Access Control in Decentralised Publish/Subscribe Systems

Lauri I.W. Pesonen, David M. Eyers, and Jean Bacon
University of Cambridge, Computer Laboratory
JJ Thomson Avenue, Cambridge, CB3 0FD, UK
{firstname.lastname}@cl.cam.ac.uk

Abstract— **Publish/subscribe has emerged as an attractive communication paradigm for building Internet-wide distributed systems by decoupling message senders from receivers. Large scale publish/subscribe systems are likely to employ components of the event transport network owned by cooperating, but independent organisations. As the number of participants in the network increases, security becomes an increasing concern. So far most of the research on publish/subscribe has focused on efficient event routing, event filtering, and composite event detection. Very little research has been published regarding securing publish/subscribe systems. This paper extends our previous work to present and evaluate a secure multi-domain publish/subscribe infrastructure that supports and enforces fine-grained access control over the individual attributes of event types.**

I. INTRODUCTION

Publish/subscribe has become an increasingly popular communication paradigm for large-scale distributed systems. A publish/subscribe system decouples the publishers from the subscribers by introducing an event service between the communicating parties. This decoupling along with asynchronous messaging allows publish/subscribe systems to scale in both geographic distance as well as in the number of participating nodes. Modern highly scalable publish/subscribe systems implement the event service as a decentralised network of brokers that is used to deliver publications from a publisher to all interested subscribers.

Publish/subscribe systems have been advocated especially for large-scale systems where the event service covers a large geographic area. Such a publish/subscribe system would have to span multiple domains, be they independent administrative domains inside a single organisation, multiple independent organisations, or a combination of the two.

In the past most publish/subscribe oriented research has concentrated on routing algorithms, content-based filtering, and broker network topologies. Relatively little research has been done with respect to security in publish/subscribe systems. On the other hand, the fact that widely distributed publish/subscribe systems are likely to span multiple domains requires that security issues be addressed. In particular, large-scale publish/subscribe systems will need to include some form of access control.

Our overall research aim is to develop Internet-scale publish/subscribe networks that provide secure, efficient delivery of events, fault-tolerance and self-healing in the delivery infrastructure and a convenient event interface.

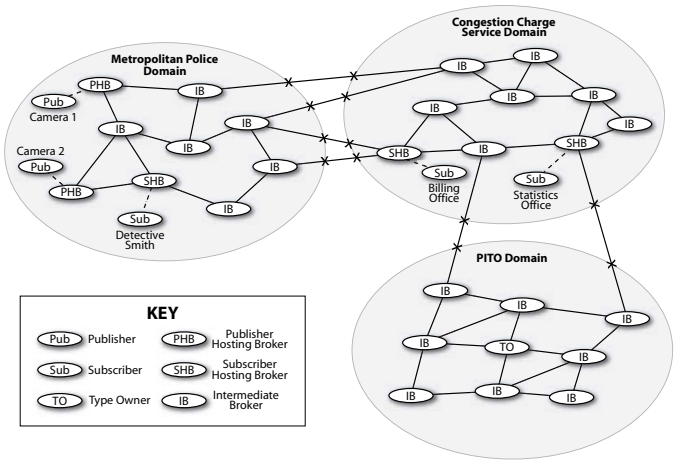

Figure 1. An overall view of our multi-domain publish/subscribe deployment.

In this paper we propose a capability-based, decentralised access control architecture for multi-domain publish/subscribe systems. The architecture provides a mechanism for authorising clients to publish and subscribe to specific event types or topics. The client's privileges are checked by the local broker that the client connects to in order to access the publish/subscribe system.

One possible approach is to implement access control at the edge of the broker network and assume that all brokers can be trusted to enforce the access control policies correctly. Any malicious, compromised or unauthorised broker would be free to read and write any events that pass through it on their way from the publisher to the subscribers. This might be acceptable in a relatively small system deployed inside a single organisation, but it is not appropriate in a multi-domain environment where organisations share a common infrastructure.

In order to enforce access control inside the broker network we propose encrypting event content and controlling access to the encryption keys. With encrypted event content only those brokers that have access to the encryption keys are able to access the event content (i.e. publish, subscribe to, or filter). We effectively move the enforcement of access control from the brokers to the encryption key managers.

We expect that access control has to be enforced in a multi-domain publish/subscribe system when multiple organisations form a shared publish/subscribe system that is used to deploy

multiple independent applications. Access control might also be needed when a single organisation consists of multiple sub-domains that deliver confidential data over the organisation-wide publish/subscribe system. E.g. the confidentiality of car numberplate sighting events must be protected in order to protect the privacy of citizens, but even more importantly the integrity of numberplate sighting events must be protected to prevent an intermediary system administrator from modifying the location of particular numberplate sightings in order to avoid congestion charges. Both cases require access control because event delivery in a dynamic publish/subscribe infrastructure based on a shared broker network may well lead to events being routed through unauthorised domains on their way from publishers to subscribers.

There are two main incentives for domains to share a common underlying publish/subscribe infrastructure. First, shared broker networks will generally be able to provide greater geographical reach without significant extra cost (i.e. initial infrastructure deployment cost dominates operational costs). Second, sharing a broker network will almost always increase the overall interconnectivity of the publish/subscribe infrastructure, thus providing higher fault-tolerance.

Figure 1 shows the multi-domain publish/subscribe network we use as an example throughout this paper. It is based on the United Kingdom Police Forces, and we show three particular sub-domains:

- **Metropolitan Police Domain.** This domain contains a set of CCTV cameras used to publish information about the movements of vehicles around the London area. We have included Detective Smith as a subscriber in this domain. Detective Smith has a court order that permits him to subscribe only to the *Numberplate* events necessary to track the car with numberplate "AE05 XYZ". Detective Smith is authorised to access all of the attributes of any such numberplate sightings.
- **Congestion Charge Service Domain.** The CCS contains the systems that manage the charges levied on the vehicles that have passed through the London Congestion Charge zone each day. The source numberplate recognition data comes from the cameras in the Metropolitan Police Domain. The fact that the CCS are only authorised to read a subset of the vehicle event data will exercise some of the key features of the enforceable publish/subscribe system access control presented in this paper. The CCS has a statistician whose role is to count the number of cars passing through a particular part of the congestion control area. The statistician is only authorised to access time and location information from numberplate sighting events. In contrast, the billing staff are authorised only to access the numberplate and time information.
- **PITO Domain.** The Police Information Technology Organisation (PITO) is the centre from which Police data standards are managed. It is the owner of the *Numberplate* event type in this particular scenario.

