Secure Event Dissemination in Publish-Subscribe Networks

Mudhakar Srivatsa and Ling Liu

College of Computing, Georgia Institute of Technology

{mudhakar, lingliu}@cc.gatech.edu

Abstract. Content-based publish-subscribe (pub-sub) systems are an emerging paradigm for building large-scale information delivery systems. Secure event dissemination in a pub-sub network refers to secure distribution of events to clients subscribing to those events without revealing the secret attributes in the event to the unauthorized subscribers and the routing nodes in a pub-sub network. A common solution to provide confidentiality guarantees for the secret attributes in an event is to encrypt so that only authorized subscribers can read them. There are two important challenges in building a secure and scalable content-based event dissemination infrastructure: (i) How to handle complex and flexible subscription models while preserving the efficiency and scalability of key management algorithms? (ii) How to securely route events on a pub-sub network while preserving the confidentiality of its attributes? In this paper, we describe the design and implementation of PSGuard, for secure event dissemination in pub-sub networks. PSGuard provides a careful combination of cryptographic and systems solutions to build an efficient and secure pub-sub networks. Concretely, we exploit hierarchical key derivation algorithms to encode publication-subscription matching semantics for scalable key management and develop a probabilistic multi-path event routing algorithm to minimize the amount of information that can be inferred by the routing nodes. An experimental evaluation of our prototype system shows that PSGuard meets the security requirements while maintaining the performance and scalability of a pub-sub network.

1 Introduction

A publish-subscribe (pub-sub) network is a wide-area communication infrastructure that enables information dissemination across geographically scattered and potentially unbounded number of publishers and subscribers. 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 in the form of subscription *filters*. For example, in a pub-sub system disseminating confidential medical records, an event could be: $e = \langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`age`, $25\rangle$, $\langle$`patientRecord`, $record\rangle\rangle$. An example subscription could consist of the following filter: $f = \langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $>$, $20\rangle\rangle$, where EQ denotes the keyword based matching operator and $>$ denotes the greater-than numeric operator.

Secure event dissemination in pub-sub networks refers to preserving the confidentiality secret attributes in an event from unauthorized subscribers and the routing nodes in the pub-sub network. 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 that the constraints in a subscription filter f without obtaining any information about the secret attribute `patientRecord`.

There are two important challenges for secure event dissemination in pub-sub networks: scalable key management and secure content-based routing. Most existing key management solutions for pub-sub networks use group key management protocols to manage subscriber grouped 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 setup static groups for every possible subgroup. Although some optimizations have been proposed for dynamic groups such as key caching [13], the worst case key management cost remains at $O(2^{NS})$ due to its inherent design.

In this paper, we propose to improve 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. We achieve this in two steps: (i) First, we associate an authorization key $K(f)$ with a subscription filter f and an encryption key $K(e)$ with an event e. We use the encryption key $K(e)$ to *encrypt* the secret attributes in an event e and the authorization key $K(f)$ to *decrypt* the secret attributes in a matching event e. (ii) We use hierarchical key derivation algorithms [24] 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. In this paper, we present a detailed quantitative analysis of our approach and show that it incurs a small computation, communication and storage cost that is independent of the number of subscribers, thereby making our approach very efficient and scalable.

The second challenge in secure event dissemination in pub-sub networks is content-based event routing. Most existing work either assume a secure multicast channel between the publishers and the subscribers or trust the pub-sub network nodes with the confidentiality of the events routed through them [23, 13]. We believe that a semi-honest model for the routing nodes is a more realistic assumption in a wide-area network environment. We argue that the group key management based solutions by design cannot simultaneously support in-network matching and secure content-based routing under a semi-honest model. In a group key management based solution, the encryption key (group key) for an event e is determined by the set of all authorized recipients (group) of the event e; the set of all authorized recipients of a event e is determined by matching the event e against all subscriptions; hence, all publication-subscription matching needs to be done at the publisher unless the publisher decides to make unencrypted events available to at least some of the routing nodes in the pub-sub network.

In this paper we augment our key management algorithms with a secure content-based routing algorithm. Our

approach can simultaneously support in-network matching and secure content-based routing since the encryption key for an event e is independent of the set of authorized recipients of the event e. 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. Recently, Perng et. al [14] proposed using mix networks to multicast events under a honest-but-curious model for the routing nodes. They discuss techniques to mitigate information leakages against an attack that uses a priori knowledge about the popularity distribution of events to break event confidentiality and anonymity. In this paper we augment their solution with a defense against timing analysis attacks that attempts to break the confidentiality of events using a priori knowledge about the frequency at which events are published. PSGuard uses probabilistic multi-path event routing algorithms 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 frequency of all tokens (routing labels on an event) appears (nearly) indistinguishable for all the routing nodes in the pub-sub network.

The rest of this paper is organized as follows. Section 2 sketches a reference pub-sub model and a threat model. Sections 3 and 4 present the concrete algorithms used by PSGuard for secure event dissemination. Section 5 presents a sketch of our implementation on the Siena pub-sub system followed by a detailed performance evaluation. We discuss related work in Section 6 and finally conclude in Section 7.

2 Reference Model

2.1 Content-Based Pub-Sub Model

Content-Based Routing. PSGuard uses a reference pub-sub model that is very similar to the Siena pub-sub network [8]. In this paper we assume a hierarchical pub-sub network topology, for the sake of simplicity. When a node n receives a subscription request $subscribe(m, f)$ from node m, node n registers the filter f with the identity of node m. If the filter f is not *covered* by any previously subscribed filters at node n, then node n forwards $subscribe(n, f)$ to its parent node. Formally, a filter $f = \langle name, op, value \rangle$ covers a filter $f' = \langle name', op', value' \rangle$ if $(name'\ op'\ value') \Rightarrow (name\ op\ value)$, where $\Rightarrow$ denotes Boolean implication. For example, a subscription $f = \langle \texttt{age}, >, 20 \rangle$ covers the subscription $f' = \langle \texttt{age}, >, 30 \rangle$. When a node n receives an event e, it forwards the event e to only those children nodes with a matching subscription. Performance evaluations from [8] show that content-based routing and in-network matching are vital for the performance and scalability of the pub-sub system.

Subscription Authorization. Pub-Sub system uses a key distribution center (KDC) to issue authorizations keys and thus restrict the set of events that can be read by a subscriber. This is similar to the KDC used by group key management protocols to update group keys as subscribers join/leave the system. Access control in a pub-sub system is specified at the granularity of a subscription filter. A subscriber receives one authorization key for every subscription filter that it is allowed to read. An authorization for a subscription filter is associated with a lifetime.

