Providing basic security mechanisms in broker-less publish/subscribe systems

Muhammad Adnan Tariq, Boris Koldehofe, Ala' Altaweel, Kurt Rothermel
University of Stuttgart
first name.last name@ipvs.uni-stuttgart.de

ABSTRACT

The provisioning of basic security mechanisms such as authentication and confidentiality is highly challenging in a content-based publish/subscribe system. Authentication of publishers and subscribers is difficult to achieve due to the loose coupling of publishers and subscribers. Similarly, confidentiality of events and subscriptions conflicts with content-based routing. In particular, content-based approaches in broker-less environments do not address confidentiality at all. This paper presents a novel approach to provide confidentiality and authentication in a broker-less content-based publish-subscribe system. The authentication of publishers and subscribers as well as confidentiality of events is ensured, by adapting the pairing-based cryptography mechanisms, to the needs of a publish/subscribe system. Furthermore, an algorithm to cluster subscribers according to their subscriptions preserves a weak notion of subscription confidentiality. Our approach provides fine grained key management and the cost for encryption, decryption and routing is in the order of subscribed attributes. Moreover, the simulation results verify that supporting security is affordable with respect to the cost for overlay construction and event dissemination latencies, thus preserving scalability of the system.

Keywords

Publish/subscribe, P2P, Security

1. INTRODUCTION

The publish/subscribe communication paradigm has gained high popularity because of its inherent decoupling of publishers from subscribers in terms of time, space and synchronization. Publishers inject information into the publish/subscribe system, and subscribers specify the events of interest by means of subscriptions. Published events are routed to their relevant subscribers, without the publishers knowing the relevant set of subscribers, or vice versa. This decoupling is traditionally ensured by intermediate routing over a broker network (e.g. [6, 9]). In more recent systems, publishers and subscribers organize themselves in a broker-less routing infrastructure, forming an event forwarding overlay (e.g. [7, 1]).

Content-based publish/subscribe is the variant which provides the most expressive subscription model, where subscriptions define restrictions on the message content. Its expressiveness and asynchronous nature is particularly useful for large-scale distributed applications with high-volume data streams. Examples of such stream-based applications include news distribution, stock exchange, environmental monitoring, traffic control, and public sensing. Not surprisingly, publish/subscribe needs to provide supportive mechanisms to fulfill the basic security demands of these applications such as access control and confidentiality.

Access control in the context of publish/subscribe system means that only authenticated publishers are allowed to disseminate events in the network and only those events are delivered to authorized subscribers. Similarly, the content of events should not be exposed to the routing infrastructure and a subscriber should receive all relevant events without revealing its subscription to the system. These security issues are not trivial to solve in a content-based publish/subscribe system and pose new challenges. For instance, end-to-end authentication using a *public key infrastructure (PKI)* conflicts with the loose coupling between publishers and subscribers. Publishers must maintain the public keys of the interested subscribers in order to encrypt events. Subscribers on the other hand, must know the public keys of all the relevant publishers in order to verify the authenticity of the received events. Similarly, traditional mechanisms to provide confidentiality by encrypting the whole event message conflicts with the content-based routing paradigm. Therefore, new mechanisms are needed to route encrypted events to subscribers without knowing their subscriptions and to allows subscribers and publishers authenticate each other without knowing each other.

In the past, most research has focused on providing expressive and scalable publish/subscribe systems. So far little attention has been given to security issues. Existing approaches towards secure publish/subscribe systems mostly rely on the presence of a traditional static broker network [17, 21]. These either address security under restricted expressiveness by providing security for topic-based publish/subscribe [21] or rely on a dedicated network of trusted brokers that are in charge to provide content-based matching and routing [16, 14]. Furthermore, existing approaches use coarse grain epoch-based key management and cannot provide fine grain access control in a scalable manner [21, 17, 22]. Nevertheless, security in broker-less publish/subscribe systems, where the subscribers are clustered according to their subscriptions have not been discussed yet in literature.

In this paper, we present a new approach to provide authentication and confidentiality in a broker-less publish/subscribe system. Our approach allows subscribers to maintain credentials according to their subscriptions. Private keys assigned to the subscribers are labelled with the credentials. A publisher associates each encrypted event with a set of credentials. We adapted identity based encryption mechanisms [2, 10], i) to ensure that a particular subscriber can decrypt an event only if there is match between the credentials associated with the event and the key and, ii) to allow subscribers to verify the authenticity of received events. Furthermore, we address the issue of subscription confidentiality in the presence of semantic clustering of subscribers. A weaker notion of subscription confidentiality is defined and a secure connection protocol is designed to preserve the weak subscription confidentiality. Finally, the evaluations demonstrate the viability of the proposed security mechanisms.

2. SYSTEM MODEL

2.1 Content-based publish/subscribe

For the routing of events from publishers to the relevant subscribers we use the content-based data model. The *event space*, denoted by Ω, is composed of a global ordered set of d distinct attributes (A_i): $\Omega = \{A_1, A_2, \ldots, A_d\}$. Each *attribute* A_i is characterized by a unique *name*, its *data type* and its *domain*. The data type can be any ordered type such as integer, floating point and character strings. The domain describes the range $[L_i, U_i]$ of possible attribute values. A *subscription filter* f is a conjunction of predicates, i.e., $f = \{Pred_1 \wedge Pred_2 ... \wedge Pred_j\}$. $Pred_i$ is defined as a tuple (A_i, Op_i, v_i), where Op_i denotes an operator and v_i a value. The operator Op_i typically includes equality and range operations for numeric attributes and prefix/suffix operations for strings. An *event* consists of attributes and associated values. An event is *matched* against a subscription f (and subsequently delivered to the subscriber), if and only if the value of attributes in the event satisfies the corresponding constraints imposed by the subscription. The subscription containment relationship can be defined similar to Siena [6]. Let E_{f_1} and E_{f_2} denote the sets of events matching subscription f_1 and f_2, respectively. Then f_1 is said to be *covered* by another subscription f_2, if $E_{f_1} \subseteq E_{f_2}$ holds.

We consider publish/subscribe in a setting where there exits no dedicated broker infrastructure. Publishers and subscriber contribute as peers to the maintenance of an self-organizing overlay structure. Peers can join the overlay by contacting an arbitrary peer and thereafter subscribe and publish events. In order to authenticate publishers we use the concept of advertisements in which a publisher announces beforehand the set of events which it intends to publish.

2.2 Attacker model

Our attacker model is similar to that defined by EventGuard [21]. There are two entities in the system: publishers and subscribers. Both the entities are computationally bounded and do not trust each other. Authorized publishers only disseminate valid events in the system. However, malicious publishers may masquerade the authorized publishers and spam the overlay network with fake and duplicate messages. We do not intend to solve the digital copyright problem, therefore authorized subscribers do not reveal the content of successfully decrypted events to other subscribers. Malicious subscribers on the other hand are curious to discover the subscriptions of other subscribers and published events to which they are not authorized to subscribe. Furthermore, passive attackers outside the publish/subscribe overlay network can eavesdrop the communication and try to discover content of events and subscriptions.

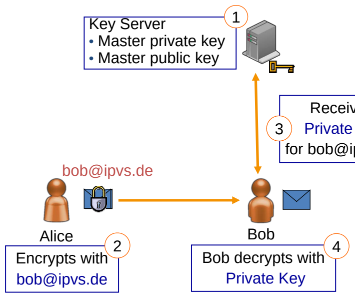

Figure 1: Identity-based encryption [18]

Finally, we assume presence of a secure channel for the distribution of keys from the key server to the publishers and subscribers. The secure channel can be easily realized by using transport layer security mechanisms such as SSL. No other requirement is placed on the underlying network infrastructure regarding confidentiality, authenticity or integrity of exchanged data.

2.3 Security goals and requirements

There are three major goals for the proposed secure publish/subscribe system, namely to support authentication, confidentiality and scalability:

Authentication: In order to avoid non-eligible publications only authorized publishers should be able to publish events in the system. Similarly, subscribers should only receive those messages to which they are authorized to subscribe.