In addition to protecting the confidentiality of events in unauthorised domains, we can also use encryption to implement a more expressive access control mechanism and lower the number of events sent. By encrypting individual attributes, instead of the whole event as a single block, we are able to enforce attribute level access control in a multi-domain environment. Publishers and subscribers can be authorised only to access a subset of the attributes in an event type. With attribute level access control a single event instance can be delivered to subscribers with differing access rights. For example, in the congestion charge example scenario above the Congestion Charge Service would be authorised to read only the numberplate field, but not the location field, because the congestion charge is based on the vehicle being seen inside the congestion controlled area, not on the specific location of the vehicle inside that area. By hiding the exact location of the vehicle from the CCS the system protects the privacy of the vehicle owner.

While access control at the edge of the broker network is already implemented at the attribute level, it cannot be enforced among brokers unless each attribute is encrypted with its own encryption key. Without attribute encryption the system would have to trust the brokers to behave correctly in ignoring content they are not authorised to access.

The rest of the paper is organised as follows. Section II introduces relevant background, like decentralised publish/subscribe systems and decentralised trust management principles. In Sect. III we present our access control model for publish/subscribe systems consisting multiple domains. In order to enforce access control policy inside the event broker network, we introduce attribute encryption in Sect. IV. Section V discusses how access control policy is managed in our architecture. Finally Sect. VI presents related work in securing publish/subscribe systems and Sect. VII provides concluding remarks.

II. Background

This section gives a brief introduction to decentralised publish/subscribe systems, with particular emphasis on the assumptions made in this paper about multi-domain publish/subscribe systems, and to some related security features proposed in our previous work that we build on in this paper.

A. Decentralised Publish/Subscribe Systems

A publish/subscribe system consists of *publishers*, *subscribers*, and an *event service*. Publishers publish events, subscribers subscribe to events that are of interest to them, and the event service is responsible for delivering published events from the publisher to all subscribers whose stated interests, i.e. subscriptions, match the given event.

The simplest form of publish/subscribe system, called topic-based publish/subscribe [1], classifies events based on their topic. Subscribers subscribe to specific topics and receive all events published to that topic. Content-based publish/subscribe [1] allows filtering based on the content of an event, i.e. a subscriber defines a filter and only those events that match the filter are delivered to the subscriber. A content-based subscription S_a is said to cover another subscription S_b if the filter expression in S_a is more general, i.e. it matches more events, than the filter in S_b. The coverage

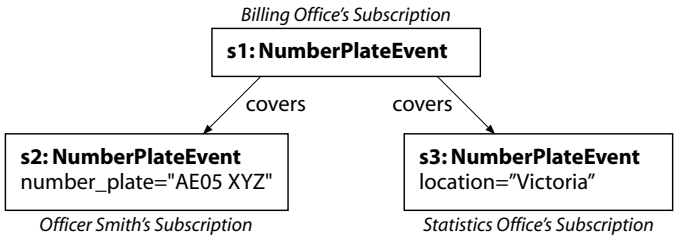

Figure 2. Subscription coverage in the example environment.

relation between the billing office's, statistics office's, and detective Smith's subscriptions can be seen in Fig. 2. The billing office's subscription covers the subscriptions of both detective Smith and the statistics office, because it specifies no filter expression. The coverage relation is useful when creating routing trees in a broker network. Instead of routing both subscriptions to the next broker in the broker network, a broker routes only the more general subscription S_a. Because S_a will match anything that S_b matches, the broker can be certain that it will receive all events that match either S_a or S_b.

Topic-based and content-based approaches can be combined so that a subscription is topic-specific, but it includes a filter expression for filtering events of that topic. This paper is concerned mostly with type-checked, content-based publish/subscribe systems where each event conforms to a predefined event type, and subscriptions may include filter expressions over the attributes of the event type.

In a decentralised publish/subscribe system the event service has been decentralised over a number of broker nodes. The brokers together form a broker network that shares responsibility for routing events from the publishers to interested subscribers. Event clients (publishers and subscribers) connect to a local broker which is trusted by the client to the extent of the authority granted to the broker by the domain's access control service. A local broker is a member of the the same domain as the client.

A broker network can have a static topology (e.g. Siena [2] and Gryphon [3]) or a dynamic topology (e.g. Scribe [4] and Hermes [5]). Our proposed approach will work in both cases. A static topology enables the system administrator to build trusted domains and make sure that confidential events are never routed through untrusted domains. This is very difficult with a dynamic topology that changes during runtime. On the other hand, a dynamic topology allows the broker network to dynamically re-balance itself when brokers join or leave the network either in a controlled fashion or as a result of a network or node failure. Therefore, a dynamic broker network provides more fault tolerance and is easier to maintain than a static topology network.

We have used Hermes as a base for our work. Hermes is a content-based publish/subscribe middleware that supports strong typing. That is, each publication is an instance of a predefined event type. Publications are type checked at the local broker of each publisher. Hermes uses a structured overlay network as a transport and therefore implements a dynamic broker network topology.

In addition to type-checking, Hermes also supports type inheritance. This means that a new type definition can extend an existing type definition by adding new attributes to the ones inherited from the super type. Type hierarchies allow domains and application developers to model *is-a* relationships between different event types. Our access control approach also supports type inheritance.

In this paper we assume that the underlying publish/subscribe system is type-based. Nevertheless, the work is equally applicable to topic-based publish/subscribe systems, except for the attribute level encryption scheme, which assumes that events are instances of predefined types.

B. Secure Event Types

We started our work towards a secure publish/subscribe system by introducing *secure event types* [6]. Secure event types provide integrity and authenticity of type definitions by using digital signatures. They also provide globally unique event type and attribute names, that are used to reference event types and attributes from access control policies. In this paper we use the globally unique names of the event type and its attributes to refer to the type or attribute from an access control policy.

An event type name, as proposed in [6], includes the public-key of the type owner. The public-key defines a unique namespace in which the type owner is free to create any names without the risk of name collisions. For example, the full name of the *Numberplate* event type would be: `PITO_pk.uk.gov.pito.Numberplate`, where `PITO_pk` represents the public-key of PITO.

C. Capabilities

Access rights in a system can be described with an *access control matrix* where the rows represent *subjects* (i.e. users), the columns represent *objects* (i.e. resources), and the cells define the *access rights* that a given subject has over a specific object.

Access control systems typically implement either a column centric view or a row centric view of the matrix. In a column centric view each column of the matrix is translated to an *access control list* (ACL) that is stored with the object that the column represents. The ACL contains entries for each subject defining the access rights for that subject regarding the specific object. An example of an ACL based system would be a typical UNIX filesystem where a simplified ACL is attached to each file.