We use an epoch based approach wherein all authorization permits are valid for one time epoch. At the end of an epoch, the subscriber will have to obtain a new authorization permit (authorization key) to read events that match the subscription filter in the next epoch. Using an epoch based subscription model not only allows the authorization service to charge its subscribers on a periodic basis but also enables the it to revoke and regenerate the authorization keys periodically.

2.2 Threat Model

We assume an honest-but-curious model for the publishers, the subscribers and the routing nodes (discretionary access control model). A curious publisher may be interested in reading the events published by other publishers. For subscribers, authorization is defined on a per subscription basis and is valid within a one subscription epoch. A subscriber S is authorized to read an event e if the event e matches one of its active subscription filters. We assume that a subscriber S who is authorized to read an event e does not reveal its contents to other unauthorized subscribers (otherwise, this would be equivalent to solving the digital copyrights problem). However, unauthorized subscribers may be curious to read those events that do not match their subscriptions. Curious routing nodes in the pub-sub network may eavesdrop on the pub-sub messages routed through them. However, we assume that the pub-sub nodes are honest in routing messages from publishers to subscribers. For instance, we do not consider message dropping or malicous message forwarding based denial of service attacks in this paper, although these issues can be handled using solutions that are orthogonal to our proposal. Finally, we assume that the underlying IP-network may not offer any confidentiality or integrity guarantees.

3 Key Management

In PSGuard, 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 [24] 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 have developed hierarchical key derivation algorithms for constructing key spaces for different types of publication-subscription matching, including: topic or keyword based matching, numeric attribute based matching, category or ontology based matching, and string prefix/suffix matching. For example, in a numeric attribute based key space construction algorithm, an authorization key $K(f_1)$ associated with the filter $f_1 = \langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $>$, 20$\rangle\rangle$ and an authorization key $K(f_1')$ associated with the filter $f_1' = \langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $>$, 30$\rangle\rangle$ must be capable of deriving the encryption key $K(e_1)$ used for encrypting the message

msg in event $e_1 = \langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`age`, 35$\rangle$, $\langle$`message`, $msg\rangle\rangle$. On the other hand, key $K(f_1)$ should be capable of decrypting the message msg in event $e_1' = \langle\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`age`, 25$\rangle$, $\langle$`message`, $msg\rangle\rangle$, but the key $K(f_1')$ should not. In this section, we illustrate our approach using numeric attribute based matching.

3.1 Numeric Attribute Based Matching

Numeric attribute based matching supports range conditions on numeric attributes. Given a numeric attribute, say `age`, we can construct a subscription filter $f = \langle$`age`, $\in$, $(l, u)\rangle$ such that the filter f matches any event $e = \langle$`age`, $v\rangle$ if and only if $l \leq v \leq u$, that is, the attribute `age` takes a value v that belongs to the range (l, u) (both end points inclusive). To enable secure publication-subscription matching, we associate an authorization key $K(f)$ with every subscription filter f and an encryption key $K(e)$ with every event e that satisfy the following properties:

- Given $K(f)$ it should be computationally easy to derive a key $K(e)$, if $v \in (l, u)$.
- Given $K(f)$ it should be computationally hard to derive a key $K(e)$, if $v \notin (l, u)$.

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 numeric attribute key tree (NAKT). Given a subscription filter $\langle$`age`, $\in$, $(l, u)\rangle$, we use the NAKT to derive a small set of authorization keys that corresponds to the attribute num with the range (l, u), denoted by $K^{num}_{(l,u)}$, where num denotes the name of the numeric attribute. The NAKT enables one to easily (computationally) derive a key $K^{num}_{(l',u')}$ from $K^{num}_{(l,u)}$ if and only if $l \leq l' \leq u' \leq u$. For any event $e = \langle$`age`, $v\rangle$, we derive the encryption key $K(e) = K^{num}_{(v,v)}$ and encrypt the event e with $K(e)$. By the construction of the numeric attribute key tree it follows that $K(e)$ is efficiently derivable from $K(f)$ if and only if $l \leq v \leq u$.

Constructing Numeric Attribute Key Tree (NAKT). Now we need to discuss how to build the NAKT tree for a given numerical attribute num using hierarchical key derivation algorithms. Without loss of generality, we assume that the actual range of the numeric attribute num is $(0, |R(num)| - 1)$, where $|R(num)|$ denotes the size of range $R(num)$. Given a numeric value $v \in (0, |R(num)| - 1)$, we map it to a key tree identifier $ktid(v)$ which is a m-bit binary representation of the number $\lfloor \frac{v}{lc(num)} \rfloor$ and $m = \log_2(\frac{|R(num)|}{lc(num)})$. Note that $lc(num)$ denotes the smallest size of a subscription on numeric attribute num. Later in the section, we use $lc(num)$ to trade-off performance and expressiveness of the numeric attribute based matching algorithm. The key tree identifiers are arranged in the form of a binary tree with depth m. Figure 1 shows a numeric attribute key tree for $R(num) = (0, 31)$ and $lc(num) = 4$. Each element in the tree labeled $ktid$ has two attributes: a key K^{num}_{ktid} and a range of numeric values v such that $ktid(v)$ share the prefix $ktid$. The key tree is designed such that given a parent key all its children keys can be easily derived; but the converse is computationally infeasible. We also use the key tree identifier for tokenization and secure content-based routing in Section 4.

Let the symbol Ø ($null$) be used to label the root element of a NAKT. We derive the authorization key for the root element corresponding to the key tree as $K^{num}_{\emptyset} = KH_{K(w)}(num)$, where KH is a keyed pseudo-random function (approximated by HMAC-MD5 or HMAC-SHA1 [11]), $K(w) = KH_{rk(KDC)}(w)$ is the authorization key for the topic w, and $rk(KDC)$ denotes the secret key of the KDC. An example topic would be $w =$ `cancerTrail`

and numeric attribute num = `age`. For example, the key for the root element for the `age` numeric attribute is derived as $K_{\emptyset}^{\texttt{age}} = KH_{K(\texttt{cancerTrail})}(\texttt{age})$, where $K(\texttt{cancerTrail}) = KH_{rk(KDC)}(\texttt{cancerTrail})$. Then, we derive the key for an internal element with $ktid = \xi \parallel b$ recursively as $K_{\xi \parallel b}^{num} = H(K_{\xi}^{num} \parallel b)$, for some $\xi \in (0+1)^*$, $b = 0$ or 1 and H is a one-way hash function (approximated by MD5 [18] or SHA1 [10]). Note that $\parallel$ denotes string concatenation. For example, $K_0^{\texttt{age}}$ is derived as $K_0^{\texttt{age}} = H(K_{\emptyset}^{\texttt{age}} \parallel 0)$ and $K_1^{\texttt{age}}$ is derived as $K_1^{\texttt{age}} = H(K_{\emptyset}^{\texttt{age}} \parallel 1)$. Having described how to construct the numeric attribute key tree, we now describe techniques to select an encryption key $K(e)$ for an event e and an authorization key $K(f)$ for a filter f.