Confidentiality: In a broker-less environment, two aspects of confidentiality are of interest: i) the events are only visible to authorized subscribers and are protected from illegal modifications, ii) the subscriptions of subscribers are confidential and unforgeable.

Scalability: The secure publish/subscribe system should scale with the number of subscribers in the system. Three aspects are important to preserve scalability: i) the number of keys to be managed and the cost of subscription should be independent of the number of subscribers in the system, ii) the key server and subscriber should maintain small and constant numbers of keys per subscription, iii) the overhead because of re-keying should be minimized without compromising the fine grained access control.

A side objective to above goals is to increase the availability of the system by mitigating the effects of event flooding and selective event drop based denial of service (DoS) attacks.

2.4 Drawbacks of traditional security mechanisms

Authentication using existing public key encryption mechanisms such as PKI allow a party to encrypt data to a particular user. Senders and receivers are strongly coupled, i.e. before a sender can encrypt a message, the receiver must generate a public/private key pair, sign its public key by a certificate authority and communicate it to the sender. Furthermore, PKI is inefficient for a large number of subscribers as each event needs to be encrypted with each subscriber's individual public key. Therefore, a mechanism is needed to enables any pair of users to securely communicate and verify each other's signatures without the need to exchange private or public keys.

It is very hard to provide subscription confidentiality in a broker-less publish/subscribe system, where the subscribers are arranged in an overlay network according to the containment relationship between their subscriptions. In this case, regardless of the cryptographic primitives used, the maximum level of attainable confidentiality is very limited. The limitation arises from the fact that a parent can decrypt every event it forwarded to its children. Therefore, mechanisms are needed to provide a weaker notion of confidentiality.

2.5 Identity-based encryption

While a traditional PKI infrastructure requires to maintain for each identity a private/public key pair which has to be known between communicating entities to encrypt and decrypt messages, *Identity-based encryption* [19, 3] provides a promising alternative to reduce the amount of keys to be managed.

In identity-based encryption (IBE), any valid string which uniquely identifies a user can be the public key of the user. A key server maintains a single pair of public and private master keys. The master public key can be used by the sender to encrypt and send the messages to a user with any identity, e.g. an email address. To successfully decrypt the message, a receiver needs to obtain a private key for its identity from the key server. Figure 1 shows the basic idea of using identity-based encryption.

We want to stress here that although identity-based encryption at the first glance, appears like a highly centralized solution, its properties are ideal for highly distributed applications. A sender needs to know only a single master public key in order to communicate with any identity. Similarly, a receiver only obtains private keys for its own identities. Furthermore, an instance of central key server can be easily replicated within the network. The replicas can function independently without having to interact with each other. This enables key servers to be created on demand for load balancing and reliability. Finally, a key server maintains only a single pair of master keys and therefore, can be realized as a smart card, provided to each participant of the system. It is interesting to note that in case of replicated key servers, changing master key pair requires one secure transfer to each replica and therefore, the cost is in order of the number of replicas. Smart cards on the other hand should be physically replaced. However their use is still practical as the master key pair is usually secure and is seldom changed.

Although identity-based encryption has been proposed some time ago, only recently *pairing-based cryptography* has laid the foundation of practical implementation of *identity-based* encryption. The main idea behind pairing-based cryptography is to establish a mapping between two cryptographic groups. This allows the new cryptographic schemes

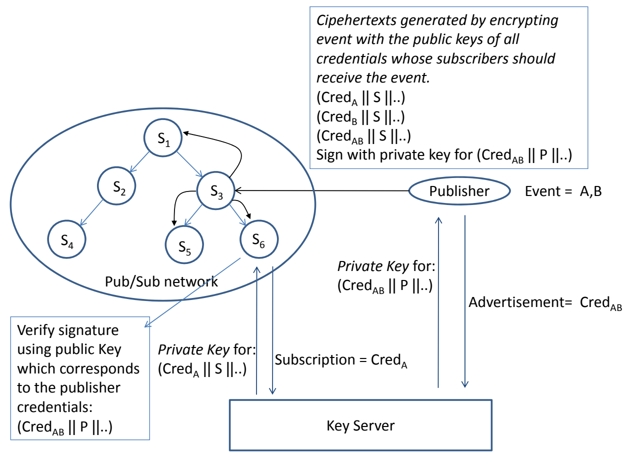
Cipehertexts generated by encrypting event with the public keys of all credentials whose subscribers should receive the event.
(Cred$_A$ || S ||..)
(Cred$_B$ || S ||..)
(Cred$_{AB}$ || S ||..)
Sign with private key for (Cred$_{AB}$ || P ||..)
S$_1$
S$_2$
S$_3$
S$_4$
S$_5$
S$_6$
Publisher
Event = A,B
Pub/Sub network
Private Key for: (Cred$_{AB}$ || P ||..)
Advertisement= Cred$_{AB}$
Verify signature using public Key which corresponds to the publisher credentials: (Cred$_{AB}$ || P ||..)
Private Key for: (Cred$_A$ || S ||..)
Subscription = Cred$_A$
Key Server

Figure 2: Approach overview: Publisher has credentials to publish events with two attributes A and B, Subscriber S_6 has credentials to receive events with attribute A

based on the reduction of one problem in one group to a different usually easier problem in another group. The mapping between cryptographic groups is achieved by means of *bilinear maps*, a technique we also apply subsequently for establishing the basic security mechanisms in the publish/subscribe system and therefore introduce here the main properties.

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be cyclic group of order p, where q is some large prime. A bilinear map is a function $e : \mathbb{G}_1 \text{x}\ \mathbb{G}_1 \rightarrow \mathbb{G}_2$ that associates a pair of elements from $\mathbb{G}_1$ to elements in $\mathbb{G}_2$. A bilinear map satisfies the following conditions:

1. *Bilinearity:* $e(u^x, v^y) = e(u^y, v^x) = e(u, v)^{xy}$, for all $u, v \in \mathbb{G}_1$ and $x, y \in \mathbb{Z}_p$.

2. *Non-degeneracy:* $e(u, v) \neq 1$, for all $u, v \in \mathbb{G}_1$.

3. *Computability:* e can be efficiently computed.

3. APPROACH OVERVIEW

For providing security mechanisms in publish/subscribe we rely on the methods of identity-based encryption and adapt it in order to support many-to-many interactions between subscribers and publishers. Publishers and subscribers interact with a key server by providing *credentials* to the key server. In turn they receive keys which fit the expressed capabilities in the credentials and thus can subsequently be used to decrypt and sign relevant messages in the content-based publish/subscribe system. In this case we say the credential is *authorized* by the key server.

Consequently, a credential consists of two parts: first a binary string which describes the capability of a peer in publishing and receiving events, and second a proof of its identity. The latter is used for authentication against the key server and verification whether the capabilities match the identity of the peer. While this can happen in a variety of ways, e.g. relying on challenge response, hardware support, etc., we pay attention mainly at expressing the capabilities of a credential, i.e. how subscribers and publishers can *create* a credential. This process needs to account for the many possibilities to partition the set of events expressed by an advertisement or subscription and exploit overlaps in subscriptions and publications. Subsequently, we use the term credential only for referring to the capability string of a credential.

The keys assigned to publishers and subscribers, and the ciphertexts are labelled with credentials. In particular, the identity-based encryption ensures, that a particular key can decrypt a particular ciphertext only if there is a match between the credentials of the ciphertext and the key. Publishers and subscribers maintain separate private keys for each authorized credential. Since we rely on the identity-based encryption paradigm, the key server in our system only maintains one master key pair and therefore also provides the same advantages in terms of achieving a good distributiveness as discussed in Section 2.5.

The public keys are generated by a string concatenation of an credential, an epoch for key revocation, a symbol $\in \{S, P\}$ distinguishing publishers from subscribers, and some additional parameters described in Section 5. The public keys can be easily generated by any peer without contacting the key server or other peers in the system. Similarly encryption of events and their verification using public keys do not require any interaction.

Due to the loose coupling between publishers and subscribers, a publisher does not know the set of relevant subscribers in the system. Therefore, a published event is encrypted with the public key of all possible credentials, which authorizes a subscriber to successfully decrypt the event. The ciphertexts of the encrypted event are then signed with the private key of the publisher as shown in Figure 2.

