Securing Publish/Subscribe for Multi-Domain Systems

Jean Bacon, David Eyers, Ken Moody, and Lauri Pesonen

University of Cambridge Computer Laboratory
JJ Thomson Avenue, Cambridge, CB3 0FD, UK
{firstname.lastname}@cl.cam.ac.uk

Abstract. Two convincing paradigms have emerged for achieving scalability in widely distributed systems: ***role-based***, policy-driven control of access to the system by applications and for system management purposes; and ***publish/subscribe communication*** between loosely coupled components. Publish/subscribe provides efficient support for mutually anonymous, many-to-many communication between loosely coupled entities. In this paper we focus on securing such a communication service (1) by specifying and enforcing access control policy at the service API, and (2) by enforcing the security and privacy aspects of these policies within the service itself. We envisage independent but related administration domains that share a pub/sub communications infrastructure, typical of public-sector systems. Roles are named within each domain and role-related privileges for using the pub/sub service are specified. Intra- and inter-domain, controlled interaction is supported by negotiated policies. In a large-scale publish/subscribe service, domains are not expected to trust all message brokers fully. Attribute encryption allows a single publication to carry both confidential and public information safely, even via untrusted message brokers across a vulnerable communications substrate. Our approach provides the application designer with fine-grained expressiveness while, at the same time, improving system fault tolerance by allowing a single shared messaging network to route both public and confidential information. Early simulations show that our approach reduces the overall traffic compared with a secure publish/subscribe scheme that encrypts whole messages.

Keywords: publish/subscribe, loosely coupled applications, content-based routing, role-based access control, attribute encryption, message confidentiality, trust

1 Introduction

We are concerned with how communication within and between large-scale, independent, widely distributed application domains should be supported and managed. Two recently emerging paradigms for achieving scalability are asynchronous, publish/subscribe-based communication and role-based access control (RBAC). In the EDSAC21 project we aim to extend and integrate these paradigms to achieve a scalable, secure middleware capable of supporting fine-grained control of communication within and between domains. In this paper we present our multi-domain architecture and an interim evaluation based on simulation.

We define a domain to be an independently administered unit in which a domain manager has, or may delegate, responsibility for naming and policy specification. The following motivating scenarios have in common a communication infrastructure shared by independently administered domains, some of which are strongly related and have similarly named roles. The bulk of the communication is likely to be within a domain but there is also a clear need for inter-domain communication. (1) A global company has branches (e.g. sales) in California, London and Tokyo. Some (sales) data and events should be shared between branches. (2) A number of county-level police domains need support for intra- and inter-domain messages. (3) A national health service's communication infrastructure is shared by many independent hospitals, clinics, primary-care practices etc. (4) An "active city" has independent public services such as police, fire, ambulance, hospital, and utilities. As well as communicating with similar services nationally (e.g. police with police) the different services need to cooperate, especially in emergencies. Examples are common in the public sector, where systems have been particularly susceptible to expensive failure or curtailment.

The concept of role is well established for providing scalable security administration. Role-based access control (RBAC) separates the administration of people, and their association with roles, from the control of privileges for the use of services (including service-managed data). Service developers need only be concerned with specifying access policy in terms of roles, and not with individual users. Here we focus on securing the communication service. Domain managers, or their delegates, specify communication policy in terms of message types and roles; that is, which roles may create, advertise, send and receive which types of message. Inter-domain communication is achieved through negotiated agreements, expressed as access control policy, on which role(s) of one domain may receive (which attributes of) which types of message of another.

Publish/subscribe [1] is emerging as an appropriate communication paradigm for large-scale systems. It allows loose coupling between mutually anonymous components and supports many-to-many communication. In this paper we focus on securing publish/subscribe within and between domains. For consistency with other publish/subscribe systems we use *event* as synonymous with the more general term *message*. The notion of role is ideally suited to a multicast communication style. For example, the Cambridge police domain may define a role *sergeant-on-duty* and message topics such as *traffic-accident (attribute-list)*. Authorisation policy will indicate which roles can advertise, subscribe to and publish each topic. Inter-domain communication is supported, after human negotiation, by indicating in policy that a specified role of some domain may subscribe to certain (attributes of) topics published by some other domain.