Encryption Key. A publisher P constructs the encryption key for a numeric attribute in a given event e as follows:

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

For example, a publisher P encrypts an event $e = \langle\langle$`publisher`, $P\rangle$, $\langle$`topic`, `cancerTrail`$\rangle$, $\langle$`age`, 22$\rangle$, $\langle$`message`, $msg\rangle\rangle$ is encrypted as follows. First, P identifies that the leaf node in the NAKT, which contains $v = 22$ has an identifier $ktid(22) = 101$ (see Figure 1). Then the publisher P encrypts the event e with the encryption key $K(e) = K_{101}^{\texttt{age}}$.

Authorization Key. A subscriber can subscribe for any range over the numeric attributes (limited by the least count $lc(num)$). The subscription range may span one or more elements in the NAKT. Given a range (l, u) we identify the smallest set of elements in the NAKT SS that spans the range (l, u) using a simple depth first search starting from the root of the NAKT. Then, we divide the subscription for the range (l, u) into multiple subscriptions, one for each sub-range in the set SS. For example, the smallest set of elements in the NAKT that spans the range $(8, 19)$ is $SS = \{(8, 15), (16, 19)\}$; hence, we split a subscription on the range $(8, 19)$ into two subscription ranges $(8, 15)$ and $(16, 19)$. Without loss of generality, we discuss how to generate authorization keys for subscriptions whose range spans exactly one element in the NAKT. The subscription f and its the authorization key $K(f)$ are as shown below.

$$\begin{aligned} f &= \langle\langle \texttt{topic}, EQ, w\rangle, \langle \texttt{num}, \geq, l\rangle, \langle \texttt{num}, \leq, u\rangle\rangle \\ K(f) &= K_{ktid(l,u)}^{num} \end{aligned}$$

For example, a subscriber S subscribes for a filter $f = \langle\langle$`topic`, EQ, `cancerTrail`$\rangle$, $\langle$`age`, $\geq$, 16$\rangle$, $\langle$`age`, $\leq$, 31$\rangle\rangle$ as follows. First, the authorization service identifies that element in the NAKT that matches the subscription range $(16, 31)$ as $ktid(16, 31) = 1$. Then the authorization service sends S the authorization key $K(f) = K_1^{\texttt{age}}$.

Matching Publications with Subscriptions using the NAKT. Given a publication with key tree identifier equal to $ktid_\alpha$ a subscriber who has subscribed for key tree identifier equal to $ktid_\phi$ does the following. The subscriber checks if $ktid_\phi$ is a prefix of $ktid_\alpha$. If so, the subscriber derives the encryption key $K_{ktid_\alpha}^{num}$ from the authorization key $K_{ktid_\phi}^{num}$. Note that the generation of children keys from its parent's key is computationally efficient because it uses a fast one-way hash function. However, it is computationally infeasible for a subscriber to derive the

keys corresponding to its ancestors or its siblings. For example, given a publication with $ktid_\alpha = 101$, a subscriber who has subscribed for $ktid_\phi = 1$ decrypts the message msg in a publication as follows. Given $ktid_\alpha = 101$ and $ktid_\phi = 1$, the subscriber first extracts the suffix 01. Then, S derives $K_{10}^{\texttt{age}} = H(K_1^{\texttt{age}} \parallel 0)$ and $K_{101}^{\texttt{age}} = H(K_{10}^{\texttt{age}} \parallel 1)$. Now, S can use $K_{101}^{\texttt{age}}$ to decrypt the secret attributes in the event.

Number of Authorization Keys. In general, if one uses a a-ary numeric attribute key tree ($a \geq 2$), any subscription range can always be subdivided into no more than $2(a-1)\log_a(\frac{|R(num)|}{lc(num)}) - 2$ sub-ranges. One can show that this is a monotonically increasing function in a (for $a \geq 2$) and thus has a minimum value when $a = 2$. Thus, a binary tree is optimal and it it requires no more than $2\log_2 \frac{|R(num)|}{lc(num)} - 2$ authorization keys for any given subscription range. In addition, one can also show that the average number of sub-ranges for a uniformly and randomly chosen subscription range of length ϕ_R is $\log_2 \frac{\phi_R}{lc(num)}$. Observe that the number of keys is at most logarithmic in $|R(num)|$; additionally one can control the number of keys by tuning the parameter $lc(num)$.

The key distribution center (KDC) expends computing power to generate keys for subscribers during the authorization phase. One can show that the maximum and the average cost of generating authorization keys for a subscription is $(4\log_2 \frac{|R(num)|}{lc(num} - 2)$ and $(\log_2 \frac{|R(num)|}{lc(num} + \log_2 \frac{\phi_R}{lc(num} - 1)$ hash operations respectively. The publishers and the subscribers expend computing power to derive keys for encrypting/decrypting events. One can show that the maximum and the average cost of deriving the encryption/decryption keys is $(\log_2 \frac{|R(num)|}{lc(num})$ and $(\log_2 \frac{\phi_R}{lc(num})$ hash operations respectively. Note that these average assume that the subscription range is chosen uniformly and randomly over $R(num)$.

Tables 1 and 2 shows the maximum and the average number of keys, key generation cost and key derivation cost for different values of $|R(num)|$ assuming $lc(num) = 1$. Observe that the number of authorization keys is very small. The key generation cost at the KDC is only of the order of few tens of microseconds and thus allowing the KDC to handle large subscription traffic. The key derivation cost is only a few microseconds, thereby adding minimal overhead to the throughput and latency of the published events.

We have described the design of our key derivation algorithm using numeric attribute based matching. We refer the readers to our technical report [1] for other types of publication-subscription matching and the techniques we have developed to handle complex subscriptions that include one or more of the above matching constraints combined using $\wedge$ and $\vee$ Boolean operators.

Unsubscription by Rekeying. In PSGuard, 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), 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 [25]. At the beginning of a new epoch, if the subscribers need to refresh their subscriptions then they must obtain new authorization keys from the KDC. 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

R	# Keys	Key Gen (μs)	Key Derive (μs)
10^2	12	23.66	6.37
10^3	18	34.58	9.10
10^4	26	49.14	12.74

Table 1: Max Cost

ϕ_R	# Keys	Key Gen (μs)	Key Derive (μs)
10	3.32	14.20	3.02
10^2	6.64	17.22	6.04
10^3	9.97	20.25	9.07

Table 2: Avg Cost: R = 10^4

length is outside the scope of this paper.