In a row centric view each row of the matrix is translated to a set of capabilities that are stored with the subject. Each capability defines what access rights the subject, or holder of the capability, has over a given object.

A common use for certificates is to map principals to trusted identities. Rather than using certificates as an authentication token, however, it is also possible to use certificates for authorisation of actions. Certificates used in this manner are often referred to as *signed capabilities* or *authorisation certificates*.

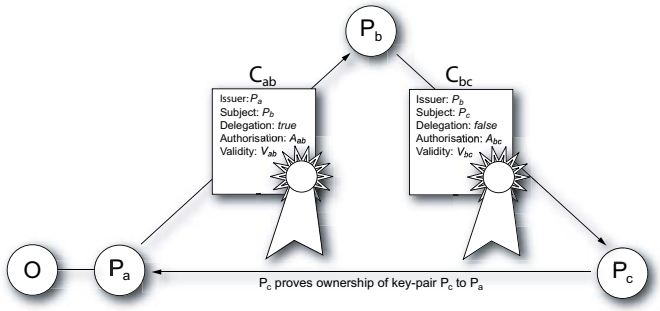

Figure 3. An SPKI authorisation certificate loop with three principals and two level delegation.

D. Decentralised Trust Management

Decentralised trust management was first introduced by Blaze et al. in [7]. *PolicyMaker* was later followed by *KeyNote* [8], and the *Simple Distributed Security Infrastructure* (SDSI) [9], which was later integrated with the *Simple Public-Key Infrastructure* (SPKI) [10] to create SDSI 2.0 [11].

The central idea in decentralised trust management is to decentralise access control decision making, and policy and credential management over all the principals in the system. This is achieved by requiring the *owner* of an object, or their delegatee, to issue capabilities to authorised subjects that grant access to the object. Distributing management responsibilities over all of the principals results in an extremely scalable access control system.

In SPKI the decentralisation is based on *certificate loops*. A typical certificate loop is depicted in Fig. 3 where the owner, P_a of an object O, grants P_b an SPKI authorisation certificate, C_{ab}, with access rights, A_{ab}, for the object O. P_b then further delegates access rights A_{bc}, where $A_{bc} \subseteq A_{ab}$, to P_c by granting P_c another authorisation certificate C_{bc}. Now, when P_c wants to access O, she shows P_a both certificates C_{ab} and C_{bc}. P_a is now able to form a certificate chain from P_c to P_b via C_{bc} and from P_b to itself via C_{ab}. Finally P_c authenticates herself to P_a by proving ownership of the key-pair P_c. P_c does this by executing a public-key challenge-response protocol with P_a. This completes the certificate chain which now forms a *certificate loop* flowing from P_a to P_b to P_c and back to P_a again. P_a has now verified that P_c is authorised to access O within the privileges granted by $A_{ab} \cap A_{bc}$[1]. Typically the verification is performed by an access control service rather than P_a. Note that in SPKI, principals and key-pairs are synonymous.

An SPKI authorisation certificate is a 5-tuple containing the following items: *Issuer*, *Subject*, *Delegation*, *Authorisation*, and *Validity*. *Issuer* is the public-key (or a hash of the public-key) of the issuer of the certificate. *Subject* is the public-key of the entity the certificate is issued to. *Delegation* is a boolean value specifying whether the *Subject* is permitted to further delegate the *Authorisation* granted by this certificate. In a certificate chain all certificates bar the last one must have *Delegation* set to $true$, otherwise the certificate chain is not valid. *Authorisation* is an application specific representation of access privileges granted to the *Subject* by the *Issuer*. And finally, *Validity* defines the date range when the certificate is valid and optional on-line validity tests, e.g. *certificate revocation lists* (CRLs).

Two certificates are reduced to a single 5-tuple as follows:

$$< I_1, S_1, D_1, A_1, V_1 > + < I_2, S_2, D_2, A_2, V_2 > \Rightarrow < I_1, S_2, D_2, A_1 \cap A_2, V_1 \cap V_2 >$$
$$\text{iff } A_1 \cap A_2 \neq \emptyset,\ V_1 \cap V_2 \neq \emptyset,\ S_1 = I_2 \text{ and } D_1 = true$$

This 5-tuple reduction rule is applied recursively to the certificate loop in order to collapse the loop to a single 5-tuple that will then be evaluated.

Our work relies on SPKI authorisation certificates to propagate authorisation from resource owners to domains in a decentralised and scalable fashion.

III. Access Control in Publish/Subscribe

We envision a multi-domain publish/subscribe system, as explained in Sect. I, where each domain contains a number of event clients and brokers, and an *access control service*. The access control service is responsible for granting privileges to brokers and clients in that domain according to the domain's internal access control policy.

One of the domains in the system is the *coordinating domain* which coordinates the formation of the shared publish/subscribe system. The coordinating domain forms the shared publish/subscribe system by inviting other domains to join the existing system. Because the coordinating domain forms the publish/subscribe system, it can be seen as the owner of the shared infrastructure and is therefore responsible for managing access to it.

The incentive for domains to join the network is twofold: on the one hand domains are interested in implementing shared applications with other domains, e.g. when one domain produces events while others consume them. On the other hand, even with domain-internal applications, the increased number of brokers increases the reliability and geographic coverage of the broker network, i.e. a broker network with a larger number of nodes is increasingly fault-tolerant and the extended coverage enables participating domains to reach a larger geographic area with lower infrastructure investments. Both incentives are attractive to domains, but only if the system provides appropriate access control mechanisms to prevent unauthorised access to deployed applications.

The publish/subscribe system needs to control access to a number of resources, e.g. the shared publish/subscribe infrastructure and various event types that have been introduced to the system. All of the following actions have to be authorised: (i) nodes, i.e. event brokers and event clients, connecting to the broker network; (ii) type owners introducing new types to the system; (iii) type owners extending an existing event type by inheritance; (iv) event clients accessing the publish/subscribe API, i.e. to publish or subscribe to events of a given event type.

[1]The verification process will also consider optional validity conditions (e.g. expiration dates) for each certificate in the chain. Any single invalid certificate will render the whole chain invalid.