Authentication into roles must be securely enforced to control the use of all protected services. We have addressed this in earlier papers. For the communication service, RBAC policy indicates the visibility (to roles, intra- and inter-domain) of specified attributes of message types. The fact that advertisement is required before messages can be published, and both are RBAC-controlled, prevents the spam that pervades email communication between humans. Without such control denial-of-service through publication or subscription flooding could degrade large-scale inter-software communication in the same way that it consumes resources in email management. In our system a spammer could only be an authorised, authenticated member of a role and therefore could be held accountable.

If the network and message brokers could be guaranteed 100% secure and trustworthy, then RBAC would achieve precisely the visibility specified by policy. In practice, we have to protect confidential data on the wire and in the brokers by means of encryption. We offer fine-grained security, in that message attributes are encrypted selectively, with key management transparent to the client level. We assume that some form of message encryption is always needed, since nodes of a communication service are not likely to be trusted universally with all data and the network is vulnerable to listeners. Encryption overhead per se does not need to be justified, and our evaluation indicates that our approach incurs less overhead than using whole-message encryption.

The contribution of this paper is to show how role-based access control, together with fine-grained data encryption and the associated key management, can be integrated with publish/subscribe based communication to create a secure middleware suitable for a wide range of large-scale, widely distributed application domains. First, we set the scene by discussing related research on secured publish/subscribe in Section 2. Section 3 gives background in publish/subscribe systems and role-based access control, emphasising, without loss of generality, the systems we have used for our implementation and evaluation, Hermes and OASIS. We then outline how RBAC and publish/subscribe are integrated. Section 4 presents our multi-domain architecture in more detail. Section 5 uses a multi-domain, networked city as a case study and describes the scenarios evaluated in Section 6. Section 7 presents our conclusions in the context of our ongoing and future research.

2 Related Work

To our knowledge, the architecture we outlined in [2] was the first to consider access control for a publish/subscribe service. There, we took a typical private-sector application, a newsfeed service, comprising a single naming and protection domain. We did not consider public-sector, multi-domain examples, where it becomes natural for a message-broker substrate to be shared, and where different levels of trust in brokers must be accommodated. This work did not address data encryption and key management.

Some authors explicitly exclude security as being orthogonal to the design issues of publish/subscribe [3]. Others have limited their work to the communications level [4]. Others have discussed how publish/subscribe systems might be secured but without explicit design details and evaluation. Wang et al. present a number of considerations for publish/subscribe access control in [5] but without proposing an architecture to solve the problems they raise. Similarly, in [6], Miklós provides semantics defining a security ordering based on event attribute values, but does not describe a practical test prototype. The approach of Miklós is likely to be too restrictive in practice; it will not scale well due to the detailed specifications required to define event security classes and how they interact.

Opyrchal and Prakash concentrate on the separate problem of providing confidentiality for events during the last hop from the local broker to the event subscriber in an efficient manner, with as few encryptions as possible [7]. Limiting the number of last hop encryptions is valuable if the local brokers have poor resources. We assume that the local brokers are powerful enough to deliver events to their subscribers over TLS connections [8]. While more resource-intensive, TLS provides us with strong client and server-side authentication and key management in addition to data encryption.

In sentient and ubiquitous computing environments privacy should be a major concern, for example, when individuals can be recognised automatically and tracked. This issue is not often considered. An exception is the Gaia project where the approach is to guarantee anonymity [9]. [10] is also concerned with anonymity in location systems. Publish/subscribe is based on mutual anonymity at the client level. Parametrised RBAC gives the option of anonymity or identification. However, principals are not anonymous to the system when authenticated into roles and the privileges of misbehaving principals can be withdrawn promptly. Attribute-level policy expression controls the selective propagation of identity attributes at a fine grain.

3 Background and Integration

Although our approach is generally applicable, our design and implementation are based on *Hermes* publish/subscribe and OASIS RBAC. This section provides a brief overview of publish/subscribe systems and role-based access control, describing the features specific to Hermes and OASIS. We then show how a publish/subscribe system can be secured by RBAC.