Multiple Publishers. When multiple publishers publish on a common topic, it might be essential to ensure that the publications from a publisher P is not readable by another publisher P'. PSGuard 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 KDC can generate per publisher authorization key for topic w as $K^P(w) = KH_{rk(KDC)}(P \parallel w)$. The KDC distributes $K^P(w)$ to a publisher. The KDC uses $K^P(w)$ to derive authorization keys for subscribers that subscribe to a topic w from publisher P. This only incurs almost no additional key generation cost. On the other hand, the subscriber group approach has to maintain separate groups for every publisher P.

3.2 Comparison with Subscriber Group Approach

3.2.1 Overview

In this section, we first illustrate (using examples) there important benefits of our approach over the traditional subscriber group based approach in terms of the key management cost. We then use a formal quantitative analysis to derive theoretical lower bounds on the performance and scalability benefits offered by our approach.

Number of Keys. First, let us suppose that a subscriber S has subscribed for a range $(0, R-1)$. Using the subscriber group based approach the number of keys is bounded by the number of possible subscription ranges and the number of subscription groups: $min(\frac{R(R-1)}{2}, 2^{NS})$. In contrast, PSGuard uses efficient key derivation algorithms to ensure the number of authorization keys is *independent* of the number of subscribers. Using the PSGuard approach the key server maintains only one key. The number of keys maintained by a subscriber S is at most logarithmic in R.

Communication Cost. Second, the subscriber group based approach would require changes to the groups and group keys whenever a new subscriber joins the system. For example, let subscriber S_1 subscribe for a range $(20, 30)$. We have one group $G = \{S_1\}$. Let us suppose that a new subscriber S_2 subscribes for a range $(25, 40)$, then we have three groups: $G_1 = \{S_1\}$ (for the range $(20, 25)$), $G_2 = \{S_1, S_2\}$ (for the range $(25, 30)$), and $G_3 = \{S_2\}$ (for the range $(30, 40)$). Observe that the group key server has to not only maintain more keys (computing and storage cost), but also update subscriber S_1 with new group keys (communication cost). On contrary, our approach requires no key updates and thus incurs much lower communication costs. We use a quantitative analysis to show that the PSGuard can offer 2-3 orders of magnitude reduction in communication costs.

KDC Scalability. Third, in the subscriber group approach, the key server has to maintain all subscriptions made by all active subscribers in order to determine the key updates (as illustrated above). The PSGuard approach

allows the key server to be *stateless* and ensures that the cost of handling a subscription request is very small (independent of NS). In the PSGuard approach, the key server does not have to maintain information about active subscriptions and active subscribers or update any authorization key as more subscribers join the pub-sub network. The stateless nature of our key server allows us to distribute and *on-demand* replicate the key server it to handle bursty loads. Note that the key server replicas need no consistency and concurrency control since they share no common state other than the master key $rk(KDC)$; recall that all authorization keys are derivable from $rk(KDC)$.

3.2.2 Quantitative Analysis

We now use an analytical model to compare the cost (messaging, computation and storage) of our approach versus the subscriber group approach (using lazy revocation). In order to make a fair comparison we assume that the time interval for lazy revocation in the subscriber group approach equals the length of one time epoch T. We assume an $M/M/N$ model for subscribers [25] with λ denoting the arrival rate per inactive subscriber and μ denoting the departure rate per active subscriber and N denotes the total number of subscribers (active + inactive). One can show that the average number of active subscribers at any point in time is $NS = N^* \frac{\lambda}{\lambda+\mu}$. In steady state, the average rate of subscribers joining the system = the average rate of subscribers leaving the system = $N^* \frac{\lambda\mu}{\lambda+\mu}$. Let us assume that we have only one topic and one numeric attribute whose range is of size R. Without loss of generality we assume that the least count parameter is set to one. Let ϕ_R denote the average size of a subscription range. We assume that the subscription ranges are chosen uniformly and randomly over $(0, R-1)$.

In the subscriber group approach, when a new subscriber S joins the system the KDC has to determine the number of active subscribers $NS_{overlap}$ who have an overlapping subscription range with S. A subscription for $(x_s, x_s+\phi_R)$ by S overlaps with $(x_{s'}, x_{s'}+\phi_R)$ by some other subscriber S' overlaps if $|x_s - x_{s'}| \leq \phi_R$, that is, if $x_{s'}$ lies beteween $x_s - \phi_R$ and $x_s + \phi_R$, the subscription ranges are guaranteed to overlap. Since subscription ranges are chosen uniformly, the probability that the subscription range of S overlaps with that of some active subscriber S' is $\frac{2\phi_R}{R}$. Since, the susbcriptions of any two subscribers are independently chosen, $NS_{overlap} \sim binomial(\frac{2\phi_R}{R}, NS)$ (if, $\phi_R < \frac{R}{2}$). Note that if $\phi_R \geq \frac{R}{2}$ the probability of overlap is one. Hence, the average number of overlapping subscribers $NS_{overlap} = NS * min(\frac{2\phi_R}{R}, 1)$. In the following portions of this section, we assume that $\phi_R < \frac{R}{2}$ to estimate $NS_{overlap}$ (if $\phi_R \geq \frac{R}{2}$, then $NS_{overlap} = NS$).

For every overlapping subscriber S', the number of keys that need to be updated is: zero if S subscribes for a superset of S', three to four if S subscribes to a subset of S' and two otherwise. The average number of keys that need to be updated per active subscriber with an overlapping range is two. This incurs a messaging cost of $2 * NS_{overlap}$ keys. In addition, the new subscriber S has to be sent $NS_{overlap}$ keys. Hence, the total messaging cost is $3 * NS_{overlap} = 3 * NS * \frac{2\phi_R}{R}$. Given a subscriber join rate of $N^* \frac{\lambda\mu}{\lambda+\mu}$ and a time interval of length T, the total messaging cost is $C_{subscribergroup} = N^* \frac{\lambda\mu}{\lambda+\mu} * T * 6 * NS * \frac{\phi_R}{R}$.