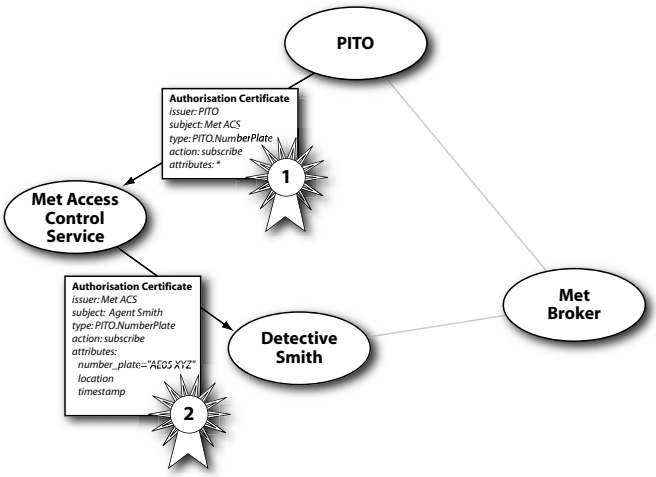

Figure 4. The authorisation certificates form a certificate chain linking detective Smith to the type owner, i.e. PITO.

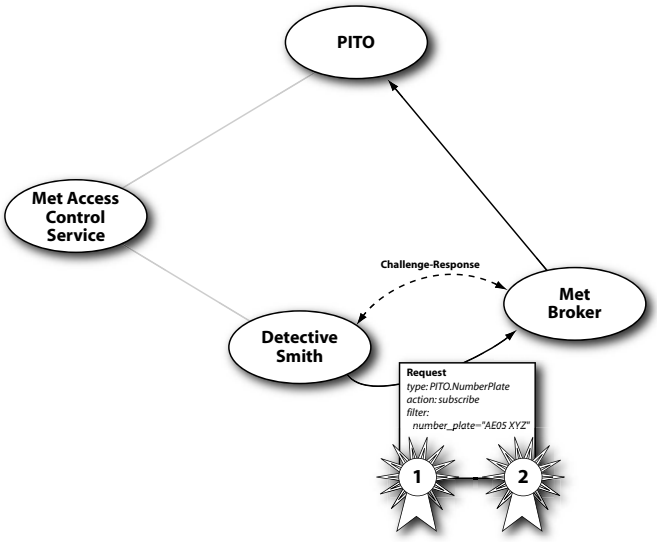

Figure 5. The Met Broker closes the certificate chain into a certificate loop by challenging Detective Smith to authenticate himself.

We propose a common approach to access control for publish/subscribe systems where access control decisions in all four cases are ultimately rooted at the appropriate resource owner. In (i) and (ii) the resource owner is the coordinating domain. In (iii) and (iv) the resource owner is the type owner. Employing authorisation certificates in the architecture and distributing the access control policy management, decision making, and credential management over all resource owners enables the access control architecture to scale well in an environment consisting of multiple independent domains.

A. Delegating Authority

A resource owner delegates authority to another subject by issuing an authorisation certificate for use by that subject. In our model the resource owner grants an authorisation certificate to a domain's access control service. The authorisation certificate authorises the access control service to further delegate the granted authority to nodes (e.g. event brokers, event clients, and access control services in sub-domains) in that domain. The access control service is then responsible for further delegating the authority to nodes by issuing authorisation certificates to them according to the domain-internal access control policy.

The authorisation certificates form a certificate chain that links the node to the resource owner through the access control service, as seen in Fig. 4. The link between the resource and the resource owner is implied either by the resource owner's public key being a part of the resource's identifier (i.e. event types, see Sect. II-B) or by the resource owner being explicitly specified (i.e. the coordinating domain being specified as the owner of the publish/subscribe system in the configuration of all participating nodes). This is shown as the grey line between the Met Broker and PITO in Fig. 4. The verifier forms a certificate loop out of the certificate chain by verifying that the node making the access request is able to authenticate itself as the subject in the last authorisation certificate in the chain, as depicted in Fig. 5.

B. Verifying Authority

When a client makes a publish/subscribe API request, it shows the authorisation certificate it received from the access control service and the authorisation certificate(s) linking the access control service to the resource owner. The authorisation certificates form a certificate chain from the client to the resource owner via the client's domain's access control service. The verifier then (i) verifies the signatures on the authorisation certificates; (ii) verifies the certificate chain that they form, i.e. the verifier reduces the chain to a single 5-tuple (see Sect. II-A), verifies that the validity conditions are met and that the client's request is authorised by the authorisation certificates; and finally (iii) executes a public-key challenge-response protocol with the requesting client, as depicted in Fig. 3. If everything validates correctly, the client's request is processed.

Note that the certificate chain must end at a resource owner that the verifier trusts. That is, the prover and the verifier must have a common trust root. In the coordinating domain's case, all nodes participating in the shared publish/subscribe system trust the coordinating domain (all nodes in the system have the coordinating domain's public-key and the node's local access control service's public-key installed as trusted keys). In the case of the type owners, the owner's public-key is included in the name of the type (see Sect. II-B). This links together the root of the certificate chain, i.e. the type owner, and the type name. Thus the verifier is able to trust that the issuer of the first certificate in the certificate chain is actually authorised to do so.

We assume that authorisation certificates will be delivered to the domain access control services out-of-band (i.e. outside of the publish/subscribe system that is being protected). Out-of-band delivery and the use of decentralised trust management principles allows the resource owners to change access control policy after deployment by issuing new and revoking existing authorisation certificates. For example, the coordinating

domain is able to accept a new domain to the common broker network simply by issuing an authorisation certificate to the joining domain's access control service without having to update policy files at each node in the publish/subscribe system. Similarly, a type owner is able to grant a new domain access to an existing event type by issuing the domain's access control service an authorisation certificate granting access rights to that event type.

C. Access Rights

Our model has four types of access rights that can be granted to nodes. The first type, access to the broker network, is required from every node that wants to join the broker network. Without access to the broker network a node is not able to use the publish/subscribe system. The second type, the right to introduce new event types, is necessary if a domain wants to deploy its own event types on the shared publish/subscribe system. If a domain is not allowed to deploy its own event types, that domain is effectively dependant on the event types defined and deployed by other domains and as such unable to deploy its own applications. The third type of authority allows domains to extend other existing event types by inheriting the existing type. Where the right to introduce new types to the system is controlled by the coordinating domain, the right to extend an event type is controlled by the owner of the type being inherited. The credentials in both cases must be included in the type definition, as explained in [6], so that the authenticity and integrity of the event type definition can be defined without any external evidence. The final type of authority is the authority to access the publish/subscribe API for a specific event type. The access rights specify the type of access granted to the node, i.e. publishing rights, subscription rights or both. The access rights also specify which attributes the node is allowed to access. In some cases, e.g. the Congestion Control Service presented in Sect. I, a node is authorised to receive or publish an event type, but it is not authorised to see all the content in those events. In the example of the Congestion Control Service the service needs to see the numberplate of a car entering the congestion controlled area, but there is no need to see the location of the car when it was seen, because the car is required to pay a fee based on it entering the area, not based on its location when there.