3.1 Publish/Subscribe Systems

Large-scale, publish/subscribe messaging technology typically comprises a network of brokers, which provide a communication service, and lightweight clients, which use the service to advertise, subscribe to and publish messages [11, 12]. Such systems are subject to failures of nodes and links, and their components may join and leave dynamically. A communication service must be robust under these conditions, fault-tolerant and dynamically reconfigurable. For this reason the message brokers are often built above a peer-to-peer overlay network [13], since peer-to-peer naming and protocols provide the necessary robustness.

Publish/subscribe systems are classified as type/topic- or content/attribute-based. Hermes [13, 14] is a distributed, content-based publish/subscribe architecture with an integrated programming model and strong message typing. It is built on a peer-to-peer routing substrate to provide scalable event dissemination and fault tolerance.

A Hermes system consists of two kinds of component: *event brokers* and *event clients*, the latter being *publishers* and *subscribers*. Event brokers form the application-level overlay network that performs event propagation by means of a content-based routing algorithm. Event clients publish, or subscribe to, events in the system. An event client has to maintain a connection to a *local event broker*, which then becomes *publisher-hosting*, *subscriber-hosting*, or both. An event broker without connected clients is called an *intermediate broker*.

Hermes supports *event typing*: every published event (or *publication*) in Hermes is an instance of an *event type*. An event type has an *event type owner*, an *event type name* and a list of typed *event attributes* so that, at runtime, publications and subscriptions can be type-checked by the system. Hermes event types are organised into inheritance hierarchies, but our work does not depend on this. We show later how inheritance can be used within domains when it is available.

Each event type defined within a domain is registered by its owner via a local event broker. This causes encryption status and keys to be set up within the domain and a rendezvous node to be selected for peer-to-peer routing. Before a publisher can publish an event instance, it must submit an *advertisement* to its local event broker, indicating the event type that it wishes to publish. Subscribers express their interest in the form of *subscriptions* that specify the desired event type and a conjunction of (content-based) filter expressions over the attributes of this event type.

The rendezvous node for an event type is selected by hashing the type name to a broker identifier – an operation that is supported by the peer-to-peer routing substrate [15]. Advertisements and subscriptions are routed towards the rendezvous node, and brokers along the path set up filtering state for them.

Most publish/subscribe systems, including Hermes, optimise content-based routing of events with a *subscription coverage relation*, that states which subscriptions are subsumed by others [11]. This allows brokers to reduce the number of events sent through the system by enabling them to filter non-matching events as close as possible to the publisher; these filters become increasingly specific as events approach subscribers.

For reliability reasons, rendezvous nodes are replicated for each event type (for example, broker instances can be selected by concatenating a *salt value* to the type name before hashing [16]). In Hermes, a rendezvous node keeps an authoritative copy of the event type definition, which is cached at other brokers throughout the system for type-checking advertisements, subscriptions, and publications. In our current work, authoritative, domain-specific type information is stored within the originating domain and rendezvous nodes hold a copy.

3.2 Role-Based Access Control

Role-Based Access Control (RBAC) [17] is an established technique for simplifying scalable security administration by introducing *roles* as an indirection between *principals* (i.e. users and their agents) and *privileges*. Privileges, such as the right to use a service or to access an object managed by a service, are assigned to roles. Separately, principals are associated with roles. The motivation is that users join, leave and change role in an organisation frequently, and the policy of services is independent of such changes.

The *Open Architecture for Secure Interworking Services* (OASIS) [18,19], 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 *treating-doctor(doctor-ID, patient-ID)*. Without parametrisation it becomes necessary to define an unmanageably large number of roles for an organisation of any size.

3.3 Integration

In OASIS RBAC, the authorisation policy for any service specifies how it can be used in terms of roles and environmental constraints. Here, we use OASIS to protect the publish/subscribe service in this way at a local broker. The service's methods include *define(message-type)*, *advertise(message-type)*, *publish(message-type, attribute-values)* and *subscribe(message-type, filter-expression-on-attribute-values)*. OASIS policy indicates, for each method, the role credentials, each with associated environmental constraints, that authorise invocation. *define* is used to register a message type with the service and specify its security requirements at the granularity of attributes. On *advertise*, *publish* and *subscribe* these requirements are enforced. We can therefore support secure publish/subscribe within a domain in which roles are named, activated and administered.