In PSGuard the average number of authorization keys for a uniformly and randomly chosen subscription range

ϕ_R (> 1) is $\log_2 \phi_R$. Note that this cost is independent of the number of active subscribers NS. The total messaging cost for one time epoch is $C_{psguard} = N * \frac{\lambda\mu}{\lambda+\mu} * T * \log_2 \phi_R$. Hence, $C_{subscribergroup} : C_{psguard} = 6 * NS * \frac{\phi_R}{R * \log_2 \phi_R}$. Observe if $\phi_R \ll R$, that there is very little or almost no overlap between the subscription ranges from any two subscribers) then the subscriber group approach may perform better than the PSGuard approach. Observe that if $\phi_R = R = 1$ presents the worst case scenario for the subscriber group approach since this would result in 100% overlap between any two subscription ranges.

One should observe that the uniform and random distribution for subscription ranges presents the best case scenario for the subscriber group approach since it increases the likelihood of smaller subscriber groups that contain mutually disjoint sets of subscribers. However, in most applications, subscriber interest follows auto-correlated heavy tailed distribution that allows a group of subscribers to share common interests. Formally, let us suppose that $f(x)$ denotes a probability density function that a subscriber susbcribes for a range $(x, x + \phi_R)$. Using the same analysis as above, the probability of overlap is given by $op = \sum_x (f(x) * \sum_{y=x-\phi_R}^{x+\phi_R} f(y))$. For the sake of simplicity let us suppose that $\phi_R \ll R$ such that $f(x)$ can be approximated to linear function over the small range $(x - \phi_R, x + \phi_R)$. In this case, the probability of overlap could be approximated to $op = 2\phi_R * \sum_x f(x)^2$. Given that $\sum_x f(x) = 1$, one can show that op is minimal when $f(x) = \frac{1}{R}$ for all x, that is, if $f(x)$ follows a uniform and random distribution. On the other hand, the PSGuard approach is *agnostic* to the distribution of subscriber interests. Hence, $C_{subscribergroup} : C_{psguard} = 6 * NS * \frac{\phi_R}{R * \log_2 \phi_R}$ represents an *absolute minima* for the cost ratio.

Tables 3 and 4 summarizes an analytical comparison of our PSGuard approach against the subscriber group approach. Note that H denotes the computation cost for a one-way hash function and D denotes the cost of a decryption operation. Tables 5 and 6 shows the lower bound on cost ratio for varying subscription range ϕ_R and the number of subscribers NS respectively. Table 5 shows that the subscriber group approach incurs at least 2-3 orders of magnitude higher cost than the PSGuard approach, clearly demonstrating the lack of scalability in the susbcriber group approach. Table 6 indicates that for $NS \leq 100$, the group key management approach may perform better; although it does not scale well for higher values of NS. However, in our experimental section we use a more realistic heavy tailed distribution (to model groups of subscribers with common interests) and show that the group key management approach may offer marginally better performance only when $NS \leq 8$.

3.2.3 Performance Enhancement

In our implementation and experiments, we have used a *key caching* mechanism to further decrease the computational overhead in the PSGuard approach. When a subscriber S derives an encryption key $K^{num}_{ktid_\alpha)}$ from an authorization key K^{num}_{ϕ} ($ktid_\phi$ is a prefix of $ktid_\alpha$) it caches all the intermediate keys computed in this process in its local key cache. Now, the subscriber S can compute an encryption key $K^{num}_{ktid_{\alpha'}}$ from a cached key $K^{num}_{ktid_{\phi'}}$ such that $ktid_\phi$ is a prefix of $ktid_{\phi'}$ which in turn is a prefix of $ktid_{\alpha'}$. Observe that computing $K^{num}_{\alpha'}$ from $K^{num}_{\phi'}$ costs $H^*(|ktid_{\alpha'}| - |ktid_{\phi'}|)$ and that from K^{num}_{ϕ} costs $H^*(|ktid_{\alpha'}| - |ktid_{\phi}|)$ and $|ktid_\phi| \leq |ktid_{\phi'}|$, where $|ktid|$ denotes the number of bits in $ktid$. Indeed $K^{num}_{ktid_{\phi'}}$ would be the *optimal* cached key to derive $K^{num}_{ktid_{\alpha'}}$ if $ktid_{\phi'}$ is a prefix of $ktid_{\alpha'}$ and $|ktid_{\alpha'}| - |ktid_{\phi'}|$ is minimal. One should note that such optimizations are more meaningful when the

	Join Message	Join Compute	Storage	Stateless
PSGuard	$\log_2 \phi_R$	$H * 2\log_2 \phi_R$	1	Yes
Subscriber Group	$6 * NS * \frac{\phi_R}{R}$	-	$2 * NS$	No

Table 3: KDC Costs

	Join Message New Subscriber	Join Message Active Subscribers	Storage	Event Processing
PSGuard	$\log_2 \phi_R$	-	$\log_2 \phi_R$	$D + H * \log_2 \phi_R$
Subscriber Group	$2 * NS * \frac{\phi_R}{R}$	$4 * NS * \frac{\phi_R}{R}$	$2 * NS * \frac{\phi_R}{R}$	D

Table 4: Subscriber Costs

events exhibit temporal locality. For example, let us consider stock quotes. Assuming that the stock price changes only nominally over small periods of time, two consequtive stock quote events are likely to carry prices that are numerically very close to one another.

4 Secure Content-Based Event Routing

We have so far described our approach to efficient and scalable key management for pub-sub systems. In this section, we present a secure content-based event routing algorithm that assumes a more realistic semi-honest model for the routing nodes. In this model, a routing node may be curious to infer information about the events routed on the pub-sub network. We argue that the group key management based approaches by design cannot simultaneously support in-network matching and secure content-based routing because the encryption key (group key) for an event e depends on the set of all recepients (group) of the event e. Hence, either the publisher has to perform all publication-subscription matching before disseminating the event on a pub-sub network (no in-network matching) or reveal the plain-text event to at least some routing nodes on the pub-sub network. Opyrchal and Prakash [13] uses the later strategy thereby benefiting from the scalability of in-network matching, but compromising the confidentiality of events routed on the pub-sub network.