The authorities described below can be combined into a single authorisation certificate assuming that the granted authority is a subset of the issuer's access rights. For example, PITO, as the coordinating domain of the UK Police Network, could issue a single certificate granting both `connect` and `install` rights to the access control service of the Metropolitan Police Force domain (e.g. `action: connect|install`). The authority field also supports the asterisk wildcard, e.g. `action:*` in order to enable all actions. It is important to notice that the resource owner can only grant authority for her own resources. For example, PITO, as a type owner, can grant the Met domain the authority `type:*`, but the access rights will apply only to the types owned by PITO. Similarly with coordinating domains, the authority `network:*` will grant access to the networks owned and managed by that coordinating domain.

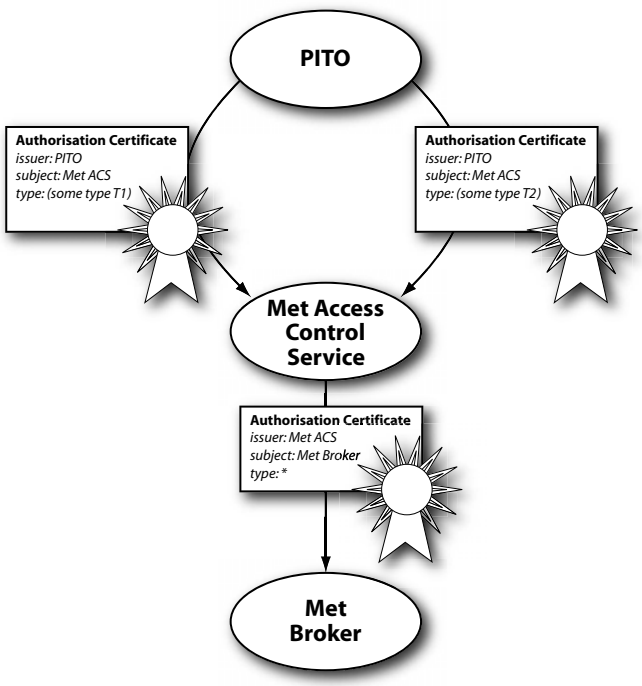

Figure 6. Once the Met Broker has received the `type:*` authorisation certificate, it can use it to access any further type authorised to the Met ACS.

In most cases we expect the event brokers of a domain to shared the privileges of the access control service. For example, all event brokers in a domain are expected to be able to access all event types that the access control service can access, because it allows all brokers to implement efficient content-based routing for all event types accessed by that domain. To achieve this the access control service can issue authorisation certificates to the brokers with an unrestricted action field, i.e. `action:*`. This grants the brokers access to everything that the access control service is allowed to access, assuming that the broker can show the authorisation certificate of the access control service that links the broker's certificate to the resource owner. If the access control service is granted new access rights after the deployment of the brokers, as is the case with *T2* in Fig. 6, the access control service can just broadcast the new authorisation certificate to all brokers. Because of the blanket authorisation granted to the brokers, they are able to utilise the new authorisation certificate without requiring new certificates to be issued to them.

Obviously granting blanket authorisations can be dangerous, so one must be exercise care when doing so. We would assume that a blanket authorisation would be specific to event types, or a even a type owner, e.g. `PITO_pk.uk.gov.pito.*`.

In some rare cases it might be necessary to deny some brokers access to particular event types. But these cases are usually handled more efficiently and more elegantly by placing all privileged brokers into a sub-domain of their own with their own privileged access control service.

1) Broker Network Access: The lowest level access control decision in the system is granting nodes access to the broker network. The authority is rooted at the coordinating domain which is seen as the owner of the shared publish/subscribe

system resource.

The aim of controlling access to the broker network is to prevent unauthorised parties accessing the event service in any way. If a malicious node is able to access the event service, it is able to launch a simple denial of service attack by issuing large amounts of subscription requests, or by injecting invalid routing messages to the broker network, which can lead to formation of network partitions. By controlling access to the broker network we can easily prevent trivial DoS attacks.

The granted authority is very coarse compared to the publish/subscribe API related access rights: it grants or denies access to a specific publish/subscribe network. The authority specifies the name of the network and the authorised action which in this case are `UK Police Network` and `connect`, respectively:

```
issuer: PITO
subject: Metropolitan Police
network: UK Police Network
action: connect
```

When connecting to a broker, both parties should verify each others' credentials. This is to prevent a client from connecting to a malicious broker that pretends to be part of the system, but in reality is not.

We assume that all links in the publish/subscribe system are protected by *Transport Layer Security* (TLS). Securing the communication links between nodes with TLS is a simple way to prevent trivial network sniffing and flooding attacks. Nodes create a TLS connection between each other. During the TLS handshake both parties present their credentials (an authorisation certificate granting access to the given broker network) and prove ownership of their own public-key. Notice that the trust relationship between the two nodes is based on them sharing a common ancestor in the certificate tree, e.g. the coordinating domain is the root of the authorisation certificate chain for two brokers from two different domains. Or the local access control service is the a common ancestor for an event client and a broker of the same domain.

2) Introducing New Types: The authority to introduce new types is also controlled by the coordinating domain since this is seen as a function related to the publish/subscribe system. The right to introduce new types to the system enables domains to deploy their own applications. Otherwise a domain is dependent on the types and applications deployed by other domains. In the following example PITO has granted the Met domain the right to deploy new event types:

```
issuer: PITO
subject: Metropolitan Police
network: UK Police Network
action: install
```

3) Extending Types: Extending an existing event type is related to the type that is being inherited. Therefore the owner of that type is responsible for managing access rights to extend the given type.

By using wildcards the type owner is able to include multiple types into one certificate. In the example below, authority is granted to extend all types with names starting with `uk.gov.pito` owned by the PITO:

```
issuer: PITO
subject: Metropolitan Police
type: uk.gov.pito.*
action: extend
```

The authorisation certificate granting authority to install a new type or extend an existing type must be included in the new type definition. This way the type definition is a self-contained package and its validity can be verified without external assistance when a client hosting broker is validating a client's request [6].

