-sub network is a hierarchical topology (see Figure 1). When a node n receives a subscription request $subscribe(m, f)$ from node m, it registers filter f with the identity of node 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.

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, advertisements and unadvertisements flow from the root (publisher) to the leaves (subscribers) of the tree. Similarly, subscriptions and unsubscriptions are propagated from the leaves to the root of the tree. Note that a node n in the pub-sub network may belong to one or more dissemination trees. When a node n receives a publication $publish(e)$ for an event e, it uses the pub-sub dissemination tree to identify all active subscriptions whose filters $\{f_1, f_2, \cdots, f_p\}$ are matched by the event e. Then, node n forwards event e only to those children nodes $\{x_1, x_2, \cdots, x_q\}$ that have subscribed for some filter f_i ($1 \leq i \leq p$).

2.2 Threat Model

The pub-sub overlay service model comprises of three entities: publishers, subscribers and routing nodes. In this section, we present our threat model and the trust assumptions that EventGuard makes on all these entities.

Publishers. EventGuard assumes that authorized publishers are honest and publish only valid events. One could build a feedback mechanism wherein the subscribers rank the publishers periodically [22, 24]. 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 an authorized publishers and *spam* or *flood* the network and consequently the subscribers, with incorrect or duplicate publications, advertisements or unadvertisements.

Subscribers. EventGuard assumes that authorized subscribers are *semi-honest*. 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 the nodes on the pub-sub network may be untrusted. However, we also

assume that a significant fraction of the pub-sub nodes are non-malicious so as to ensure that the pub-sub network is *alive*. A pub-sub network is alive if it can route messages and maintaining 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 or randomly drop pub-sub messages. Further, malicious nodes may attempt to *spam* or *flood* other nodes and subscribers.

Finally, EventGuard assumes that the underlying IP-network may not guarantee confidentiality, integrity or authenticity. However, we assume that the underlying domain name service (DNS), the network routers, and the related networking infrastructure is secure, and hence cannot be subverted by an adversary.

Pub-sub systems typically support two levels of event matching – *topic-based* and *content-based*. In a topic-based matching scheme [1], every event is marked with a topic and all filters use only the equality operator (EQ). Content-based matching schemes [5, 2, 3] are layered on top of topic-based matching scheme and they allow more sophisticated event matching and filtering, e.g., $\langle$`stock price`, GT, 100$\rangle$. Due to space constraints, in this paper we describe EventGuard mechanisms in the context of a topic-based pub-sub system. Interested readers may refer to [21] for a description of EventGuard techniques for handling complex event filtering conditions on numeric attributes and subject (concept) hierarchies.

3 EventGuard Overview

3.1 Design Goals

EventGuard has fundamentally two sets of design goals: security goals and performance goals.

Authentication. In a pub-sub system a publication (and advertisement, unadvertisement) is sent from a publisher (sender) to a subscriber (receiver) through the pub-sub network (channel). It is important to make sure that all publications (and advertisements, unadvertisements) are authentic in order to avoid spoofed publications. On the other hand, subscriber authenticity is important when the application requires that subscribers should receive only the publications to which they are authorized (paid) to access. In addition, sender authentication within the pub-sub network is critical when pub-sub nodes are compromised. A compromised node can insert bogus messages, and route messages to arbitrary destinations.

Confidentiality and Integrity. Confidentiality and integrity of a message sent by a publisher (sender) to a subscriber (receiver) is defined with respect to the pub-sub nodes (channel). We require that the pub-sub network nodes or any observer of the pub-sub network should neither be able to gain knowledge about the messages routed through them nor corrupt them in an undetectable manner. Concretely, we need to guarantee three types of confidentiality and integrity.

First, we need publication confidentiality to ensure only authorized subscribers can read an event. We also need publication integrity to protect publications from unauthorized modifications.

Second, subscribers may wish to keep their subscriptions private. Concretely, the subscriber would like the pub-sub network to evaluate subscription filter $f(pbl)$ with respect to the publication pbl without revealing f to the pub-sub nodes. Further, we need subscription integrity to safeguard subscriptions from unauthorized modification when routing them through the pub-sub overlay.

Third, we need the pub-sub network to perform content-based routing without requiring the publishers and the subscribers to trust the network with the content. Content confidentiality is especially important when content being published contains sensitive information, which the publishers and the subscribers may wish to keep a secret from the pub-sub nodes. Content integrity prevents messages in transit from unauthorized modification by pub-sub nodes.

Availability. EventGuard refers availability to the resilience of the pub-sub system against Denial of Service (DoS) attacks. There are three major types of DoS attacks possible on pub-sub systems: (i) *flooding* based attacks attempt to flood the pub-sub system with large amount of bogus and duplicate messages, (ii) *fake* unsubscribe (and unadvertise) attack attempts to send spurious unsubscribe (and unadvertise) requests; for example, if a node x' ($\neq x$) sends $unsubscribe(x, f)$ to node x's parent then it would deny x of all events e that is covered by filter f, and (iii) *selective or random dropping* attack attempts to drop messages either selectively (say, based on the publication's topic) or randomly.

In addition to the security goals, EventGuard has two important performance related goals.

Performance and Scalability. We require the EventGuard mechanisms to *scale* with the number of nodes in the network. In addition, EventGuard should add minimal performance overhead to a pub-sub system.