The overlay network is maintained according to the containment relationship between the subscriptions. Subscribers with coarser subscriptions are placed near the root and forward events to subscribers with less coarser subscriptions. A subscriber attempts to decrypt the received ciphertexts using its private keys and verify the authenticity of the event. If successful, the ciphertexts of the event is forwarded to the parent as well as children. Otherwise, the ciphertexts are dropped.

To maintain such a topology each subscriber should know the subscription of its parent and child peers. When a new subscriber arrives, it sends the connection request along with its subscription to a random peer in the overlay network. The connection request is forwarded by possibly many peers in the overlay network before it reaches the right peer to connect. Each forwarding peer matches the subscription in the request with the subscription of its parent and child peers to decide the forwarding direction. Maintaining a relationship between subscriptions clearly contradicts subscription confidentiality. Therefore, we show the approach to ensure a weaker notion of subscription confidentiality in Section 6.

This section has introduced only the main steps in establishing security. The subsequent sections discuss i) how to create credentials systematically to support scalability , ii) how keys are generated and how publications are encrypted, decrypted and verified, iii) how the publish/subscribe overlay is maintained and a weak notion of subscription confidentiality is preserved.

4. CREATION OF CREDENTIALS

For the description of the mechanisms behind the creation of credentials, we will first propose a systematic way of decomposing the event space for a content-based subscription model. Later we show how subscriptions and advertisements are mapped to the subspaces of the event space and credentials are created. Further extensions like considering string attributes and complex subscriptions are discussed subsequently.

4.1 Event Space decomposition for numeric attributes

At first we introduce a systematic decomposition of the event space by focusing on numeric attributes. Later we will discuss also how other types of attributes can be supported.

The event space, composed of d distinct numeric attributes can be geometrically modelled as a d-dimensional space such that each attribute represents a dimension in the space. Subscriptions and advertisements are represented by hyperrectangles in the space, whereas published events represent points.

With the spatial indexing approach, the event space is hierarchically decomposed into regular subspaces, which serve as enclosing approximation for the subscriptions and advertisements. The decomposition procedure divides the domain of one dimension after the other and recursively starts over in the created subspaces. Figure 3 visualizes the advancing decomposition with the aid of a binary tree. Each tree level represents one step of the recursive process, starting with the root where the event space is still undivided. The first partition of the event space is created by dividing along the first dimension and creates two sub-spaces. In one subspace, the domain of the first dimension is $[L_1, X_1]$ and in the other sub-space, it is $(X_1, U_1]$. The break line X of a domain always equals $\frac{L+U}{2}$ so that the sub-spaces cover equally sized domains of the divided dimension. After j steps, the event space is divided into 2^j sub-spaces, which equals the number of nodes at level j in the binary tree. The number of leaf nodes in the binary tree is equal to $\prod_{i=1}^{d} T_i$, where $T_i = \frac{U_i - L_i}{Granularity(i)}$ and $Granularity(i)$ defines the smallest addressable value of the attribute A_i.

Sub-spaces can be identified by *dz-expressions*. A dz-expression is a bit string of "0"s and "1"s. A subspace represented by dz-expression dz_1 is covered by the sub-space represented by dz_2, iff dz_2 is a prefix of dz_1.

4.2 Mapping to credentials

Subscription or advertisement of a peer can be composed of several subspaces. A credential is assigned for each of the mapped subspace. For instance, in Figure 3, f_2 is mapped to two subspaces and therefore posses two credentials $\{000, 010\}$.

An event can be approximated by the smallest (finest granularity) subspace that encloses the point represented by it. To deliver the encrypted event a ciphertext must be generated for each subspace that encloses the event so that the peer whose subscription mapped to any of these subspaces should be able to successfully decrypt the event. An event dz_e matches (is enclosed in) a subspace dz_s if dz_e is covered by dz_s. In general, the number of subspaces matched by an event dz_e is in the order of $log_2(\prod_{i=1}^{d} T_i)$ and is equal to $|dz_e| + 1$. For example, an event 0010 is matched by the five subspaces $0010, 001, 00, 0$ and ϵ.

For an event space with a large set of numeric attributes, the number of mapped subspaces and therefore credential for a subscription can be very large. This effects the scala-

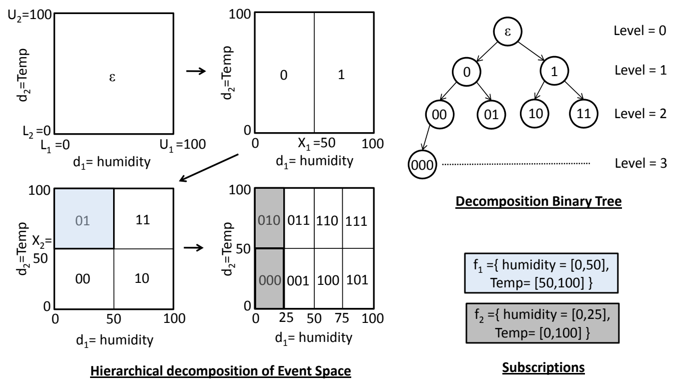

Figure 3: Numeric Attribute

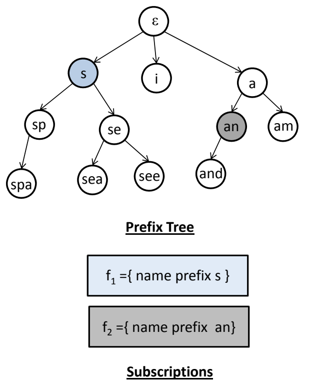

Figure 4: Prefix matching

bility of the system in terms of space required to store the keys, computational cost at the key server and communication cost between key server and the peers. There are three possibilities to mitigate this problem. The first possibility is to allow a coarsening of individual subscriptions by mapping to a smaller set of bigger subspaces. For example, f_2 in Figure 3 can be coarsened by mapping it to the sub-space 0. The coarsened subspace encloses more events and therefore a peer can decrypt more events than authorized by the original subscription. The second possibility is to reduce the granularity of each attribute but it effects the expressiveness of all the subscriptions. For example, if the finest addressable value of a numeric attribute is 8 then range such as $[1, 5]$ is not possible.

A third and better possibility to reduce the number of credentials is to decompose the domain of each attribute into subspaces separately. The spatial indexing procedure is the same as above, however, in this case a separate decomposition tree is built for each attribute. Each peer receives credentials separately for each attribute in its subscription. The number of credentials maintained for each subscription or advertisement are bounded by $\sum_{i=1}^{d} log_2(T_i)$. Similarly, the number of subspaces matched by an event are $\sum_{i=1}^{d} log_2(T_i)$. In our system, we used the third approach.

4.3 String attributes

The above spatial indexing technique can work with any ordered data type with a known domain. String attributes usually have a maximum number of characters. This allows them to have known bounds. They can be linearised by hashing or other linearisation mechanisms and thus can also be indexed [13].

Credential for more expressive string operations such as prefix matching can be generated using a *trie*. A trie is an ordered data structure for storing strings in a way that allows fast string lookup and prefix matching. Each node in the trie is labelled with a string, which serves as a common prefix to all its descendants as shown in Figure 4. Each peer is assigned a single credential, which is the same as its subscription or advertisement. Events correspond to the leaf nodes of the trie. To deliver an encrypted event a ciphertext must be generated with the label of each node in the path from the leaf to the root of the trie, so that a peer whose subscription matches any of the labels should be able to successfully decrypt the event. In general the number of nodes on the path from the leaf to the root of the trie are equal to L, where L is the length of the label assigned to leaf node. Similar mechanism can be used to generate credentials for suffix matching.

4.4 Complex subscriptions

For a complex subscription with predicates on different attributes, a subscriber receives separate credentials and thus keys for each attribute. Using these keys, a subscriber should be able to successfully decrypt any event with the corresponding attributes, if he is authorized to read the values associated with the attributes. Any cryptographic primitive can be easily used for this purpose. For example, similar to PSGuard [22], the keys for individual attributes can be XORed together and hashed to get the actual key for decrypting the event.