4) Accessing the Publish/Subscribe API: Accessing the publish/subscribe API is always related to a specific event type. Therefore the type owner, who is seen as the resource owner, is responsible for issuing an authorisation certificate to the access control service in each domain that is authorised to access that event type. The domain's access control service is then responsible for delegating a subset of its authority to nodes in the domains by issuing the nodes authorisation certificates that specify the nodes authority with respect to the given type.

The authority for accessing the publish/subscribe API can be very fine grained. On the one hand the access rights can be very specific granting the node only publication rights for a specific event type with a subset of the event type's attributes. On the other hand the access rights can be extremely generous granting the client access to all types and all attributes in those types both for publishing and subscribing. In the following example the Met domain has granted a CCTV camera located near Victoria the right to publish *Numberplate* events:

```
issuer: Metropolitan Police
subject: CCTV Camera at Victoria
type: uk.gov.pito.Numberplate
action: publish
attributes: location = "Victoria"
            numberplate
            timestamp
```

Notice that the authority can place restrictions on event content. For example, above the value of `location` is restricted to be `Victoria`. When publishing this means that the publisher hosting broker forces the attribute value if the publisher tries to publish a different value than `Victoria`. Similar restrictions can also be used when granting subscription rights to an event client. For example, detective Smith of the Metropolitan Police might have temporary subscription rights to *Numberplate* events where the value of the `numberplate` field is restricted to a specific number plate that is relevant to a case that the police officer is working on. In such a case the local broker would force a filter, e.g. `numberplate = "AE05 XYZ"`, on detective Smith's subscription.

In addition to the restricted access to the `location` attribute, the CCTV camera has unrestricted access to the `numberplate` and `timestamp` fields.

Assuming that the access control policy is enforced by encrypting attributes it is not possible to force restrictions on individual attributes in the broker network. For example, PITO

can not force one domain to publish *Numberplate* events with the location set to `Victoria` in all publications. Restricting attributes on a domain level would mean that each broker on the publication's path from the publisher to all subscribers would have to decrypt all attributes of the event and check that the event content conforms to the credentials of the previous broker. Therefore we assume that restrictions on attribute values are used only when issuing authorisation certificates to event clients where the client hosting broker is able to enforce the restriction.

When a client has access only to a subset of the attributes in an event type, the client hosting broker will replace the other attributes with null values. In the case of a publisher, the publisher hosting broker sets all attributes that the publisher is not authorised to access to `null`. If a subscriber is not authorised to access an attribute in a publication, the subscriber hosting broker delivers the publication to the subscriber with the unaccessible attributes set to `null`.

IV. Attribute Encryption

Real-world events often include confidential data that should be accessible only to authorised subjects, e.g. in our numberplate example the location of the numberplate sighting is seen as a piece of confidential data that should not, in most cases, be revealed to parties that are privy to the numberplate, because the privacy of the numberplate owner is compromised when the event specifies both the sighted numberplate as well as the location where it was seen. For example, the CCS billing service is not allowed to see the location where a given numberplate was sighted in order to protect the privacy of the numberplate owner. Detective Smith on the other hand is authorised by a special court order to see both fields in order to be able to track the numberplate through the city. These conditions will need to be specified in the rules of a deployed policy management system. That is, the policy defines which brokers have access to which attributes in event types.

The policy is enforced by encrypting each attribute of each event type in the system with its own encryption key. Authorised brokers are given the encryption key that allows the broker to encrypt (in case of publishing events or subscribing to an event type) or decrypt (in case of content-based filtering and delivering events to subscribers) event content. Notice that clients do not access the encryptions keys directly. Instead the local broker of a client handles encryption and decryption for the client. It is assumed that the client is able to trust the local broker sufficiently for the broker to act on the client's behalf. Therefore, clients must always connect to brokers that have the required access rights for the client to access the event types and attributes that it needs to access.

To protect the confidentiality of attributes in events we must also encrypt the attribute filters defined in subscriptions. Otherwise unauthorised brokers could deduce the value of an attribute by looking at the subscription filter that matched the given event.

Although our approach introduces run-time overhead due to the cryptographic operations on the attributes of publications and subscriptions, it allows the same publication to be disseminated to subscribers with different privileges, thus using the event dissemination tree efficiently. Our experiments in [12] indicate that encrypting each attribute separately instead the complete event can decrease the overall cryptographic overheads. This is because in most cases filtering is based on just one or two attributes instead of the whole event content.

A. Coverage Relations with Encrypted Filters

In order to take advantage of subscription coverage in content-based publish/subscribe systems when encrypting attributes, we extend the coverage relation to handle subscriptions where the filter expressions have been encrypted.

We treat the filter expression in a subscription as a conjunction of attribute filters. Each attribute filter is encrypted. An encrypted attribute filter expression is covered by a previous encrypted filter expression if the previous filter is the same or more general, and the broker making the comparison is authorised to access the attribute being filtered, i.e. the broker has access to the encryption key. A subscription is then covered by another subscription if all its filter expressions are covered. More formally, if s_a and s_b are two subscriptions with a conjunction of filter expressions f^i and g^i encrypted under the key k_i.

$$s_a = f^1_{k_1} \wedge f^2_{k_2} \wedge \ldots \wedge f^n_{k_n} \quad (1)$$

$$s_b = g^1_{k_1} \wedge g^2_{k_2} \wedge \ldots \wedge g^m_{k_m}, \quad (2)$$

then s_a covers ($\sqsupseteq$) s_b is defined as follows:

$$s_a \sqsupseteq s_b \iff \forall i \exists g^i_{k_i}.\ f^i_{k_i} \sqsupseteq g^i_{k_i} \ \wedge\ k_i \in K_{broker}, \quad (3)$$

where K_{broker} is the set of encryption keys accessible to the broker.

B. Encryption Keys

We use symmetric keys to encrypt and decrypt attribute values. These keys are distributed only to the brokers that are trusted with the attribute values. The system will never deliver these keys to clients. This reduces the number of nodes that are trusted with sensitive keys, and that take part in key management protocols. Note that this does not affect security since local brokers encrypt and decrypt attribute values on behalf of connected clients, and deliver events to clients over TLS secured communication links.

Encryption in a decentralised publish/subscribe system can be seen as a sub-category of secure group communication. In both cases the key management system must scale well with the number of nodes, nodes might be spread over large geographic areas, there might be high rates of churn in group membership, and all members must be synchronised with each other in time in order to use the same encryption key at the same time.

There exists a number of scalable key management protocols for secure group communication[13]. We have implemented the *One-Way Function Tree* (OFT) [14] protocol on top of a structured overlay network as a proof of concept. Our implementation uses the same structured overlay network

used by the broker network as a transport. The OFT protocol is based on a binary tree where the participants are at the leaves of the tree. It scales in $\log_2 n$ in processing and communication costs, as well as in the size of the state stored at each participant, which we have verified in our simulations.