Ease of Use and Simplicity. We require that EventGuard

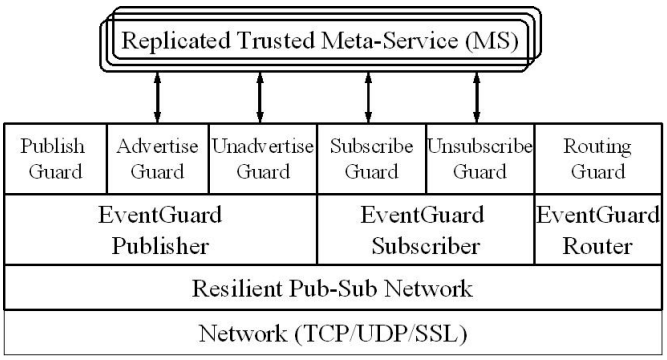

Figure 2: EventGuard Architecture

mechanisms be simple and easy to deploy, operate and administer.

3.2 System Architecture

EventGuard is designed to be completely modular and operates entirely above a content-based pub-sub core. Figure 2 shows EventGuard's architecture. EventGuard comprises 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 resilient pub-sub network that is capable of handling node failures and selective & random dropping-based DoS attacks. The third component is a light-weight trusted meta-service (MS) to provide subscription and advertisement services to a pub-sub system.

Security Guards. EventGuard takes a unified approach to secure a pub-sub network. Each security guard secures one pub-sub operation against all potential attacks. Concretely, 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 three important building blocks: token, key and signature. We use tokens as pseudo-names for topics to mitigate selective dropping attacks at the pub-sub network level. We protect the confidentiality and integrity of publications from pub-sub nodes and from unauthorized subscribers using cryptographic keys. We protect the pub-sub service from spam, flooding based DoS attacks and spoofed messages using signatures. We describe the three basic building blocks in Section 4.1 and discuss how to design security guards using these building blocks in the rest of Section 4.

Resilient pub-sub network. EventGuard achieves resilience to message dropping based attacks by constructing a network topology that is richer than the popular tree-based event dissemination topology [19]. We improve the resilience of the pub-sub network by modifying the tree structure to incorporate multiple independent paths [20] from a publisher to its subscribers.

Trusted Meta-Service. EventGuard relies on a thin *trusted* meta-service (MS) to create tokens and keys for controlling confidentiality of topics and publications. It also creates signatures for authenticating subscribers and publishers and guarding the pub-sub service from flooding-based DoS attacks. Four of the five types of pub-sub messages, subscribe, unsubscribe, advertise, unadvertised, require MS to generate tokens, keys and signatures. However, the most common operations, publish and routing, do not require the direct support from MS. We design the MS with the following three objectives in mind. First, we aim at minimizing the amount of work assigned to the MS. Keeping the MS simple enables one to ensure that the MS is relatively bug-free and is thus well-protected from malicious nodes. Second, we would like to limit the number of secrets maintained by the MS to at most one small key. Having to maintain only a few small keys secret enables the MS's administrator to afford physical security for those keys in the form of, say a smart card. Third, it should be possible to easily replicate MS; the MS replicas should be able to function independently without having to interact with one another. This enables meta-services to be created and destroyed *on demand* to handle varying load.

The other potential benefits of supporting a light-weight MS in the pub-sub systems are accounting and audit capabilities. For instance, the pub-sub system may want to impose a cost model on the pub-sub system to ensure that subscribers pay the system for their subscriptions and publishers pay the system for their advertisements. Accounting and pricing can be a valuable means to reduce spam in wide-area distributed systems, for e.g., many authors have proposed to condone email spam by associating a cost with every email [6]. Furthermore, one can also provide auditing capability at the MS to resolve any accounting or pricing related issues regarding subscribers and publishers.

4 EventGuard: Security Guards

In this section 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.

4.1 Tokens, Keys and Signatures

The first building block is the concept of per-topic token. In EventGuard, publishers can publish and advertise events in terms of topics. We create a token for each topic. Tokens are essential for protecting messages (e.g., subscriptions) from selective dropping DoS attack. Concretely, by introducing tokens, nodes in the pub-sub network are not aware of the topic names; instead they match and route events on the pub-sub network based on tokens until the events ultimately reach the appropriate subscribers.

Token is a pseudonym for a topic name. There is a one-to-one mapping between a topic name w and it's token $T(w)$. However, given a token $T(w)$ it is computationally infeasible to guess the topic name w. A subscriber subscribes for a topic w by subscribing for its token with filter $f(w) = \langle \texttt{topic}, EQ, T(w) \rangle$, where `topic` is the attribute name for topic, EQ denotes the equality operator, and $T(w)$ denotes the token corresponding to the topic w.

The second building block is the concept of per-topic key. In an EventGuard powered pub-sub system, keys are fundamental for achieving confidentiality and integrity. By encrypting message content with a secret encryption key, we can prevent contents of publications from unauthorized reads and writes.

Every topic w in the pub-sub system has an associated key $K(w)$. The MS is responsible for providing $K(w)$ to a subscriber when the subscriber subscribes for topic w and to publishers when they advertise for topic w. The encryption key $K(w)$ enables the publisher to encrypt events that belong to topic w. Now only a legal subscriber to topic w would be able to decrypt the message. The publication of content pbl under a topic w would be constructed as $e = \langle \alpha_1, \alpha_2 \rangle$, where $\alpha_1 = \langle \texttt{topic}, T(w) \rangle$ and $\alpha_2 = \langle \texttt{content}, E_{K(w)}(pbl) \rangle$, where `content` denotes the attribute name for the published message. Note that $E_K(pbl)$ denotes the encryption of pbl with encryption key K and some symmetric key encryption algorithm E (e.g., DES [9] or AES [13]). The nodes in the pub-sub system are not aware of keys; they would still be able to route an event based on its token.