In a content-based publish/subscribe system, a subscription defines a conjunction on predicates. An event matches a subscription if and only if all of the predicates in the subscription are satisfied. To ensure event confidentiality, a subscriber must not be able to successfully decrypt any event which matches only parts of its subscriptions. However, assigning keys for individual attributes and XOR based decryption does not prevent this behaviour. For example, consider a subscriber with two subscriptions $f_1 = \{Area = [10, 20] \wedge location = Stuttgart\}$ and $f_2 = \{Area = [40, 80] \wedge location = London\}$. If the credentials and therefore keys are assigned for individual attributes then the subscriber can also decrypt the events matching the subscriptions $f_3 = \{Area = [10, 20] \wedge location = London\}$ and $f_4 = \{Area = [40, 80] \wedge location = Stuttgart\}$. Although he is not authorized to read events matching subscriptions f_3 and f_4. To properly ensure event confidentiality, all the keys associated with a subscription should be bound together, so that keys associated with different subscriptions should not be combined together.

5. METHODS FOR SECURITY

In this section we will describe the construction of security mechanisms to achieve authentication of publishers and subscribers as well as confidentiality of events.

One naive solution would be to directly use the techniques from PKI by assigning public/private key pair to each credential. Publishers and subscribers can contact key server to obtain the public/private key pairs that corresponds to their credentials. However, PKI does not provide a mechanism to bound together the public/private key pairs associated with the same subscription (cf. Section 4.4) and therefore, cannot be used.

The security mechanisms described in this section are adapted from attribute-based encryption schemes [2, 10]. In particular, our modifications, i) allow publishers to sign and encrypt events at the same time by using the idea of identity-based signcryption [25], ii) include some additional ciphertexts that increase the efficiency of the system and, iii) allow subscribers to verify the signatures associated with all the attributes simultaneously. Our modifications do not change the basic structure of the schemes and therefore preserves the same security strength.

5.1 Security parameters and initialization

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ denote the bilinear groups of prime order p i.e. $|\mathbb{G}_1| = |\mathbb{G}_2| = p$, $e : \mathbb{G}_1 \text{x } \mathbb{G}_1 \rightarrow \mathbb{G}_2$ denote an admissible bilinear map, g denote a generator in $\mathbb{G}_1$, and $H_1 : \{0,1\}^* \rightarrow \{0,1\}^{n_u}$, $H_2 : \{0,1\}^* \rightarrow \{0,1\}^{n_m}$ denote collusion resistant cryptographic hash functions.

The initialization algorithm i) chooses $\alpha \in \mathbb{Z}_p$, ii) computes $g_1 = g^\alpha$, iii) chooses $g_2, u', m' \in \mathbb{G}_1$, iv) selects vectors $\vec{u} = (u_i), \vec{m} = (m_i)$ of length n_u and n_m with every element chosen uniformly at random from $\mathbb{G}_1$. The *Master Public Key* is composed of $(e, g, g_1, g_2, u', m', \vec{u}, \vec{m})$. This master public key is known to every peer in the system and is used for encryption (cf. Section 5.4, Encryption) and signature verification (cf. Section 5.5, Verification). The *Master Private key* is g_2^α, and is only known to the key server.

5.2 Key generation for publishers

Before starting to publish events, a publisher contacts the key server along with the credentials for each attribute in its advertisement. If the publisher is allowed to publish events according to its credentials, the key server will generate separate private keys for each credential. Let $Cred_{i,j}$ denote the credential with label j for the attribute A_i, e.g $Cred_{location,00}$ denotes credential 00 of attribute *location*.

Public key: The public key of a publisher p for credential $Cred_{i,j}$ is generated as

$$Pu^p_{i,j} := (Cred_{i,j} \;||\; A_i \;||\; P \;||\; Epoch).$$

Private keys: The key server will generate the corresponding private keys as follows:

For each credential $Cred_{i,j}$ and a publisher p, let $v_p = H_1(Pu^p_{i,j})$ be a bit string of length n_u and let $v_p[k]$ denote the kth bit. Let $\Gamma_{i,j} \subseteq \{1, 2, ..., n_u\}$ be the set of all k for which $v_p[k] = 1$. Choose $r_{i,j} \in \mathbb{Z}_p$ at random and compute:

$$Pr^p_{i,j} := (g_2^\alpha (u' \prod_{k \in \Gamma_{i,j}} u_k)^{r_{i,j}}, g^{r_{i,j}}) =: (Pr^p_{i,j}[1], Pr^p_{i,j}[2])$$

5.3 Key generation for subscribers

Similarly, to receive events matching its subscription, a subscriber should contact the key server and receives the private keys for the credentials associated with each attribute A_i.

Public key: In case of subscribers, the public key for a credential $Cred_{i,j}$ is given as

$$Pu^s_{i,j} := (Cred_{i,j} \;||\; A_i \;||\; S \;||\; Epoch).$$

A different symbol S is used to differentiate the keys used for the verification of valid events from the ones used to provide event confidentiality.

Private keys: The private keys are generated as follows: The key server chooses $r_s \in \mathbb{Z}_p$ at random. The same r_s is used for all credentials associated with a subscription. For each credential $Cred_{i,j}$ it calculates $\Gamma_{i,j}$ similar to the publisher's case (cf. Section 5.2). Chooses $r_{i,j} \in \mathbb{Z}_p$ and computes:

$$Pr^s_{i,j} := (g_2^{r_s} (u' \prod_{k \in \Gamma_{i,j}} u_k)^{r_{i,j}}, g^{r_{i,j}}) =: (Pr^s_{i,j}[1], Pr^s_{i,j}[2])$$

Furthermore, a credential independent key $Pr^s[3] = g_2^{r_s + \alpha}$ is generated. Later we will see that r_s along with $Pr^s[3]$ is needed to bind the keys of a subscription together.

5.4 Publishing events

Encryption. When a publisher wants to publish an event M, it chooses $q_i \in \mathbb{Z}_p$ at random for each attribute A_i of the event, such that $q = \sum_{i=1}^{d} q_i$. These random values ensure that only the subscribers who have matching credentials for each of the attributes should be able to decrypt the event. Furthermore, it generates a fixed length random key SK.

To encrypt an event, a publisher uses master public key and performs the following steps:

Step1: Compute:

$$C_1 = e(g_1, g_2)^q SK \;\; and \;\; C_2 = AES(M, Pu^p_{i,j})^{SK}$$

The cost of asymmetric encryption generally increases with the size of the plaintext. Therefore, only a fixed length random key SK is encrypted using the private keys of publisher. The actual event message M is encrypted with a symmetric encryption algorithm such as AES, using the key SK.

The ciphertext C_2 also includes the public keys of the credentials which authorizes the publisher to send the event. The inclusion of these public keys increases the efficiency of signature verification process at the expense of a small increase in the ciphertext size (one public key per attribute).

Additionally, a subscriber does not know about the credentials with which the ciphertext is encrypted and cannot tell in advance whether he is authorized to read the message. Therefore, to enable the subscribers to detect the successful decryption of events, either the event should be appended with a predefined number of zeros ($M||0^*$) or a hash of the event ($H_2(M)$) should be included in the ciphertext.

Step2: For each attribute A_i, compute $C_i = g^{q_i}$ and $C_i' = e(g_1, g_2)^{q_i}(A_i||0^*)$, where 0^* represents a predefined length of zeros. The ciphertext C_i' is used for the routing of encrypted events as discussed in section 7.

Step3: For each attribute, a ciphertext should be send for each credential that matches its value. For example, in case of a numeric attribute with value mapped to 0000, a ciphertext should be disseminated for the credentials 0000,000,00 and 0.

For each credential $Cred_{i,j}$ that matches the value of the attribute A_i, compute $C_{i,j} = (u' \prod_{k \in \Gamma_{i,j}} u_k)^{q_i}$, where $\Gamma_{i,j}$ is calculated as described above.

The ciphertexts are ordered according to the containment relationship (in descending order) between their associated credentials, e.g. for the above example the order is [$C_{i,0}$, $C_{i,00}$, $C_{i,000}$, $C_{i,0000}$].