A domain-structured OASIS system is engineered with a per-domain, secure OASIS server, as described in [18], and a per-domain policy store containing all the role activation and service-specific authorisation policies. This avoids the need for small services to perform authentication and secure role activation. The domain's OASIS server carries out all per-domain role activation and monitors the role membership rule conditions while the roles are active. This optimisation concentrates role dependency maintenance within a single server and provides a single, per-domain, secure service for managing inter-domain authorisation policy specification and enforcement.

4 A Multi-Domain Architecture

In this section we present an architecture for an RBAC-secured, multi-domain publish/subscribe system based on a shared event-broker network. We assume that domains are given unique names within the system as a whole and that roles are named and managed within a domain. We assume that each domain provides a management interface through which role activation policies and services' authorisation policies can be specified and maintained.

A group of domains may have a parent domain from which an initial set of role names and policies is obtained. For example, county police domains may agree to use a nationally defined set of police roles; health service domains may start from an initial national role-set. The domain management interface allows local additions and updates, for example, when government changes national policy. Parametrised roles allow domain-specific parameters, for example *sergeant(Cambridgeshire)*. This avoids the role explosion when non-parametrised RBAC is used on a large scale.

4.1 The Event-Broker Infrastructure

RBAC enforces authorisation policy at the level of clients of the publish/subscribe service. At the service level we have to protect confidential data on the wire and in the broker network. Publisher-hosting brokers must encrypt messages to secure confidential information, first checking against policy that the publisher is authorised to send the attribute values. Subscriber-hosting brokers must decrypt messages and deliver to the subscriber the attributes that it is allowed by policy to read. These policies are specified when the message type is defined.

We distinguish between trusted and untrusted brokers. For example, a national police service may comprise some tens of county-level domains, deploying a (sub)network of brokers, trusted by all police domains. Statically, these brokers are trusted by police to encrypt and decrypt police data. Dynamically, under monitoring, some broker may come under suspicion and have trust withdrawn from it. The police domains may choose to route data through the untrusted brokers of other services, for example in rural regions. In general, police domains may interoperate with other emergency service domains and with the media or public via parts of the broker network that are untrusted.

A shared broker infrastructure may be built up when public sector domains agree to interoperate. Alternatively, a broker infrastructure may be provided commercially or as a public service, and independent, distributed applications

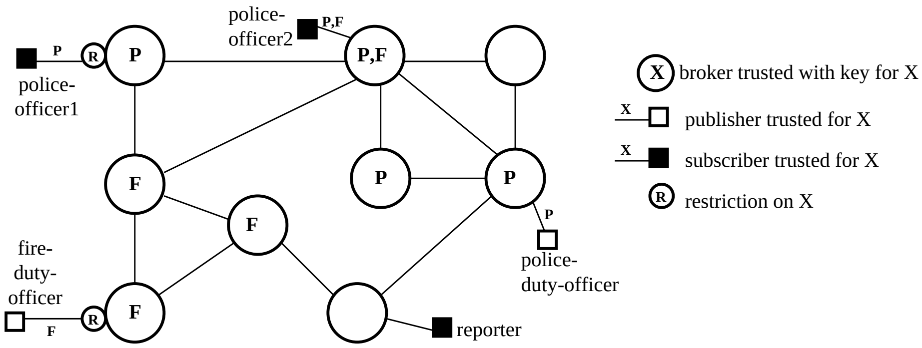

Fig. 1. Illustration of Secure Publish/Subscribe

may use it to communicate intra- or inter-application. In both scenarios the domains/applications will have different levels of trust in the various brokers.

A shared event-broker infrastructure offers both direct and indirect benefits: management overheads are reduced by operating only a single broker network instead of a separate one for each domain, with federation via gateways (as in [20]). Untrusted brokers can augment trusted brokers' routing abilities, ensuring better resilience to failures. These direct benefits are particularly significant when the network has many domains, and/or the domains are small. The indirect benefits of using a shared network are equally important: networks of trust can be established and reconfigured more easily, since the privileges of brokers and clients are controlled dynamically within a homogeneous access control scheme. Also, encrypting attributes separately allows a single event to contain both public and private information.