The third building block in EventGuard is the concept of signature. 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 [8] 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 $pk(x)$. Subscriptions, unsubscriptions, advertisements and unadvertisements are signed by the MS, while publications are signed by its publisher. This ensures that malicious nodes cannot flood the pub-sub network with bogus publications or phony subscriptions.

There are at least two alternative approaches to signatures. One apparent alternative is to use keyed message authentication codes (MACs). Shared MAC keys between a publisher and a subscriber allow the subscriber to authenticate all publications it receives. However, there is a dilemma with this approach. On one hand, we cannot afford to give away MAC keys to pub-sub network nodes since a malicious node may use this key to flood messages on the pub-sub network. On the other hand, without these MAC keys, nodes on the pub-sub network would neither verify the authenticity of messages nor control flooding based DoS attacks.

The second alternative to signatures is to use a Byzantine fault-tolerant (BFT) information dissemination protocol [11]. Let m denote an upper bound on the number of malicious nodes in the pub-sub network. The publisher initiates a publication message M by sending it to $2m+1$ seed nodes. Any non-seed node u would consider the message M authentic if and only if it received $m+1$ identical copies of the message M from $m+1$ distinct nodes in the system. Note that if $m+1$ copies of a message are identical then at least one of the copy is guaranteed to have originated from a non-malicious node. Node u continues propagating the message M (usually by broadcast) until all subscribers receive the message M. An obvious advantage of BFT techniques is that it does not pay the overhead of using a PKI based signature. On the flip side however, BFT techniques incur much higher communication cost. This makes BFT techniques suitable only for environments that inherently support broadcast commu-

nication (e.g., local area networks). For wide-area Internet applications like pub-sub systems it is important to keep the communication cost very low.

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.

4.2 Subscribe Guard

Subscribe guard is designed for achieving subscription authentication, subscription confidentiality & integrity, and preventing DoS attacks based on spurious subscriptions. When a subscriber S wishes to subscribe for a topic w, it sends a request to a MS. At this point, the MS may act as the authority for implementing a cost model for the pub-sub system and 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, $sb(w)$ denote the subscription permit issued by MS for the subscription $\phi'(w)$, and $\phi(w)$ denote the transformed subscription message used by the subscriber S in EventGuard.

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

A constraint $\phi = \langle name_\phi, ANY, value_\phi \rangle$ covers an attribute $\alpha = \langle name_\alpha, value_\alpha \rangle$ if $name_\phi$ equals $name_\alpha$. The values $value_\phi$ and $value_\alpha$ are ignored by Siena, but are used by EventGuard to embed signatures in messages.

The MS 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-pseudo random function KH (approximated by HMAC-MD5 [10]) and a secret key K. 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 pseudo-random function H (approximated by MD5 [12] or SHA1 [18]). $UST^S(w)$ is an unsubscribe 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 $sig^S_{MS}(T(w)) = \langle r, s \rangle$ is an ElGamal signature by the MS on the token $T(w)$ in the subscription permit $sb(w)$ provided to subscriber S. Since the r-component of the signature is always unique, we use the r-component of this signature as the subscription identifier ($sbId$). 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 duplicate subscription based flooding attack. 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 unsubscribe 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 an EventGuard subscription filter $\phi(w)$. The subscriber S could then submit $\phi(w)$ to the pub-sub network. Consequently, any publication that includes the token $T(w)$ is routed to S. Note that pub-sub nodes cannot perform unauthorized reads or writes on a subscription message, thus guaranteeing subscription confidentiality and integrity.

4.3 Advertise Guard

Advertise guard is designed for achieving advertisement authentication, advertisement confidentiality & integrity, and preventing DoS attacks based on bogus advertisement. When a publisher P wishes to publish events under topic w, it sends a request to the MS. At this point the MS may charge a publication fee to the publisher that is some arbitrary function of the topic w. Let $\phi'(w)$ be the original advertisement filter for topic w, $ad(w)$ denote the advertisement permit given by the MS to the publisher P and $\phi(w)$ denote the transformed advertisement.

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

The key $K(w)$, and the token $T(w)$ are computed in the same manner as that for subscriptions. The special token $UAT^P(w)$ is used to prevent fake unadvertisement

based DoS attack (discussed in unadvertise). The publisher then constructs the advertisement filter $\phi(w)$ and propagates it to the pub-sub network. Note that the signature $sig^P_{MS}(T(w) \parallel P \parallel pk(P))$ ties the publisher's name (P) to its public-key ($pk(P)$). The public-key $pk(P)$ is essential for the pub-sub nodes and the subscribers to verify the authenticity of publications.

4.4 Publish Guard

Publish guard is designed to safeguard the publication from publication confidentiality & integrity, publication authenticity, and DoS attacks based on bogus publications. Suppose a publisher P wishes to publish a publication pbl under topics w_1, w_2, $\cdots$, w_m. The content pbl could be any arbitrary sequence of bytes including text, multimedia, and so on. For each topic w_i, the publisher uses the topic's token $T(w_i)$ and its encryption key $K(w_i)$ provided by the MS during advertisement (see advertise guard). A publication event e is constructed as follows. Let e' denote the original publication message, e denote a legal event publication transformed from e' using tokens and content encryption of publication messages.

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

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, $sig_P(e)$. 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.