Table 1: Cost of security mechanisms

	Public parameters	Private keys	Ciphertext Size	Encryption cost	Decryption cost	Sign cost	Verification cost
Numeric	$O(1)$	$O(\sum_{i=1}^{d} log_2\ T_i)$	$O(\sum_{i=1}^{d} log_2\ T_i)$	$O(\sum_{i=1}^{d}\ log_2\ T_i)$	$O(d)$	$O(d)$	$O(d)$
Prefix	$O(1)$	$O(d.L)$	$O(d.L)$	$O(d.L)$	$O(d)$	$O(d)$	$O(d)$

Signature. Finally, the publisher signs the ciphertexts using its private keys. It computes $v_m = H_2(M)$ a bit string of length n_m. Let $v_m[k]$ denote the kth bit and $\Gamma_m \subseteq \{1, 2, ..., n_m\}$ be the set of all k for which $v_m[k] = 1$. For each attribute, the credential $Cred_{i,j}$ which authorizes the publisher to send the corresponding attribute value, we compute:

$$C_{i,j}^{sign}[1] := Pr_{i,j}^{p}[1](m' \prod_{k \in \Gamma_m} m_k)^{q_i} \ and\ C_{i,j}^{sign}[2] := Pr_{i,j}^{p}[2]$$

The credentials $Cred_{i,j}$ are same to those included in C_2.

5.5 Receiving events

Decryption. On receiving the ciphertexts, a subscriber tries to decrypt them using its private keys. The ciphertexts for each attribute are strictly ordered according to the containment relation between their associated credentials, therefore a subscriber only tries to decrypt the ciphertext whose position coincides with the position of its credential in the containment hierarchy of corresponding attribute. The position of a credential can be easily determined by calculating its length. For example, for a numeric attribute, credential 0000 occupies $4th$ position in the containment hierarchy i.e. after 0,00 and 000. Subscribers decrypt the ciphertext in the following manner.

Step1: Compute $\forall A_i \in \Omega : D_{i,t}$, where t is the credential assigned to the subscriber for the attribute A_i.[1]

$$D_{i,t} := \frac{e(Pr_{i,t}^{s}[1], C_i)}{e(Pr_{i,t}^{s}[2], C_{i,t})} = e(g_2, g)^{r_s \cdot q_i}$$

Step2: Compute $D' := \prod_{i=1}^{d} D_{i,t} = e(g_2, g)^{r_s \cdot q}$

Step3: Remove r_s from D' to recover the actual message.

$$D'' := \frac{e(\prod_{i=1}^{d} g^{q_i}, Pr^s[3])}{D'} = e(g_1, g_2)^q$$

Each subscription requires a different random value r_s. If keys from different subscriptions are combined together D' cannot be computed correctly in Step 2 and thus r_s cannot be cancelled out in step 3, preventing the decryption of the event.

Step4: Since $C = e(g_1, g_2)^q SK$, a simple division will give the key SK i.e. $\frac{e(g_1,g_2)^q SK}{e(g_1,g_2)^q} \rightarrow SK$. The key SK is then used to decrypt the actual event message M. The successful decryption of message is detected by looking for predefined number of zeros appending the message or comparing the message hashes.

Verification. A subscriber will only accepts the message if it is from an authorized publisher. To check the authenticity of an event, subscribers use the master public key and performs the following steps:

Step1: Compute: $V_L := e(\prod_{i=1}^{d} C_{i,j}^{sign}[1], g)$, where $\prod_{i=1}^{d} C_{i,j}^{sign}[1]$ represents the product of all received $C_{i,j}^{sign}[1]$ ciphertexts.

Step2: Compute: $V_{R1} := \prod_{i=1}^{d} e(g_1, g_2)$

Step3: Compute:
$V_{R2} := e(\prod(u' \prod_{k \in \Gamma_{i,j}} u_k), \prod_{i=1}^{d} C_{i,j}^{sign}[2])$,
where $\prod(u' \prod_{k \in \Gamma_{i,j}} u_k)$ represents the product of all $Pu_{i,j}^{p}$ in C_2 and $\prod_{i=1}^{d} C_{i,j}^{sign}[2]$ is the product of all received $C_{i,j}^{sign}[2]$ ciphertexts.

Step4: Compute: $V_{R3} := e(m' \prod_{k \in \Gamma_m} m_k, \prod_{i=1}^{d} C_i)$.

The received event is authentic if the following identity holds: $V_L = V_{R1} \times V_{R2} \times V_{R3}$

Remember the ciphertext C_2 contains the public keys of the credentials which authorize the publisher to send the event. If no such public keys are included in C_2, then the subscriber should try to authenticate the event by checking for all possible credentials which a publisher might hold to publish the event. For example, an event with a single numeric attribute and a value mapped to 0000 can be published by the publisher with credentials 0000,000,00 or 0. In this case our approach is as follows: a subscriber checks the authenticity of the event for each attribute A_i separately[2], by verifying that for one of the possible credentials $Cred_{i,j}$ the following identity holds:

$$e(C_{i,j}^{sign}[1], g) = \\ e(g_1, g_2)\ e(u' \prod_{k \in \Gamma_{i,j}} u_k, C_{i,j}^{sign}[2])\ e(m' \prod_{k \in \Gamma_m} m_k, C_i)$$

In this case the total verification cost is $\sum_{i=1}^{d} log_2(T_i)$.

Table 1 shows the worst case costs of our security mechanisms for numeric and string attributes.

5.6 Rekeying

When a subscriber arrives or leaves the system, the keys of all the subscribers with the corresponding credentials should be changed in order to provide forward and backward secrecy. Changing the keys for each incoming or outgoing subscriber impacts the scalability of the systems in terms of communication overhead and load on the key server. Therefore, our approach is to use periodic rekeying by dividing the system time into epochs.

The length of the epoch is directly related to the overhead incurred due to rekeying. Big length epochs are used because they require less frequent rekeying, however, they can only provide coarse grain access control. Our approach enables fine grain access control within big length epochs by the ad-

[1]A subscriber might have many credential for a single attribute, e.g. $log_2(T_i)$ in the worst case for a numeric attribute. Our overlay topology maintenance and event dissemination mechanisms (Section 6) ensures that subscriber knows the exact credential needed to decrypt the event.

[2]Combined verification, as mentioned above, in this case will result in $(log_2(T_i))^d$ and thus is not feasible.

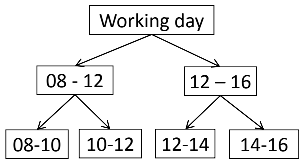

Figure 5: Fine grain key management within epoch

dition of a time attribute in the system. The time attribute specifies credentials of peers to receive or send the events at finer granularity within each epoch. For example, in Figure 5 the epoch length of one working day is hierarchically divided into smaller times slots. Only the peers with the credentials of a time slot can receive or send the event in that time slot. The time attribute is treated like any other attribute in the system. Each encrypted event contains the ciphertexts for each credential which authorizes a subscriber to receive events in the current time slot. Similarly, for successful decryption of an event, a subscriber should use the private key that corresponds to required time credential. The construction of our security mechanisms guarantee that events cannot be retrieved without having required time keys. Furthermore, the time keys from different subscriptions cannot be combined to retrieve unauthorized events.

Of course, the flexibility of having fine grain access comes at the cost of managing more keys and communicating more ciphertexts. However, the overhead is small compared to the granularity of access, e.g. access control at the granularity of a second for an epoch of one month, requires a subscriber or a publisher to maintain 22 decryption keys in the worst case and each encrypted event contains 24 more ciphertexts.

Each subscriber or publisher contacts the key server at the end of the epoch and receives the private keys for the new epoch. It is interesting to note that in case of smart cards, private keys are generated locally, without incurring any communication overhead. Otherwise, the overhead for regenerating private keys and securely communicating them to respective publishers or subscribers will be shared between the replicas of key server.

6. SUBSCRIPTION CONFIDENTIALITY

In this section, we address subscription confidentiality in a broker-less publish/subscribe system, where publishers and subscribers are responsible for maintaining the overlay network and forwarding events to relevant subscribers. First, we describe the maintenance of the publish/subscribe overlay network. Later we define a weaker notion of subscription confidentiality and detail the mechanisms behind.