Efficient group key management is not the focus of this paper. Overall, the efficiency of key distribution will have little impact on performance, since symmetric keys are distributed only to brokers, as opposed to publishers and subscribers. Relatively few entities are involved in key dissemination, and changes will be infrequent. However, correct key management is essential for the security of the system.

V. Access Control Policy

Whilst this paper primarily focuses on mechanisms for the enforcement of access control, the specification of access control policy is also necessary for any complete deployment. Our current implementation decouples policy management from access control enforcement, allowing many possible forms of policy management. This section discusses using the OASIS Role-Based Access Control (RBAC) framework for the policy management of our distributed publish/subscribe architecture.

The access control policy for a resource is done at multiple levels: for example, at the resource owner and the authorised access control service. In the first level, the resource owner authorises the access control services within each domain to access a resource. In the second level, the access control service of a domain implements an independent, domain-internal access control policy which defines the access rights granted to domain members. This means that the resource owner has no control over authorisation certificates issued to domain members by the domain's access control service. We assume that the resource owner is willing to trust the access control service of a domain within the extent of the authorisation certificate. This seems reasonable assuming that the resource owner is able to easily revoke access from misbehaving access control services. We will provide an illustration using OASIS policy at the domain level, but potentially the different levels could use independent policy languages.

A. OASIS

The *Open Architecture for Secure Interworking Services* (OASIS) [15], [16], provides a comprehensive rule-based means to check that users can only acquire the privileges that authorise them to use services by activating appropriate roles. A role activation policy comprises a set of rules, where a role activation rule for a role r takes the form:

$$r_1, .., r_n, a_1, .., a_m, e_1, .., e_l \vdash r$$

where r_i are prerequisite roles, a_i are appointment certificates (most often persistent credentials) and e_i are environmental constraints. The latter allow restrictions to be imposed on when and where roles can be activated (and privileges exercised), for example at restricted times or from restricted computers. Any predicate that must remain true for the principal to remain active in the role is tagged as a *role membership condition*. Such predicates are monitored, and their violation triggers revocation of the role and related privileges from the principal.

An authorisation rule for some privilege p takes the form:

$$r, e_1, .., e_l \vdash p$$

An authorisation policy comprises a set of such rules. OASIS has no negative rules, and satisfying any one rule indicates success.

OASIS roles and rules are parametrised. This allows fine-grained policy requirements to be expressed and enforced, such as exclusion of individuals and relationships between them, for example *caseAssignment(detective-ID, case-ID)*. Without parametrisation it becomes necessary to define an unmanageably large number of roles for an organisation of any size.

B. OASIS Policy in Our Example Scenario

In our scenario Detective Smith is only permitted to receive events relating to the sighting of a particular numberplate. We have indicated how the publish/subscribe system can enforce these types of access control, but have not discussed how to specify this in terms of policy within a domain.

In this section we show how the OASIS policy language could be used to specify a simple rule required by our example scenario. We propose that the courts are equiped with a means to issue a warrant as an OASIS appointment certificate. This appointment certificate has parameters that specify which case and which numberplate the warrant has been issued for.

The domain access control policy can then ensure that the subscription privilege is granted only to detectives who have been assigned to the case. An environmental predicate, $caseAssignment$, records the mapping between cases and detectives.

An appropriate OASIS role activation rule would be:

$$\begin{gathered} detective(detId), \\ caseAssignment(detId, case), \\ courtOrder(case, np) \vdash npTracker(detId, np) \end{gathered}$$

The target role membership certificate, $npTracker$ is parameterised with the detective's identity and the numberplate being tracked. This certificate can subsequently be represented as an authorisation certificate that the detective presents to a local broker when they want to subscribe to numberplate events.

C. Access Rights Revocation

Rapid, reliable, distributed revocation of certificates is a non-trivial problem, yet one for which we need a mechanism if access control policy in an access control system such as ours might ever need to be revised.

Traditional approaches to certificate revocation include expiry dates, certificate revocation lists (CRLs), and various online tests. When a given revocation occurs through any of these mechanisms, the OASIS policy rules for which that credential was a prerequisite for can be scanned to effect *active*

security – we can send events to the other parties that need to be notified of this revocation. That is, if a subject loses her role membership or authority, all registered parties will be notified. This allows, for example, the Met event brokers to be notified if detective Smith loses the authority to subscribe to *Numberplate* events.

In the case of resource owners, we assume that the authorisation certificates issued to domains are relatively long lived. If a resource owner wants to change the access control policy related to a resource, in most cases it is not an option to wait for an authorisation certificate to expire in order to revoke a domains access rights concerning a resource, e.g. access to a publish/subscribe broker network or to an event type. Therefore the publish/subscribe system must provide a mechanism for resource owners to actively revoke authorisation certificates issued to domains. Such a mechanism can be based on CRLs, a notification based active security approach, or other on-line checks.

For domain access control services there are more options. A domain may decide to issue short-lived certificates in order to avoid having to deal with active certificate revocation mechanisms. It is plausible to assume that all event clients in a domain are required to reacquire their authorisation certificates regularly, e.g. once every day or once every hour. The domain is free to make a trade-off between convenience and the risk of unauthorised access between the time when the domain's access control policy changes and the current authorisation certificate expires. If such a trade-off is not acceptable in a domain, the domain can deploy active revocation methods similar to the ones discussed above to revoke access rights.

The proper approach to certificate revocation is very much application and environment dependent. In some cases very short expiry dates will suffice. Other applications and environments will require more complex real-time approaches.

D. Policy Evaluation at the Local Broker

Authorisation to advertise, publish or subscribe may also depend on dynamic conditions such as event type or content, date, time or frequency of publication. For example, a publisher may be restricted to publish events only between 9am and 5pm.

In our current model the publisher's authorisation certificate would expire at 5pm each day. The publisher would have to reacquire her certificate in the morning before 9am every day. In order to allow longer lived certificates and to lower the load on the access control service, we could implement the evaluation of the dynamic access condition at the verifier, i.e. at the publisher hosting broker in this case. A policy language such that used in OASIS could achieve this, as illustrated in section V-B.

Delegating the evaluation of the relatively simple dynamic conditions to the broker would require for us to define a minimal policy language which allows the access control service to define these conditions in the authority field of authorisation certificates. The broker could then evaluate the condition in the authority field and grant access if the condition evaluates to true. Defining such a policy language is part of our planned future work with respect to access control in multi-domain publish/subscribe systems.