Note that a publisher uses an ElGamal signature to sign its publications. The first component of the signature is used as the publication identifier ($pbId$). The signature serves two purposes. First, it enables nodes in the pub-sub network to check the validity of a publication. Second, we use the publication identifier (the r-component of the signature) to detect and condone a DoS attack based on publication flooding (discussed later under routing guard).

4.5 Unsubscribe Guard

Unsubscribe guard is designed to prevent unauthorized unsubscribe messages, flooding of unsubscribe messages. 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. Note that S received the signature $sig^S_{MS}(T(w))$ and the unsubscribe token $UST^S(w)$ when it subscribed for topic w. 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 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, $usb(w)$ to denote an unsubscription permit given by the MS and $\phi(w)$ to denote the transformed unsubscription request.

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

Note that the signature $sig_{MS}(T(w) \parallel sbId)$ 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 request, delete the subscription corresponding to $sbId$ and use the unsubscription identifier $usbId$ (the r component of signature $sig_{MS}(T(w) \parallel sbId)$) to detect unsubscription flooding based DoS attacks.

4.6 Unadvertise Guard

Unadvertise guard is designed to prevent unauthorized unadvertise messages, and flooding of unadvertise messages. 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 unadvertise token $UAT^P(w)$ is used to prevent fake unadvertise based DoS attack. Let $\phi'(w)$ denote the original unadvertisement message for topic w,

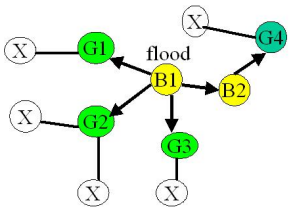

Figure 3: Handling Flooding based DoS attacks in EventGuard

$uad(w)$ denote the unadvertisement permit given by MS and $\phi(w)$ denote the transformed advertisement request.

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

The publisher P uses the signature $sig_{MS}(T(w) \parallel P \parallel adId)$ included in the unadvertisement permit $uad(w)$ to create an unadvertise request and submit it to the pub-sub overlay network. Nodes in the network use the MS's signature to check the validity of an unadvertisement, delete the advertisement corresponding to $adId$ and use the unsubscription identifier $uadId$ (the r component of signature $sig_{MS}(T(w) \parallel P \parallel adId)$) to detect unadvertisement flooding based DoS attacks.

4.7 Routing Guard

The pub-sub network nodes route messages based on tokens. Besides performing the functionality of a regular pub-sub node, we require the nodes to perform additional checks to ensure safety from DoS attacks.

EventGuard requires nodes on the pub-sub network to check for authentic signatures and detect duplicate messages. With the guarantee of sender authenticity and the prevention of duplicate messages, no flooding attack could propagate beyond one non-malicious node. Figure 3 illustrates this point. In Figure 3, a malicious node $B1$ attempts a flooding based DoS attack on0 all its neighbor nodes. Observe that no invalid message (incorrect signatures) and no duplicate message from node $B1$ would propagate beyond the non-malicious 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 in three steps. First, we require a node to maintain the identifiers seen in the last max_delay time units. 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 or the signature check fails.

4.8 Rekeying

Per-topic encryption keys (authorization keys) are like capabilities issued to authorized subscribers. Hence, when a subscriber unsubscribes, the per-topic encryption key needs to be changed and the new key has to be communicated to all other authorized subscribers. We observe that changing the per topic encryption key on every unsubscription can be very expensive. As a result, EventGuard resorts to periodic rekeying.

We periodically change all per-topic encryption keys by changing $rk(MS)$. More specifically, we divide time into epochs of $epoch$ time units (say, one month). All subscriptions and advertisements need to be renewed at the beginning of every time epoch. We number epochs with consecutive integers starting from epoch number 0. The secret key used by the MS in the T^{th} epoch is derived from the primary secret key $rk(MS)$ as $rk(MS, T)$ = $KH_{rk(MS)}(T)$. The MS uses $rk(MS, T)$ to replace $rk(MS)$ during the T^{th} epoch for generating authorization keys. Hence, if a subscriber S unsubscribes for topic w in epoch T, it would still be able to read the contents of publications under topic w till the end of epoch T (but not after epoch T) by sniffing the pub-sub network.

Note that $rk(MS, T)$ is used only for generating per-topic encryption keys. The MS always uses the primary secret key $rk(MS)$ for signing subscriptions, unsubscriptions, advertisements and unadvertisements. Also, all topic tokens, unsubscribe tokens and unadvertise tokens are derived using the primary secret key $rk(MS)$. Hence, tokens do not change across epochs. Therefore, none of the subscription and advertisements disseminated into the pub-sub network needs to be changed every epoch. Our rekeying technique requires only the subscribers and the publishers to obtain new per-topic encryption keys from the MS every epoch. Additionally, periodic rekeying facilitates the MS to bill the subscribers and the publishers for the next epoch.

5 $r-$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 attack. In this section, we present techniques to restructure the pub-sub network in way that can effectively handle random message dropping based DoS attacks. We first define a r-resilient pub-sub network as follows:

Definition *r-resilient pub-sub network*: A pub-sub network is said to be r-resilient ($0 < r < 1$) if $r * 100\%$ of the messages are resilient to dropping attack.