6.1 Publish/Subscribe Overlay

The publish/subscriber overlay is a virtual forest of logical trees, where each tree is associated with an attribute (cf. Figure 6). A subscriber joins the trees corresponding to the attributes of its subscription. Similarly, a publisher sends an event on all the trees associated with the attributes in the event.

Within each attribute tree, subscribers are connected according to the containment relationship between their credentials associated with the attribute. The subscribers with coarser credentials (e.g. the ones mapped to coarser subspaces in case of numeric attributes) are placed near the root of the tree and forward events to subscribers with finer credentials. A subscriber with more than one credentials can be handled by running multiple virtual peers on a single physical node, each virtual peer maintaining its own set of tree links. For example in Figure 6, the subscriber s_3 has two credentials $\{000, 010\}$ and is connect to two places in the tree.

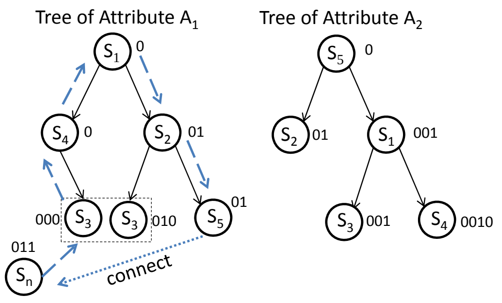

Figure 6: Publish/Subscriber system with two numeric attributes

In order to connect to an attribute tree, a newly arriving subscriber s_n sends the connection request along with its credential to a random peer s_r in the tree. The peer s_r compares the request credential with its own; if the peer's credential covers the request credential and the peer can accommodate more children, it accepts the connection. Otherwise, the connection request is forwarded to all the children with covering credentials and the parent peer with exception of the peer, from which it was received. In this way the connection request is forwarded by many peers in the tree before it reaches the suitable peer with covering credential and available connection. Figure 6 shows the path followed by a request from a subscriber s_n until it reaches the desired parent subscriber.

The drawback of maintaining separate trees for each attribute is that the subscribers also receive events that match only a part of their subscription (false positives). However, it cannot effect event confidentiality because false positives cannot be decrypted without having required credentials for each attribute.

6.2 Weak subscription confidentiality

It is infeasible to provide strong subscription confidentiality in a broker-less publish/subscribe system because the maintenance of the overlay topology requires each peer to know the subscription of its parent as well as its children. To address this issue, a weaker notion of subscription confidentiality is required.

Definition 6.1. *Let s_1 and s_2 denote two subscribers in a publish/subscribe system which both possess credentials for an attribute A_i.* Weak subscription confidentiality *ensures that at most the following information can be inferred about the credentials of the subscribers:*

1. *The credential of s_1 is either coarser or equal to the credentials of s_2.*
2. *The credential of s_1 is either finer or equal to the credentials of s_2.*
3. *The credential of s_1 and s_2 are not in any containment relationship.*

Definition 6.1. is consistent with the subscription security model used for broker-based publish/subscribe systems in the literature, Content-based approaches rely on the containment relationship between subscriptions, which mandates that even a broker should know if two subscriptions are related [17].

6.3 Secure connection protocol

In the following, we propose a secure connection protocol, which maintains the desired overlay topology without violating the weak subscription confidentiality. For simplicity and without loss of generality, here we discuss the secure connection protocol with respect to a single tree associated with a numeric attribute A_i and each of the subscribers owns a single credential.

The secure protocol is based on the idea that in the tree subscribers are always connected according to the containment relationship between their credentials, e.g. a subscriber with credential 00 can only connect to the subscribers with credentials 0 or 00.

A new subscriber s encrypts *secret words*[3] with the public keys $Pu^s_{i,j}$ for all credentials that cover its own credential e.g. a subscriber with credential 00 will generate ciphertexts by applying the public keys $Pu^s_{i,0}$ and $Pu^s_{i,00}$. The generated ciphertexts are added to a *connection request* (CR) and the request is forwarded to a random peer in the tree. A connection is established if the peer can decrypt any of the ciphertexts using its private keys.

Filling the security gaps: By looking at the number of ciphertexts in the connection request the peer could detect the credential of the requesting subscriber s. For example, a subscriber with credential 00 can only connect to 0 or 00 and therefore, a connection request will have two ciphertexts, whereas the the connection request for 000 will have three ciphertexts. In the worst case, a subscriber has a credential of the finest granularity. This can be covered by $log_2(T_i)$ other credentials and therefore a connection request contains in the worst case that many ciphertexts. To avoid any information leak, ciphertexts in the connection request are always kept in $O(log_2\ T_i)$ ($O(L)$ for prefix matching) by adding random ciphertexts if needed. Furthermore, the ciphertexts are shuffled to avoid any information leak from their order.

A different *secret word* is used for the generation of each ciphertext to avoid any information leak to the peer which has successfully decrypted one of the ciphertexts and thus has recovered the secret word. Otherwise, the peer can try to generate ciphertexts by encrypting the secret word with public keys for $O(log_2\ T_i)$ credentials and can easily determine the random ciphertexts in the connection request and thus the credentials of the requesting subscriber s. Finally, to avoid an attacker to generate arbitrary connection request messages and try to discover the credential of other peers in the system, the connection request is signed by the key server. This step needs to be performed only once, when a newly arriving subscriber authorizes itself to the key server in order to receive private keys for its credentials.

Overall algorithm: The secure connection protocol is shown in Algorithm 1. In the algorithm, the procedure *decrypt_request* tries to decrypt one of the ciphertexts in the connection request message.

[3]A secret word can be a sequence of random bits

Algorithm 1 Secure connection protocol at peer s_q

```
1:  upon event Receive(CR of $s_{new}$ from $s_p$) do
2:    if $decrypt\_request(CR) == SUCCESS$ then
3:      if degree($s_q$) == available then // can have child peers
4:        connect to the $s_{new}$
5:      else
6:        forward CR to $\{child\ peers\ and\ parent\} - s_p$
7:    if $decrypt\_request(CR) == FAIL$ then
8:      if $s_p$ == parent then
9:        Try to swap by sending its own CR to the $s_{new}$.
10:     else
11:       forward to parent
```

A child peer s_q receives CR (of subscriber s_{new}) from the parent s_p only if the parent cannot accommodate more children. If s_q cannot be the parent of s_{new}, i.e., s_{new}'s credential is coarser than that of s_q, then it tries to swap its position with s_{new} by sending its own connection request (cf. *Algorithm* 1, *lines 7-9*). However, if none of the children of parent s_p can connect or swap with s_{new}, then there is no containment relationship between the credentials of the children and s_{new}. In this case a parent should disconnect one of its children in order to ensure the new subscriber is connected to the tree.

6.4 Discussion

For an attribute A_i, let $\mathbb{S}_{\preceq}$ be the set of peers in the system whose credentials covers the credential of the subscriber s_1. Let $\mathbb{S}_{\succeq}$ denote the set of subscribers whose credentials are covered by the credential of the subscriber s_1 and $\mathbb{S}_{\npreceq}$ denote the set of subscribers whose credentials have *no* containment relation with the credential of the subscriber s_1. The secure connection protocol for the maintenance of the tree associated with the attribute A_i leaks the following information:

1. Any peer $s^{'} \in \mathbb{S}_{\preceq}$ can infer that the credential of s_1 is either finer or equal to its own credential.

2. Any peer $s^{''} \in \mathbb{S}_{\succeq}$ can infer that the credential of s_1 is either coarser or equal to its own credential.

3. Any peer $s^{'''} \in \mathbb{S}_{\npreceq}$ can infer that credential of s_1 are not in any containment relationship with its own credential.

In general, confidentiality decreases with the decrease in granularity of the credential of $s^{'}$ i.e. $s^{'}$ with the credential of finest granularity can determine the subscription of peer s_1 with certainty. Similarly, confidentiality decreases with the increase in granularity of the credential of $s^{''}$ i.e. $s^{''}$ with the most coarse credential can exactly determine the subscription of s_1. For example, if $s^{''}$ has subscribed to the whole domain of the attribute and the secure connection protocol allows it to connect to s_1, then s_1 has also subscribed for the whole domain.