VI. Related Work

Wang et al. in [17] categorised the various security issues that need to be addressed in publish/subscribe systems in the future. The paper is a comprehensive overview of security issues and as such tries to draw attention to the issues rather than provide solutions.

Opyrchal and Prakash address the problem of event confidentiality at the last link between the subscriber and the subscriber hosting broker in [18]. They correctly state that a secure group communication approach is infeasible in an environment like publish/subscribe that has highly dynamic group memberships and widely varying subscriptions. They propose a scheme utilising key caching and subscriber grouping in order to minimise the number of required encryptions when delivering a publication from a subscriber hosting broker to a set of matching subscribers. We assume in our work that the subscriber hosting broker is powerful enough to manage a TLS [19] secured connection for each local subscriber.

Srivatsa and Liu present EventGuard in [20]. EventGuard provides event confidentiality, integrity and authenticity in decentralised publish/subscribe systems and as such is very similar to our work. Srivatsa and Liu address the security problems in publish/subscribe system in a bottom-up approach where they provide confidentiality and integrity guarantees with cryptographic primitives. In contrast to our work they do not address managing access control. Our approach is a top-down approach where we motivate the need for access control in multi-domain environments and provide a solution built on primitives like certificates and digital signatures.

Zhao and Sturman propose an approach to dynamic access control in a content-based publish/subscribe system in [21]. In contrast to our work they propose a centralised, access control list based architecture, which, while perfectly acceptable for single domain deployments, will not in our opinion scale to multiple domains.

We base our work on that presented in previous papers related to the topic: [12] introduces the multi-domain environment and proposes a high-level access control approach based on role-based access control, [6] provides the basis for our capability-based access control model, which was originally introduced in [22].

VII. Conclusions and Future Work

We have presented an SPKI-based access control architecture for multi-domain publish/subscribe systems. By applying decentralised trust management, we are able to administer and enforce access control in publish/subscribe systems that span multiple independent administrative domains both conveniently and in a scalable manner.

There are aspects of future work resulting from this paper. So far we have focused on allowing clients to verify that they are communicating using consistent event types/topics through checking the certificate chains that authorise their use of them. We are keen to move towards encrypting events, or

attributes of them, so that we can enforce access control in an untrusted broker network, and thus evolve away from a boundary-oriented access control approach.

VIII. Acknowledgements

Lauri Pesonen is supported by EPSRC (GR/T28164). David Eyers is supported by EPSRC (GR/S94919).

References

[1] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, Aug. 2001.

[3] P. R. Pietzuch and S. Bhola, "Congestion Control in a Reliable Scalable Message-Oriented Middleware," in *Proc. of the 4th Int. Conf. on Middleware (Middleware '03)*, M. Endler and D. Schmidt, Eds. Rio de Janeiro, Brazil: Springer, June 2003, pp. 202–221.

[4] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications (JSAC)*, vol. 20, no. 8, pp. 1489–1499, Oct. 2002.

[5] P. R. Pietzuch and J. M. Bacon, "Hermes: A Distributed Event-Based Middleware Architecture," in *Proc. of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*. Vienna, Austria: IEEE, July 2002, pp. 611–618.

[6] L. I. W. Pesonen and J. Bacon, "Secure event types in content-based, multi-domain publish/subscribe systems," in *SEM '05: Proceedings of the 5th international workshop on Software engineering and middleware*. New York, NY, USA: ACM Press, Sept. 2005, pp. 98–105.

[7] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proc. of the IEEE Conference on Security and Privacy*. Oakland, CA, USA: IEEE, May 1996.

[8] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "KeyNote: Trust management for public-key infrastructures (position paper)," in *Proc. of the Cambridge 1998 Security Protocols International Workshop*, vol. 1550, 1998, pp. 59–63.

[9] R. L. Rivest and B. Lampson, "SDSI – A simple distributed security infrastructure," Presented at CRYPTO'96 Rumpsession, Oct. 1996.

[10] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylönen, "SPKI certificate theory," Internet Engineering Task Force, RFC 2693, Sept. 1999.

[11] CIS, "SDSI (a simple distributed security infrastructure)," Sept. 2001. [Online]. Available: http://theory.lcs.mit.edu/~cis/sdsi.html

[12] J. Bacon, D. M. Eyers, K. Moody, and L. I. W. Pesonen, "Securing publish/subscribe for multi-domain systems," in *Middleware*, ser. Lecture Notes in Computer Science, G. Alonso, Ed., vol. 3790. Springer, 2005, pp. 1–20.

[13] S. Rafaeli and D. Hutchison, "A survey of key management for secure group communication," *ACM Computing Surveys*, vol. 35, no. 3, pp. 309–329, 2003.

[14] D. A. McGrew and A. T. Sherman, "Key establishment in large dynamic groups using one-way function trees," TIS Labs at Network Associates, Inc., Glenwood, MD, Tech. Rep. 0755, May 1998.

[15] J. Bacon, K. Moody, and W. Yao, "Access control and trust in the use of widely distributed services," in *Middleware 2001*, vol. LNCS 2218. Springer-Verlag, Nov. 2001, pp. 300–315.

[16] ——, "A Model of OASIS Role-Based Access Control and its Support for Active Security," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 4, pp. 492–540, Nov. 2002.

[17] C. Wang, A. Carzaniga, D. Evans, and A. L. Wolf, "Security issues and requirements in internet-scale publish-subscribe systems," in *Proc. of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*. Big Island, HI, USA: IEEE, 2002.

[18] L. Opyrchal and A. Prakash, "Secure distribution of events in content-based publish subscribe systems," in *Proc. of the 10th USENIX Security Symposium*. USENIX, Aug. 2001.

[19] T. Dierks and C. Allen, "The TLS protocol, version 1.0," Internet Engineering Task Force, RFC 2246, Jan. 1999.

[20] M. Srivatsa and L. Liu, "Securing publish-subscribe overlay services with eventguard," in *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*. New York, NY, USA: ACM Press, 2005, pp. 289–298.

[21] Y. Zhao and D. C. Sturman, "Dynamic access control in a content-based publish/subscribe system with delivery guarantees," in *Proc. of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, p. 60.

[22] L. I. W. Pesonen, D. M. Eyers, and J. Bacon, "A capabilities-based access control architecture for multi-domain publish/subscribe systems," in *Proceedings of the Symposium on Applications and the Internet (SAINT 2006)*. Phoenix, AZ: IEEE, Jan. 2006, pp. 222–228.