Our key management algorithms make it feasible for us to simultaneously support in-network matching and secure content-based routing since the encryption key for an event e, namely $K(e)$, is independent of the set of the recepients of the event e. However, encrypting the entire event e with its encryption key $K(e)$ does not allow content-based routing and in-network matching. In this section, we first build techniques for content-based routing using tokenization. Using tokens for content-based routing and in-network matching allows the curious routing nodes to infer nothing more than the fact that an event matches a subscription. However, even such an approach

ϕ_R	$C_{subscribergroup} : C_{psguard}$
10	1.81
10^2	9.04
10^3	60.18
10^4	451.81

Table 5: Theoretical Lower Bound: $NS = 10^3$ and $R = 10^4$

NS	$C_{subscribergroup} : C_{psguard}$
10	0.09
10^2	0.90
10^3	9.04
10^4	90.36

Table 6: Theoretical Lower Bound: $\phi_R = 100$ and $R = 10^4$

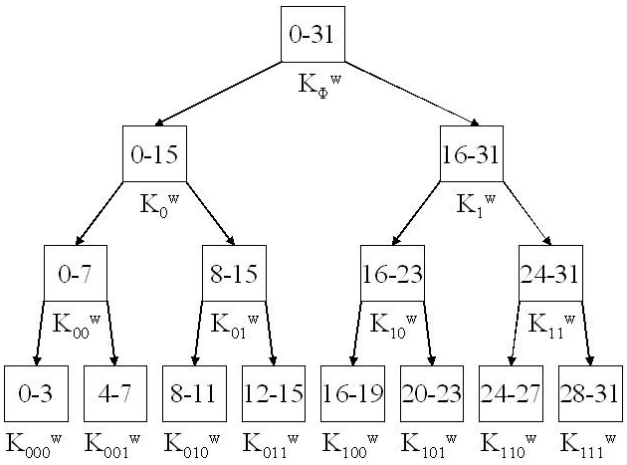

Figure 1: Key Tree: Range Queries on Numeric Attributes

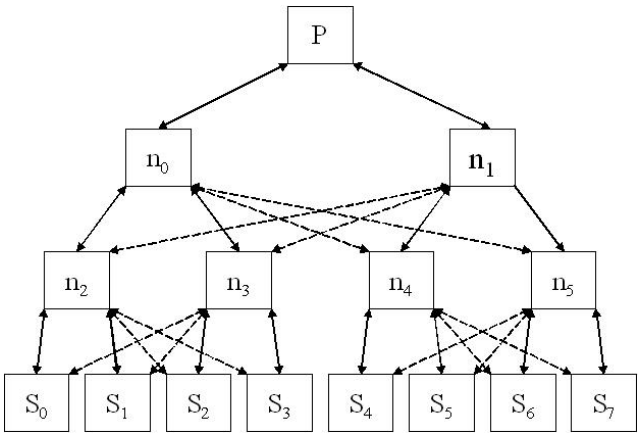

Figure 2: Probabilistic Multi-Path Event Routing ($ind = 2$)

is vulnerable to some inference attacks. Perng et. al [14] proposed using mix networks to multicast events under a honest-but-curious model for the routing nodes. They discuss techniques to mitigate information leakages against an attack that uses a priori knowledge about the popularity distribution of events to break event confidentiality and subscriber/publisher anonymity. In this paper we focus on timing analysis attacks that attempts to break the confidentiality of events using a priori knowledge about the frequency at which events are published. We present a probabilistic multi-path event routing algorithm to allow secure content-based routing by minimizing the amount of information that could be inferred by the routing nodes.

4.1 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 a solution proposed by Song et al. [19] for searches on encrypted data to construct our algorithm. Let us consider a topic w. The KDC generates a token $T(w)$ for the topic w using a PRF F as $T(w) = F_{rk(KDC)}(w)$, where $rk(KDC)$ is the KDC'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: $\langle r, F_{T(w)}(r)\rangle$, 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 $F_{tok}(r) = match$. For a proof of correctness please refer [19].

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. 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.

4.2 Probabilistic Multi-Path Event 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 [14]. 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 $\lambda'_t = \text{c}$ 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 more would 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.

In the following section, we present techniques to construct multiple independent paths to route an event from a publisher to all its subscribers. In our experimental section, we show the efficacy of using probabilistic multi-path event routing in minimizing the amount of information inferred by the curious routing nodes under both the collusive and non-collusive settings. Identifying other inference attacks on our routing algorithm and developing defenses is part of our on-going work.

4.2.1 Constructing a Multi-Path Event Dissemination Network

As described in our reference model (see Section 2), a pub-sub network typically constructs a tree-like topology with the publisher as its root, the subscribers as its leaves, and the routing nodes as intermediate elements in the tree. In this section, we modify a $a-$ary tree such that it has ind independent paths (for any $ind \leq a$). If we have ind independent paths between a publisher P and a subscriber S, then ind colluding nodes (one per independent path)

are required obtain complete information about the frequency of a token routed from publisher P to subscriber S. By randomly distributing the nodes in a pub-sub network, one could ensure that amount of information inferred by malicious routing nodes is minimal. For the sake of simplicity, we illustrate our technique by modifying a binary tree ($a = 2$) network to yield a network with $ind = 2$.

Figure 2 shows the key idea behind constructing a multi-path event dissemination network G^2. Let d denote to the maximum depth of tree, with root (publisher) at depth 0 and the leaves (subscribers) at depth d. 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. We add one additional edge to every subscriber and every node in the system. Concretely, for every node n (and the subscriber S) we add an additional edge from n to $sibling(parent(n))$. The solid lines in the Figure 2 shows the original edges in a binary tree network and the dashed lines indicate additional edges added to the network. We now claim that the network G^2 has the following property.

Claim 4.1 *G^2 has $ind = 2$ independent paths from the publisher P to every subscriber.*

We prove Claim 4.1 using Theorem 4.2 which explicitly constructs two independent paths from the publisher (root) to any subscriber (leaf) on the network.

Theorem 4.2 *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 graph G. 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 network G^2.*

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 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 $ind \leq a$. We construct a network G^{ind} by connecting all nodes n to $parent(n)$ and $ind - 1$ distinct siblings of $parent(n)$ (these siblings indeed exist since $ind \leq a$).

Claim 4.3 *G^{ind} has ind independent paths from the publisher P to every subscriber.*

As ind increases the cost of setting up the event dissemination routes on the pub-sub network increases. Nonetheless, the cost of actually routing events through these routes does not increase due to probabilistic multi-path event routing. While there are ind possible paths to route an event, the publisher uniformly and randomly chooses only one path to route the event. Additionally, one could easily extend our probabilistic multi-path routing algorithm

to route an event on two or more independent paths (in parallel). This would make our event dissemination system more fault tolerant and resilient to message dropping based denial of service (DoS) attacks by malicious routing nodes. Nonetheless, in this paper we focus only on semi-honest (honest-but-curious) routing nodes.

In our implementation, we keep the network construction affordable by limiting the maximum number of independent paths to ind_{max}. However, if ind_{max} is not large enough then $S_{app} < S_{max}$, that is, the malicious routing nodes would be able to infer some information. Nonetheless, if S_{app} is sufficiently larger than S_{act} then the amount of information inferred by malicious routing nodes is largely constrained. In our experimental section, we measure the apparent entropy S_{app} as observed by the routing nodes under both collusive and non-collusive settings.