7. SECURE EVENT DISSEMINATION

The secure connection protocol ensures that the credential of a parent peer covers the credentials of its children. Therefore, a parent peer can decrypt every event, which it forwards to the children. Regardless of the cryptographic primitives, a parent can eventually discover the credentials of its child peers e.g. by maintaining histories. In our approach we used one hop flooding to avoid this problem.

In one hop flooding, a parent assumes that the children have the same credentials as its own and forwards each successfully decrypted event to all of them. In turn the children forward each event which was successfully decrypted to all of their children and so on. In this strategy, a child may have finer credentials then its parent and may receives false positives.

The detailed mechanism works as follows: To publish an event, a publisher forwards the ciphertexts of each attribute to a randomly selected subscriber on the corresponding attribute tree. All the ciphertexts of an event are labelled with a unique value such as sequence number of the event. This helps subscribers to identify all the ciphertexts of an event(as ciphertexts for each attribute are received on the separate tree).

On receiving a ciphertext associated with attribute A_i, the subscriber checks whether it can recover the $A_i||0^*$ from $C_i^{'}$ by using the private key of the credential $Cred_{i,t}$ associated with the connection from which the ciphertexts are received.The following steps are performed:

Step1: Compute $D_{i,t} = \frac{e(Pr_{i,t}^{s}[1],C_i)}{e(Pr_{i,t}^{s}[2],C_{i,t})} = e(g_2, g)^{r_s \cdot q_i}$

Step2: Remove r_s: $D^{''} = \frac{e(g^{q_i}, Pr^{s}[3])}{D_{i,t}} = e(g_1, g_2)^{q_i}$

Step3: Recover plaintext: $\frac{C_i^{'}}{e(g_1,g_2)^{q_i}} \rightarrow A_i||0^*$

The successful recovery of $A_i||0^*$ means that the credential of the subscriber matches one of the credentials in the ciphertext and therefore, the ciphertexts should be forwarded to all children and the parent. However, in case of failure, the ciphertexts are only forwarded to the parent (unless the event is received from the parent).

Once the ciphertexts of the all the attributes in the event are received, the subscriber will try to decrypt the whole event and verify its authenticity.

8. EVALUATIONS

Similar to EventGuard [21], we evaluated our solution in two aspects: i) quantifying the overhead of our cryptographic primitives, and ii) evaluating the performance of our secure publish/subscribe system by benchmarking it with an unsecured system.

8.1 Performance of cryptographic primitives

In this section, we measure the computational overhead of our security mechanisms. The security mechanisms are implemented by Pairing-Based Cryptography (PBC) library [12]. The implementation uses a 160-bit elliptic curve group based on the supersingular curve $y^2 = x^3 + x$ over a 512-bit finite field. All of our measurements were made on a 2.00 GHz Intel Centrino Duo with 2GB RAM, running Ubuntu 9. Table 2 shows the amount of time needed for cryptographic primitives to perform encryption, decryption, signature and verification. All reporting values are averaged over 1000 measurements. The message size is kept 128 bytes as this key length is good enough for most symmetric encryption algorithms.[4] Table 3 shows the overhead from the perspective of publishers and subscribers in our system. In general, the cost of verification is high due to the fact that it involves the computationally expensive pairing operations.

[4]In our system pairing based encryption is used to encrypt a random key SK, which is later used to decrypt actual event using symmetric encryption.

Table 2: Throughput of cryptographic primitives

Encryption(E)	10KB/s
Decryption(D)	10KB/s
Signature(S)	158 sign/s
Verification(V)	52 verify/s

Table 3: Computation times (in msec) from the perspective of publishers and subscribers

$Time_E$	$6.86174 + d * 5.42426$
$Time_S$	$d * 6.31925$
$Time_D$	$6.1522 + d * 6.10192$
$Time_V$	$19.30517 + d * 0.001$

8.2 Performance of publish/subscribe system

We evaluated the performance of our system according to the following criteria: i) overlay construction time, ii) event dissemination latencies and iv) resilience of the system to the event flooding based DoS attacks.

Experimental Setup: Simulations are performed using PeerSim [11]. Simulations are performed for upto $n = 2,048$ peers. The out-degree constraints of the peers are always chosen as $log(n)$. The latencies for the links between the peers varies from 24ms to 134ms.

The event space has upto $d = 16$ different attributes. The data type of each attribute is Integer, and the domain of each attribute is the range $[1, 16]$. We evaluated the system performance under uniform subscription and advertisement workloads; and with a uniform event distribution.

Publish/subscribe overlay construction. We measured the average latency experienced by each subscriber to connect to a suitable position in an attribute tree. Latency is measured from the time subscriber sends connection request message to a random peer in the tree till the time the connection is actually established. The evaluations are performed only for a single attribute tree. Figure 7(a) shows that the average connection latency increases with the number of peers in the system because of the increase in the height of the attribute tree (each new hop increases the network latency as well as time to apply security mechanisms). However the corresponding increase in latency is small due to the fact that the overall out-degree also increases with the number of peers, resulting in only a small increase in the height of tree. Furthermore, Figure 7(a) shows that there is an almost constant overhead of approximately 250-300 ms due to security mechanisms. Our evaluations with higher number of attributes indicate that the average connection latency experienced by a subscriber is independent to the number of attributes. The reason being that each attribute tree is created in parallel and a subscriber sends connection request to connect multiple attribute trees at the same time.

Event dissemination. We measured the average time needed by the event to be disseminated to all the relevant

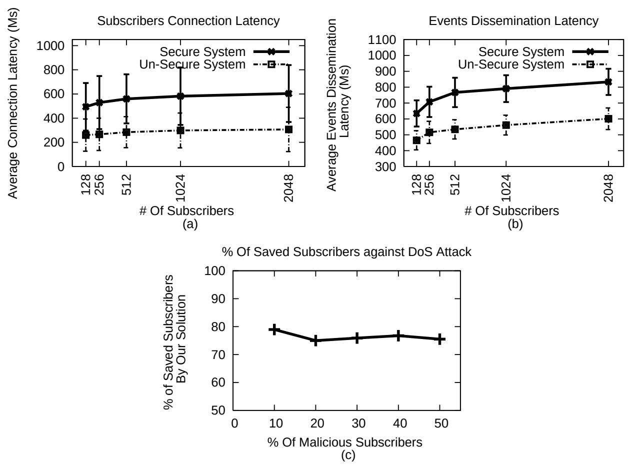

Figure 7: Performance of publish subscribe system

subscribers in the system. For each subscriber the time is measured from the dissemination of the event by the publisher till it is successfully decrypted and verified by the subscriber. For the experiment, 160 publishers are introduced in the system and each published 10 events. Figure 7(b) shows that average time to disseminate an event increases with the number of peers in the system because of the increase in number of the relevant subscribers as well as the height of the dissemination tree. Similar to previous results there is an almost constant overhead of approximately 200-250 ms due to security mechanisms. Furthermore, the standard deviation is less than figure 7(a) because the number of subscribers who should receive each published event are approximately same due to uniform subscription workload.

Resilience against event flooding based DoS attacks.

In a flooding based DoS attack, a malicious peer encrypts the fake events using public keys of the credentials matching those events, but does not own the private keys to sign the fake events. The fake events are disseminated to randomly selected peers in the network. Our secure event dissemination mechanism can easily reduce the effects of these attacks by checking the authenticity of ciphertexts received on an attribute tree, before forwarding them to child peers or parent i.e. if a subscriber successfully recovers $A_i||0^m$ from $C_i^{'}$, it should verify the signatures of the publisher for the attribute A_i before forwarding the ciphertexts. We made small changes in the security mechanisms of section 5 to achieve the goal. In particular, we modified the generation of $C_{i,j}^{sign}$[1], such that $v_m = H_2(A_i||0^*)$ so that each subscriber who have credentials to recover $A_i||0^*$ can verify the signatures without any need to decrypt the whole event[5].