There are two important design goals in constructing a r-resilient pub-sub network: (i) the pub-sub network must be resilient to message dropping attacks, and (ii) the communication cost should be minimal. We first discuss two network topologies that represent two extremities of the spectrum and then describe the EventGuard solution. The first network topology is a $a-$ary tree topology. The second network topology mirrors the propagation scheme used in Byzantine fault tolerant information dissemination [11]. The $a-$ary tree topology incurs minimum communication cost but is not resilient to message dropping attacks. Our analytical estimates show that the communication cost for the BFT dissemination algorithm is at least 20 times that of a $a-$ary tree dissemination algorithm. Refer to Appendix A for details. 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 in the order of 70ms [25], it might be advisable to limit the communication cost while tolerating a small amount of dropped messages.

In this section, we present the $r-$resilient network guard in two steps. First, we study the resilience of $a-$ary trees towards message dropping attacks. Second, we propose a network guard that strikes a trade-off between resilience to message dropping attacks and the communication cost.

5.1 Resilience to Message Dropping Attacks

A simple $a-$ary tree-based network is highly 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 denotes the fraction of

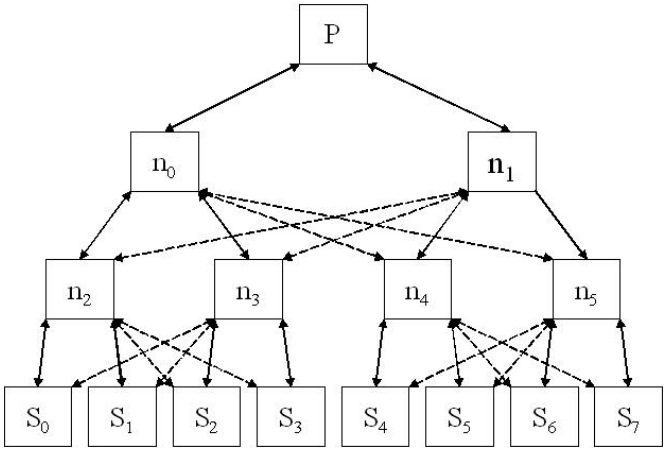

Figure 4: Constructing Resilient Networks: Thick lines represent links in the binary tree network and the dashed lines represent additional links added to binary tree network to make its $ind = 2$

nodes that are malicious and assume that malicious nodes are randomly distributed on the network. The probability that a publication reaches a subscriber is $Pr(succ) = (1-p)^h$, where $h = \lceil \log_a NS$ denotes the height of the tree and NS denotes the number of subscribers. Even when $p = 10\%$, with $h = 5$ we find that the probability that a publication is successful delivered 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 a increases the load on the publisher and the nodes on the pub-sub network increases, thereby harming the scalability of the system (recall that each node has a children nodes).

The key problem with the tree-based topology is that there is only one *independent path* from a publisher to a subscriber [20]. 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 [20].

5.2 Low Cost Resilient Networks

In this section, we modify a $a-$ary tree such that it has ind independent paths while increasing the communica-

tion cost by not more than a factor of ind ($ind \leq a$). For simplicity, we illustrate our technique by modifying a binary tree network ($a = 2$) to yield a network with $ind = 2$.

Figure 4 shows the key idea behind constructing a resilient event dissemination network G^2. 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.*

For a detailed proof of Theorem 5.2 refer to Appendix B. One can easily extend this network construction scheme for any $ind \leq a$. We can construct a resilient nework G^{ind} by connecting every node n to $parent(n)$ and $ind-1$ distinct siblings of $parent(n)$ (these siblings indeed exist since $ind \leq a$). The proofs for Claims 5.3 and 5.4 follow from Claim 5.1 and Theorem 5.2. Due to space constraint, we have omitted them from this paper.

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

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

As we increase ind, the communication cost increases by a factor ind since each message is multicast along all ind independent paths. However, we believe that $ind = 2$ would suffice for most practical pub-sub networks. Let p denote the fraction of nodes that are malicious. 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 NS = 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. This increase in load consequently affects the system's scalability. Ideally, we would like to keep a as a small constant while providing acceptable resilience to message dropping attacks on the pub-sub network. In general, our technique can be employed to construct a r-resilient network with $Pr(succ) = r$ by carefully choosing ind and a.

6 EventGuard Evaluation

We have implemented EventGuard on top of the Siena pub-sub core [5]. We used a Java based implementation of Siena [4] and added EventGuard extensions to it in the form of an *EventGuard package*. No changes were made as such to the Siena pub-sub core (e.g., the content-based routing and event matching algorithms). For further details on EventGuard implementation refer to [21].

Cryptographic Primitives. Table 1 provides a quick reference of the cryptographic algorithms used by our implementation of EventGuard. We use MD5 for the hash function H, HMAC-MD5 for the keyed hash function KH, 128bit AES encryption algorithm in cipher-block-chaining mode (CBC) for E and 512bit ElGamal algorithm for digital signatures.

We evaluate our EventGuard prototype in two steps. We first present some micro-benchmarks to quantify the overhead of EventGuard mechanisms. Then we present macro-benchmarks to quantify the performance of the entire system. We measure changes in throughput, latency of the pub-sub network as an effect of EventGuard mechanisms. We also quantify the effect of EventGuard's resilience to DoS attacks.

6.1 Micro-Benchmarks

In this section, we analytically 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 1 shows the amount of time it takes for executing different cryptographic primitives used by EventGuard. These times have been measured

H	MD5	2 MB/s-61 MB/s
KH	HMAC-MD5	1.5 MB/s-49 MB/s
E	AES-128-CBC	10 MB/s
sig	ElGamal-512-sign	714 Sign/s
sig	ElGamal-512-verify	588 Verify/s

Table 1: Computation Times for Cryptographic Primitives used by EventGuard

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 (MD5 and HMAC-MD5) depends on the block size. For instance, MD5 hashes can be computed at 2 MB/s when the block size is small (16 Bytes) and about 61 MB/s when the block size is large (1024 Bytes).