5 Implementation and Evaluation

5.1 Prototype Sketch

We have implemented PSGuard on top of an unmodified Siena pub-sub core [8]. Siena is a content-based pub-sub system whose working is very similar to our reference model in Section 2. 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. This is because PSGuard uses the same in-network matching operators as those supported by the Siena pub-sub core. We have implemented the authorization service (KDC) as a stand-alone entity. The KDC computes the authorization keys on the fly since the key derivation cost is fairly low. For a KDC with limited computing power, one could cache the derived keys to trade-off 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.

5.2 Experimental Results

The experimental results presented in this section are obtained from our prototype implementation of PSGuard on the Siena pub-sub core. First, we compare our key management algorithms with the subscriber group based approach. Second, we measure the efficacy of probabilistic multi-path event routing for secure content-based routing. Third, we present measurements of PSGuard on the throughput and latency of the pub-sub network.

Experimental Setup. We used GT-ITM [26] 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 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. We ran our implementation of PSGuard on eight 8-processor servers (64 CPUs) (550 MHz Intel Pentium III Xeon processors running RedHat Linux 9.0) connected via a high speed LAN.

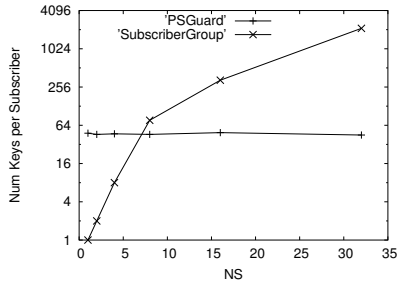

Figure 3: Num Keys per Subscriber

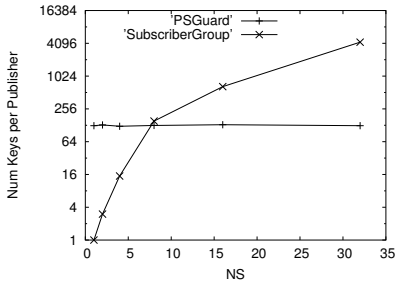

Figure 4: Num Keys per Publisher

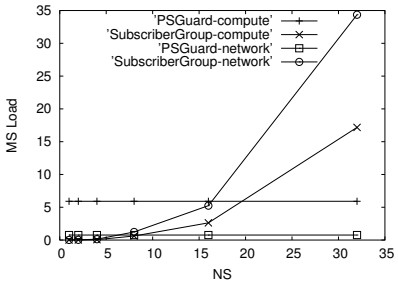

Figure 5: KDC Load

We simulated the wide-area network delays obtained from the GT-ITM topology generator.

All experimental results presented in this section were averaged over 5 independent runs. Due to the lack of real workloads in this area, we had to use a synthetic workload. We simulated 128 topics, with the popularity of each topic varying according to a Zipf-like distribution [16]. 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 rest 32 were simple topics (see Section 3.1 for examples). 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.

5.2.1 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 3 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 [13] that require 2^{NS} keys in the worst case. PSGuard 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 PSGuard approach. PSGuard 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 4 shows the average number of keys maintained per publisher as NS the number of subscribers varies. The trends shown in Figure 4 are very similar to that in 3.

KDC Load. Figure 5 shows the computing and network cost on the key server using `SubscriberGroup` based approach and `PSGuard`. Computing cost (measured in milliseconds) shows the average cost of group key management

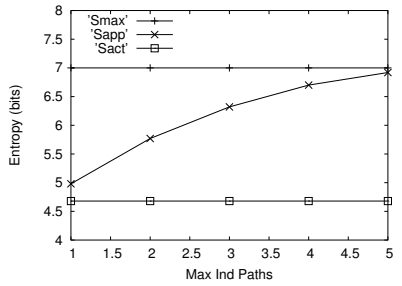

Figure 6: Secure Content-Based routing under a Non-Collusive Setting

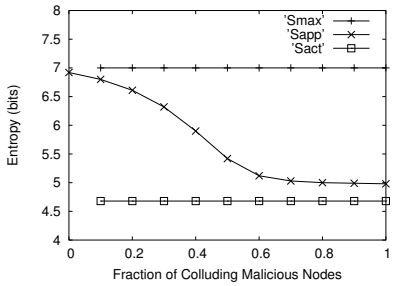

Figure 7: Secure Content-Based routing under a Collusive Setting

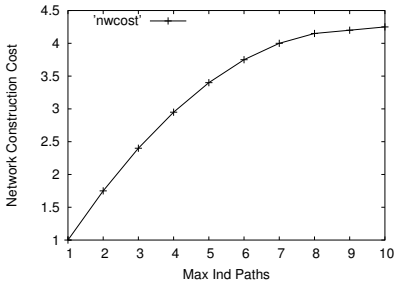

Figure 8: Cost of Constructing a Multi-Path Event Routing Network

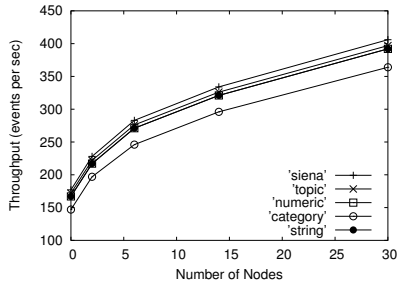

Figure 9: Throughput

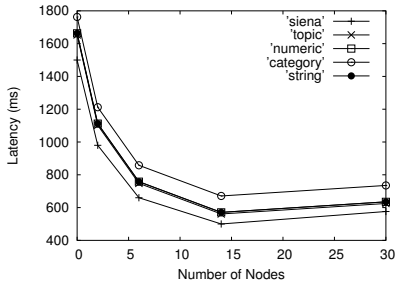

Figure 10: Latency

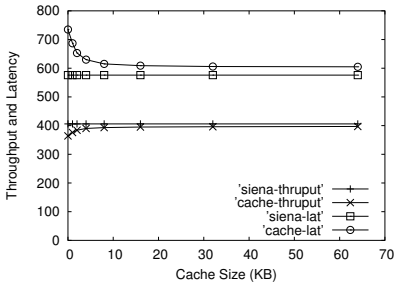

Figure 11: Key Caching

in `SubscriberGroup` and the cost of key derivation algorithm in `PSGuard` when a new subscriber joins the system. The cost incurred by the `SubscriberGroup` increases dramatically with NS, while that incurred by the `PSGuard` 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 `PSGuard`. Similar to computing cost, `PSGuard` incurs a small and constant networking cost, while that of `SubscriberGroup` explodes with NS.