In case of unsecured system fake events are delivered to all relevant subscribers, whereas using our approach a fake event is detected by the first subscriber who is authorized to receive the events encrypted with corresponding credentials. In worst case a fake event (forwarded to a peer not authorized to receive the event) will go to the root of the attribute tree before being detected as fake. Figure 7(c) shows the percentage of subscribers saved from receiving fake events by our approach as compared to the unsecured system for different percentages of malicious subscribers in the system. It is evident from the figure that even in the presence of 50% malicious subscribers approximately 75% of subscribers are saved from receiving fake messages as compared to unsecured solution. Further experiments show that the percentage of save subscribers are not much dependent on the number of malicious peers in the system and stay approximately in the range of $80 - 70\%$

[5]Similarly, the effect of replay attacks can also be reduced by adding publisher signed time stamps.

9. RELATED WORK

Over the last decade many content-based publish/subscribe systems [6, 9, 1, 5, 23] have evolved. Most systems focus on increased scalability of the system by reducing the cost of subscription forwarding and event matching. Only a few systems have addressed security issues in a content-based publish/subscribe system. Wang et al. [24] investigate the security issues and requirements that arise in an internet-scale publish/subscribe system. They concluded that due to loose coupling between publishers and subscribers, many security issues cannot be directly solved by current technology and requires further research. Hermes [16] proposes a security service that uses role-based access control to authorize subscribers as well as to establish trust in the broker network. Pesonen et al. [15] addresses the role-based access control in multi-domain publish/subscribe systems by the use of a decentralized trust management. Opyrchal et al. [14] try to leverage concepts from secure group-based multicast techniques for the secure distribution of events in a publish/subscribe system. They showed that previous techniques for dynamic group key management fails in a publish/subscribe scenario, since every event potentially has a different set of interested subscribers. To overcome the problem they proposed a key caching technique. However, broker nodes are assumed to be completely trustworthy. Eventguard [21] provides six guards/component to protect each of the five major publish/subscribe operations (subscribe, unsubscribe, advertise, unadvertise, publish) and routing. It only supports topic-based routing through the direct use of pseudorandom functions. PSGuard [22] addresses scalable key management in a content-based system by using hierarchical key derivation to associate keys with subscriptions and events. It does not address the issues related to the secure routing of events and subscription confidentiality. Event confidentiality is not properly ensured in case of complex subscriptions, i.e., the keys associated with the filters in a complex subscription are not bind together.

Another drawback with the existing solutions is their assumption about the presence of a broker network [17, 16, 21]. These solutions are not directly applicable to peer-to-peer environments where subscribers are clustered according to their interests.

The recent progress of pairing-based cryptography motivates many applications built upon Identity-Base Encryption. Attribute-Based Encryption [2, 10], is a general form of Identity-Based Encryption. It allows for a new type of encrypted access control, where the access control policies are either embedded in the user private keys or in the ciphertexts. Shi et al. [20] and Boneh et al. [4] addresses complex queries (such as conjunction, subset and range queries) over encrypted data using identity based encryption. Both approaches address the problem from a pure cryptographic perspective and are not practical in our scenario. In the construction of Boneh et al. [4] the cost of public parameters, encryption cost and ciphertext size for range queries increases with the number of dimensions and number of points in each dimension i.e. $O(d.T_i)$. Similarly the decryption cost of Shi et al. [20] is exponential in the number of attributes $O((log_2(T_i)^d)$. Therefore, instead of using their cryptographic mechanisms, we derived our mechanisms directly from attribute-based encryption. Furthermore, these systems are not targeted toward content-based systems and do not address the issues related to verification of event authenticity, subscription confidentiality and secure event routing.

10. CONCLUSIONS

In this paper, we have presented a new approach to provide authentication and confidentiality in a broker-less content-based publish/subscribe system. The approach is highly scalable in terms of number of subscribers and publishers in the system and the number of keys maintained by them. In particular, we have developed mechanisms to assign credentials to publishers and subscribers according to their subscriptions and advertisements. Private keys assigned to publishers and subscribers, and the ciphertexts are labelled with credentials. We adapted techniques from identity based encryption, i) to ensure that a particular subscriber can decrypt an event only if there is match between the credentials associated with the event and its private keys and, ii) to allow subscribers to verify the authenticity of received events. Furthermore, we developed a protocol to preserve the weak subscription confidentiality in the presence of semantic clustering of subscribers.

11. ACKNOWLEDGMENTS

This work is partially funded by SpoVNet project [8] of Baden-Württemberg Stiftung gGmbH. We would like to thank Jianwei Yin, Marco Völz and anonymous reviewers for their helpful comments.

12. REFERENCES

[1] E. Anceaume, M. Gradinariu, A. K. Datta, G. Simon, and A. Virgillito. A semantic overlay for self-peer-to-peer publish/subscribe. In *ICDCS*, 2006.

[2] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Proc. of the IEEE Symposium on Security and Privacy*, 2007.

[3] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *Intl. Cryptology Conf. on Advances in Cryptology*, 2001.

[4] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *4th theory of Cryptography Conf.*, 2007.

[5] J. A. Briones, B. Koldehofe, and K. Rothermel. Spine : Adaptive publish/subscribe for wireless mesh networks. *Studia Informatika Universalis*, 7, 2009.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. on Compt Syst*, 2001.

[7] R. Chand and P. Felber. Semantic peer-to-peer overlays for publish/subscribe networks. In *Euro-Par*, pages 1194–1204, 2005.

[8] T. S. Consortium. Spontaneous virtual networks: On the road towards the internet's next generation. *it - Information Technology*, 2008.

[9] L. Fiege, M. Cilia, G. Muhl, and A. Buchmann. Publish-subscribe grows up: Support for management, visibility control, and heterogeneity. *IEEE Internet Computing*, 10:48–55, 2006.

[10] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *conf. on Computer and communications security*, ., 2006. .

[11] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. PeerSim: A Peer-to-Peer Simulator. http://peersim.sourceforge.net/.

[12] B. Lynn. The pairing-based cryptography (pbc) library. http://crypto.stanford.edu/pbc/.

[13] V. Muthusamy and H.-A. Jacobsen. Infrastructure-less content-based publish/subscribe. Technical report, Middleware Systems Research Group, University of Toronto, 2007.

[14] L. Opyrchal and A. Prakash. Secure distribution of events in content-based publish subscribe systems. In *conf. on USENIX Security Symposium*, 2001.

[15] L. I. W. Pesonen, D. M. Eyers, and J. Bacon. Encryption-enforced access control in dynamic multi-domain publish/subscribe networks. In *DEBS*, 2007.

[16] P. Pietzuch. *Hermes: A Scalable Event-Based Middleware*. PhD thesis, University of Cambridge, Feb 2004.

[17] C. Raiciu and D. S. Rosenblum. Enabling confidentiality in content-based publish/subscribe infrastructures. In *Intl. Conf. on Security and Privacy in Communication Networks*, 2006.

[18] V. Security. Identity-based encryption technology overview. In *64th IETF Meeting*, Nov. 2005.

[19] A. Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, 1985.

[20] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *Proc. of IEEE Symposium on Security and Privacy*, 2007.

[21] M. Srivatsa and L. Liu. Securing publish-subscribe overlay services with eventguard. In *Proc. of ACM conf. on Computer and communications security*, 2005.

[22] M. Srivatsa and L. Liu. Scalable access control in content-based publish-subscribe systems. Technical report, Georgia Institute of Technology, 2006d.

[23] M. A. Tariq, G. G. Koch, B. Koldehofe, I. Khan, and K. Rothermel. Dynamic publish/subscribe to meet subscriber-defined delay and bandwidth constraints. In *Intl. Conf. on Parallel Computing (Euro-Par)*, 2010.

[24] C. Wang, A. Carzaniga, D. Evans, and A. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *Proc.of Hawaii Intl. Conf. on System Sciences (HICSS)*, 2002.

[25] Y. Yu, B. Yang, Y. Sun, and S.-l. Zhu. Identity based signcryption scheme without random oracles. *Comput. Stand. Interfaces*, 31, 2009.