We performed an analytical estimate on the computational time for all pub-sub operations: subscriptions, advertisements, publications, unsubscriptions and unadvertisements. We analyzed the cost of these operations at all four entities: a publisher, a subscriber, a pub-sub node, and the MS. We also analyzed the messaging and storage cost at these four entities. Tables 2, 3 and 4 summarizes the results obtained in this section. For a detailed discussion on EventGuard micro-benchmarks refer to [21].

6.2 Macro-Benchmarks

In this section, we present two sets of macro-benchmarks for EventGuard. The first set of experiments is simulation based. The second set of experiments is obtained from our prototype implementation of EventGuard on Siena pub-sub core.

6.2.1 Simulation based Experiments

In this section, we present performance results from simulation based experiments on EventGuard. First, we present the improvements on message confidentiality & integrity due to EventGuard. Second, we measure the throughput of the system in the presence of malicious nodes. Third, we show the average load on the MS, the publisher, the subscriber and the nodes as we vary the subscription and publication rate. Fourth, we demonstrate the resilience of the pub-sub network architecture used in EventGuard against message dropping-based DoS attacks.

Simulation Setup. We used GT-ITM [25] topology generator to generate an Internet topology consisting of 4K nodes. We linked these nodes 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 subscribers were randomly connected to one leaf node in the pub-sub network. We used discrete event simulation [9] to simulate 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 [15]. Each publisher publishes on 16 topics (randomly picked from the set of 128 topics) and each subscriber subscribes for 4 topics.

Confidentiality and Integrity. Figure 5 shows the fraction of messages that violate their confidentiality and integrity guarantees 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. Also, 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 6 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 considered two cases: Case one wherein the malicious nodes are uniformly distributed throughout the pub-sub network (EventGuard-sparse in Figure 6); and Case two wherein malicious nodes form k clusters in the pub-sub network (EventGuard-cluster-k in Figure 6). When malicious nodes are clustered together on the pub-sub network, we found that the loss in throughput for EventGuard is relatively much smaller. This is because EventGuard ensures that no flooding attack propagates beyond one non-malicious pub-sub node. Hence, clustered malicious nodes cannot significantly affect other non-malicious nodes in the system (see Figure 3).

Load. Figure 7 shows the relative computational load on the MS, the publisher, the subscribers and a pub-sub nodes as we vary the rate of subscriptions, unsubscrip-

	MS (ms)	publisher (ms)	subscriber (ms)	node (ms)
subscribe	$1.44 + 0.00117 * \|w\|$	-	1.7	1.7
unsubscribe	3.14	-	1.7	1.7
publish	-	$1.4 + (\|pbl\| + 16m) * 0.0001$	$1.7 + (\|pbl\| + 16) * 0.0001$	1.7
advertise	$1.4 + 0.00117 * \|w\|$	1.7	-	1.7
unadvertise	3.14	1.7	-	1.7

Table 2: 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 3: 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

tions and publications keeping the aggregate rate a constant (we do not consider advertisement and unadvertisement costs in this experiment). The computation load for basic operations were obtained from Table 2. We set the subscription rate to be equal to 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 permits one to add additional meta-services. The fact that the meta-services do not have to interact with one another (they share the private key $rk(MS)$) makes it possible for one to build an efficient *load balancing* system to handle the MS load and vary the number of active meta-services *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.

Selective and Random Dropping Attack. We now report the experimental results on the effectiveness of using an $r-$resilient pub-sub network 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).

Figure 8 shows 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 directly derived from Figure 8. Observe that the communication cost decreases as a increases. Also note that $a = NS$ (NS denotes the total number of subscribers) minimizes the communication cost but imposes heavy load on the publisher and the pub-sub nodes (load is proportional to a).

Figure 9 and 10 shows 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 subscribers that receive an event on topic w to $N(w)$, averaged over all topics. Observe from Figure 9 that one can improve resilience by increasing a at the cost of publisher and node load. This is equivalent to decreasing the network's height h thereby, making the network shallow and broad. Figure 10 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/node load.

6.2.2 Implementation based Experiments

In this section, we present 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 effectiveness of EventGuard against flooding based DoS attacks.

Experimental Setup. Our implementation of EventGuard

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

Table 4: EventGuard Storage Overhead: HT_{size} denotes the total size of the hashtable maintained for detecting flooding based DoS attacks (HT_{size} is at most a few tens of KBs)