5.2.2 Secure Content-Based Routing

This section presents experimental results on the efficacy of probabilistic multi-path event routing in defending against frequency inference attacks.

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 6 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 smoothen 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 7 shows the efficacy of probabilistic multi-path event routing against collusive routing nodes. As the fraction of collusive nodes increases, it is more likely that two or more colluding nodes are on two or more independent path 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.

Multi-Path Network Construction Cost. Figure 8 shows the cost of constructing a pub-sub network for different values of maximum number of independent paths (ind_{max}). The values shown in Figure 8 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 8 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.

5.2.3 Throughput and Latency

This section presents measurements on the throughput and latency of the pub-sub network with and without PS-Guard.

Throughput. We measured the throughput in terms of the maximum number of publications per second that can be handled by the pub-sub system. 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 of $q_{min} = throughput$ such that some node in the pub-sub network is saturated. Figure 9 shows the maximum throughput versus the number of nodes in the pub-sub network. Observe that the throughput of the system increases with the number of nodes. This demonstrates the scalability

of the pub-sub system and the PSGuard approach. The throughput drop for topic attributes, numeric attributes and string attributes is less than 2% below that of Siena while that for category attributes is about 11% below that of Siena.

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). Figure 10 shows latency versus number of nodes. The latency is measured keeping the throughput of the system at its maximum. Observe that latency first decreases and then begins to increase with the number of nodes. Initially, latency decreases because the per-node processing cost decreases. However, as the number of nodes continues to increase, so does the diameter of the pub-sub network. Hence, distance between the publisher and subscriber (in terms of the number of pub-sub network hops) increases with the number of nodes. This is because the wide-area network latencies are of the order of 70ms; while the overhead for encryption/decryption and key derivation is relatively much smaller. The increase in latency for topic attributes, numeric attributes and string attributes is less than 1.5% while that for category attributes is about 6% above that of Siena.

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 KDC, the publishers and the subscribers. Figure 11 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 PSGuard. Using the key caching mechanism the throughput of PSGuard was about 2.2% (as against 10.8% without caching) lower than Siena and the latency of PSGuard was about 1.5% (as against 5.7% without caching) higher than Siena (using a 64 KB cache).

6 Related Work

Several pub-sub systems [8, 5, 9, 3] have been developed to provide highly scalable and flexible messaging support for distributed systems. Siena [8] and Gryphon [5] are large pub-sub system capable of content-aware routing. Scribe [9] 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. Wang et al. [23] 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 includes confidentiality, integrity and availability. 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.

Significant amount of work has been done in the field of secure group communication [2, 21, 22, 24, 12, 6, 7, 15] on multicast networks (survey [17]). Such systems leverage secure group-based multicast techniques and group key management techniques to provide forward and backward security, scalability and performance. 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 this paper, we have analytically and experimentally demonstrated the performance and scalability issues in using traditional group key management protocols in pub-sub networks.

Opyrchal and Prakash [13] uses a key caching based optimization technique to alleviate the cost of group key management. However, their approach requires that the pub-sub network nodes (brokers) are completely trustworthy. We use a more realistic semi-honest model for the pub-sub network nodes. We have also demonstrated the infeasibility of group key management approach to simultaneously deal with in-network matching and secure content-based routing under a semi-honest model for routing nodes.

Perng et al. [14] 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 [24] to develop key management algorithms primarily in the domain of file systems [4]. To the best of our knowledge this is the first paper that applies hierarchical key derivation algorthms to enforce access control in pub-sub systems. However, our solutions do not apply to all pub-sub matching operators, although it covers most of the popular ones [8]. 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 a subscription fee per epoch.

Our earlier work [20] focused on guarding a pub-sub network from denial of service (DoS) attacks and hard proposed several techniques to safeguard a pub-sub network against message spoofing, spamming, and flooding attacks, addressing the issue of maintaining authentication and availability of publications and subscriptions. In contrast, PSGuard presented in this paper focuses solely on secure content-based event disseminatin using a semi-honest pub-sub network. Our ongoing research is to develop a secure pub-sub infrastructure that integrates [20] with PSGuard.

7 Conclusion

We have presented PSGuard − an efficient and secure event dissemination mechanism for pub-sub networks. PSGuard includes a novel key management algorithm for guaranteeing event confidentiality. Instead of associating keys with users or groups of users in group key management approaches, PSGuard associates authorization keys with subscriptions and encryption keys with publications and uses secure key derivation algorithms for scalable key management. Through simple analytical models and experimental analysis we show that PSGuard offers significantly reduces the communication, storage and state maintenance costs, albeit incurring a small computational

overhead. PSGuard simultaneously supports in-network matching and secure content based event routing using tokenization and a novel probabilistic multi-path event routing algorithm. The multi-path event routing algorithm, though incurring higher construction cost, adds no additional messaging cost or latency. A concrete implementation on top of the Siena pub-sub network and a detailed evaluation of our prototype have shown that PSGuard incurs very low key management costs for secure content-based routing, while maintaining a low throughput (2.2%) and latency (1.5%) overhead on the pub-sub network.

References

[1] Scalable access control in content-based publish-subscribe systems. Technical Report GIT-CERCS-06-05, Georgia Institute of Technology, 2006.

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

[3] 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.

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

[5] 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.

[6] R. Canetti, J. Garay, G. Itkis, and D. Micciancio. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of the IEEE INFOCOM, Vol. 2, 708-716*, 1999.

[7] R. Canetti, T. Malkin, and K. Nissim. Efficient communication-storage tradeoffs for multicast encryption. In *Advances in Cryptology - EUROCRYPT. J. Stem, Ed. Lecture Notes in Computer Science, vol. 1599, Springer Verlag, pp: 459-474*, 1999.

[8] 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.

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

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

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

[12] D. A. McGrew and A. T. Sherman. Key establishment in large dynamic groups using one-way function trees. In *Tech. Rep. No. 0755 (May), TIS Labs at Network Associates, Inc., Glenwood, MD.*

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

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

[15] A. Perrig, D. Song, and J. D. Tygar. ELK: A new protocol for efficient large group key distribution. In *Proceedings of IEEE Symposium on Security and Privacy*, 2001.

[16] 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.

[17] 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.

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

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

[20] M. Srivatsa and L. Liu. Securing publish-subscribe overlay services using eventguard. In *Proceedings of ACM CCS*, 2005.

[21] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The versakey framework: Versatile group key management. In *IEEE Journal on Selected Areas in Communications (Special Issue on MiddleWare) 17, 9(Aug), 1614-1631*, 1999.

[22] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. In *RFC 2627*, 1999.

[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 (HICSS-35)*, 2002.

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

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

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