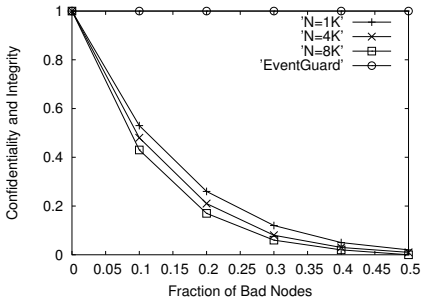

Figure 5: Confidentiality and Integrity

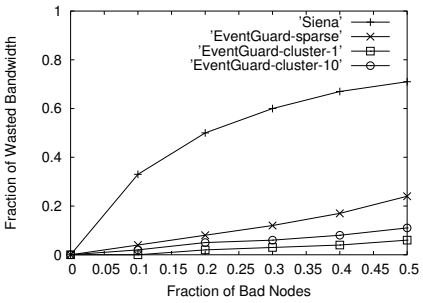

Figure 6: Flooding-based DoS Attack

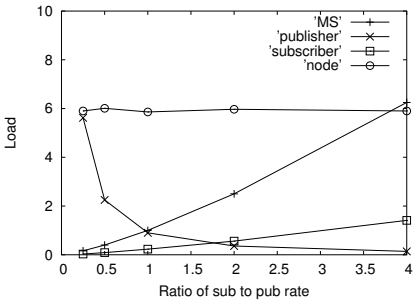

Figure 7: MS Load

is built on top of Siena pub-sub core. We ran this implementation of EventGuard on eight machines each with 8-processor (total of 64 processors) (550 MHz Intel Pentium III Xeon processors running RedHat Linux 9.0) connected via a high speed LAN. We instrumented the pub-sub core to simulate the wide-area network delays obtained from the GT-ITM topology generator.

The pub-sub network consisted for one publisher, 32 subscribers, and one MS, with the publisher as the root of the tree and the subscribers as leaves. We constructed complete binary tree topology using different number 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. Each entity (publisher, subscriber, node, and MS) was run on a separate processor.

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. 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 conclude that the node is saturated and the experiment aborted. We iteratively vary q across different experimental runs to identify the minimum value $q_{min} = throughput$ such that some node in the pub-sub network is saturated.

Figure 11 shows the maximum throughput versus the number of nodes in the pub-sub network for EventGuard and basic Siena. 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 primary overhead for EventGuard arises due to the verification of ElGamal signatures which is an expensive operation (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 11). 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 sent by a publisher till the time it is available to the subscriber (in plain-text). The latency was measured keeping the throughput at its highest (see Figure 11). Figure 12 shows latency versus the 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, and so does the latency. However, as the number of nodes increases further, the height of the dissemination tree also increases. An increase in height by one incurs an additional latency of 70ms (network latency), thereby increasing the overall latency. While the throughput always improves (until it saturates) with the number of nodes, the latency will first improve and then deteriorate. Thus, a careful choice of the number of pub-sub nodes is required in order to achieve high throughput with acceptable latencies.

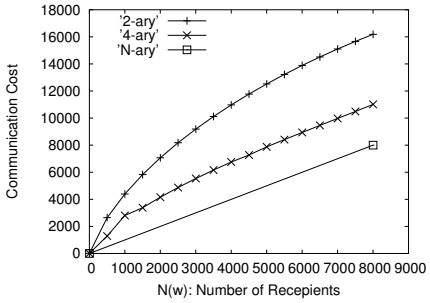

Figure 8: Communication Cost Vs Number of Recepients $N(w)$

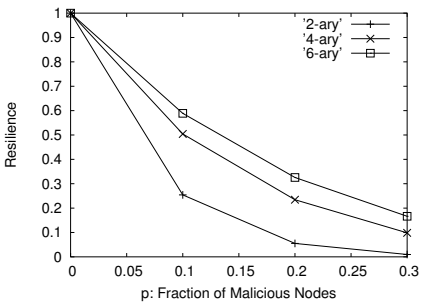

Figure 9: Resilience Vs a with $ind = 1$

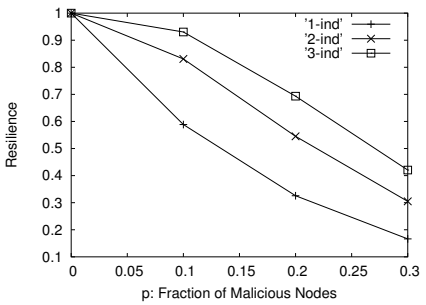

Figure 10: Resilience Vs ind with $a = 6$

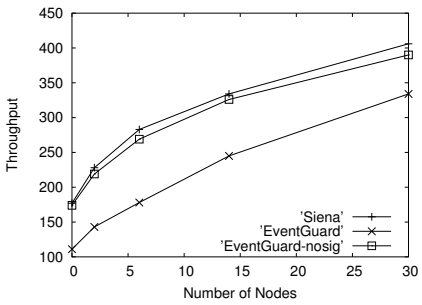

Figure 11: Throughput

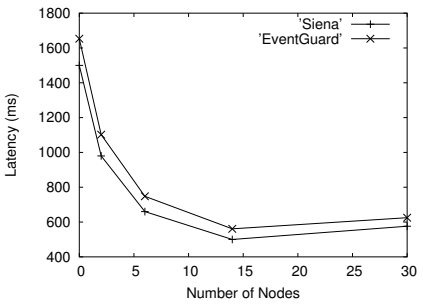

Figure 12: Latency

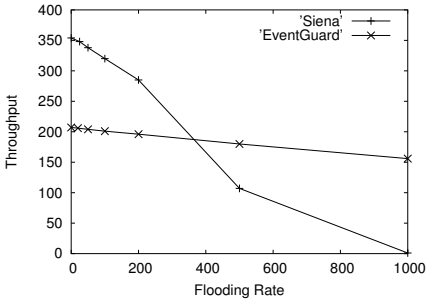

Figure 13: Resilience to Flooding-based DoS Attacks

Observe from Figure 12 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 1-2ms. In fact, the maximum increase in latency due to EventGuard is lesser 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 13 shows the throughput as fl increases both in the presence and the absence of routing guard.

Observe from Figure 13 that in the absence of routing guards the throughput deteriorates drastically with the injection of flooding-based DoS attack. In comparison EventGuard shows a much 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 DoS attack into a small neighborhood surrounding the malicious node (see Figure 3).

7 Related Work

Several pub-sub systems [5, 3, 7] have been developed to provide highly scalable and flexible messaging support for distributed systems. Siena [5] and Gryphon [3] are large pub-sub system capable of content-aware routing. Scribe [7] is an anonymous P2P pub-sub system. Most work on pub-sub systems have focused on performance, scalability and availability. Unfortunately, very little effort has been expended on studying the security aspects of these systems.

Significant amount of work has been done in the field of secure group communication on multicast networks (survey [16]). 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 arise 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 is not as flexible as the subscription model used in pub-sub systems. In contrast, EventGuard permits flexible membership at the granularity of subscriptions. Also, EventGuard uses an overlay network and does not rely on IP multicast technology primarily because the IP multicast protocol has not yet been deployed at an Internet scale.

Wang et al. [23] analyze the security issues and requirements in a content-based pub-sub system. Their paper identifies that the general security needs of a pub-sub application includes confidentiality, integrity and avail-

ability. 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. They identify that maintaining confidentiality against the pub-sub network nodes fundamentally conflicts with the pub-sub model as the pub-sub network routes information based on dynamic evaluations of publications against subscriptions. EventGuard shows that in most cases, one can achieve content-based routing while still maintaining confidentiality.

Opyrchal and Prakash [14] analyze secure distribution of events in a content-based pub-sub network from a group key management standpoint. 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 providing security to the subscribers while maintaining confidentiality even from the pub-sub network nodes.

8 Conclusion

We have presented *EventGuard*, a security framework for protecting pub-sub overlay services from various attacks such as authenticity, confidentiality, integrity, and resilience to DoS attacks. We have described the two main components of EventGuard: (1) a suite of security guards that can be seamlessly plugged-into a wide-area content-based pub-sub system and (2) a resilient pub-sub network design that is capable of secure and yet scalable message routing, countering message dropping-based DoS attacks. EventGuard presents a unified security framework that meets both the security goal of safeguarding the pub-sub overlay services from various attacks, and the performance goal of maintaining the system's overall simplicity, scalability and performance. We have demonstrated that EventGuard is easily stackable on any content-based pub-sub core by presenting a prototype implementation of EventGuard on top of Siena [5]. Our experimental evaluations show that EventGuard can secure a pub-sub overlay service with minimal performance penalty.

References

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

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

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

[4] A. Carzaniga. Siena - software. http://serl.cs.colorado.edu/ carzanig/siena/software/index.html.

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

[6] CNN. Gates: Buy stamps to send email. http://www.cnn.com/2004/TECH/internet/03/05/spam.charge.ap/.

[7] A. K. Datta, M. Gradinariu, M. Raynal, and G. Simon. Anonymous publish/subscribe in p2p networks. In *Proceedings of IPDPS*, 2003.

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

[9] FIPS. Data encryption standard (des). http://www.itl.nist.gov/fipspubs/fip46-2.htm.

[10] HMAC. Hmac: Keyed-hashing for message authentication. http://www.faqs.org/rfcs/rfc2104.html.

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

[12] MD5. The md5 message-digest algorithm. http://www.ietf.org/rfc/rfc1321.txt, 1992.

[13] NIST. Aes: Advanced encryption standard. http://csrc.nist.gov/CryptoToolkit/aes/.

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

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

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

[17] C. Raiciu and D. S. Rosenblum. A secure protocol for content-based publish/subscribe systems. http://www.cs.ucl.ac.uk/staff/C.Raiciu/files/secure_pubsub.pdf.

[18] SHA1. Us secure hash algorithm i. http://www.ietf.org/rfc/rfc3174.txt, 2001.

[19] M. Srivatsa, B. Gedik, and L. Liu. Scaling unstructured peer-to-peer networks with multi-tier capability aware topologies. In *Proceedings of Internation Conference on Parallel and Distributed Systems ICPADS*, 2004.

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

[21] M. Srivatsa and L. Liu. Eventguard: Securing publish-subscribe networks. Technical report, Georgia Institute of Technology, 2005.

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

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

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

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

Appendix A: Communication Cost

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 a $a-$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 $\lceil \log_a NS \rceil$. 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, we obtain the following constraint:

$$M^{tree}(w) \leq hN(w) \quad (1)$$

Also, the publication message is never required to traverse any link of the tree more than once. Hence,

$$M^{tree}(w) \leq \Sigma_{i=1}^{h} a^i = \frac{a}{a-1}(NS-1) \quad (2)$$

Combining the two constraints 1 and 2, we have:

$$M^{tree}(w) \leq min(hN(w), \frac{a}{a-1}(NS-1)) \quad (3)$$

Observe that the maximum communication cost for an $a-$ary tree occurs when $N(w) = NS$ and $M^{tree}_{max}(w) = \frac{a}{a-1}(NS-1)$. $M^{tree}_{max}(w)$ is minimum 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.

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. In a BFT propagation scheme, irrespective of the network topology (grid, tree) used for propagation each subscriber has to minimally receive $m+1$ identical publication messages. Hence the communication cost, denoted by $M^{bft}(w)$, satisfies the following condition: $M^{bft}(w) \geq (m+1)N(w)$. Assuming that $NS = 1000$ and about 10% of the nodes are malicious, $m = 100$, 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.

Appendix B: Independent Paths

Theorem 8.1 *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. ∎