Key Management for Trust Groups. A broker network comprising multiple trust groups must have a key manager for each group. Some domains' OASIS servers will maintain key-groups of trusted brokers and distribute keys to them, transparently to the clients of the publish/subscribe service. A broker must be provided with credentials that allow it to join a trust group. Intermediate trusted brokers decrypt messages to achieve efficient content-based routing. Untrusted brokers participate in routing at the message, rather than attribute, level; details are given below. When a broker becomes untrusted, new keys must be distributed to the remaining group members. We do not address malicious brokers with byzantine behaviour that may corrupt routing state.

In Fig. 1 the brokers are annotated with the encryption keys to which they have access; **P** for the *police* key, **F** for the *fire* key. The broker to which the reporter is attached can deliver only unrestricted public data.

Suppose inter-domain communication is negotiated and an authorised subscription is made from an external domain that has brokers in a different trust group. The police and fire services of Fig. 1 are an example. Such a negotiated agreement, that events of one domain may be subscribed to from another, implies that the local brokers of publishers and subscribers are trusted to encrypt and decrypt the authorised attributes, and have the appropriate keys.

4.2 Policy

Policy and enforcement mechanisms must be in place to support:
(i) Secure connection by a new broker in order to become part of a group of trusted brokers.
(ii) Secure connection by a client to any trusted local broker.
(iii) Secure propagation of messages through the broker network with confidentiality of attributes enforced as specified by policy.
(iv) RBAC-controlled use of the publish/subscribe service by clients.

For (i) and (ii), publishers, subscribers and brokers hold public key pairs, bound to identity certificates (e.g. X.509 [21]), to connect to their local OASIS service. Successful authentication will allow brokers to become part of a trusted group for key management purposes, and will allow clients to proceed to request activation of the roles that authorise advertisements, subscriptions and publications.

The authentication key pairs are also used in creating client and server-side authenticated TLS connections between nodes. This prevents simple network sniffing attacks by outsiders, thus helping to achieve data confidentiality and integrity, contributing to (iii). For (iii) the key management service controls the propagation of attribute decryption keys to trusted brokers.

(iv) was introduced in Section 3. The authorisation policy for the *define(type)* operation specifies the credentials and constraints required for registering new message types with the publish/subscribe service in a domain, and subsequently for managing the registered types. It controls the ability to modify and remove existing types and, in Hermes, to create sub-types. When a parent domain exists it is likely that an initial set of message types will be used by all child domains, similar to the use of nationally agreed role-names within related domains. A type-specific *read-write policy*, if present, augments and refines the advertisement and subscription policies. It defines, at the attribute level, the roles that can read and/or write the various attributes of a type and can also restrict access by attribute value, see below.

An *advertisement policy* defines which roles are allowed to advertise, and then to publish, events of each given type. Environmental constraints may also be included, see Section 3. Their actions may be subject to further *restriction*, see below, as indicated by the type-specific *read-write policy*. A *subscription policy* defines the authorised receiver roles and conditions in a similar fashion. If required, individual clients can be identified using role parameters.

Restriction. A publisher or subscriber role may be authorised by the *publication* or *subscription* policies, but restricted by the type-specific *read-write policy* to a subset of the attributes of some event type that it requests, and/or for a subset of the values of certain attributes. Rather than reject the request outright, the local broker may allow the request after applying a restriction.

In the case of a publisher, any attribute value whose *read-write policy* does not include write access is ignored. A simple approach is to omit the attribute from the marshalled data, and supply a null value to subscribers. With a type hierarchy it may be possible to restrict publications to a super-type of the requested type, if *advertisement policy* allows that. In the case of a subscriber, the natural restriction is to suppress the attribute value whenever the subscriber does not have read access to an attribute under *read-write policy*.

Authorisation to advertise, publish or subscribe may also depend on conditions such as event type or content, date, time or frequency of publication. Thus a publisher may be restricted to publish certain events between 9am and 5pm. OASIS environmental constraints can specify and enforce some of these conditions, and publish/subscribe filtering may implement some forms of restriction by attribute value. In general, specific predicates must be computed by the local broker of the client to which the restrictions apply, see Section 5.

4.3 Attribute Encryption

Real-world occurrences often include confidential data that should be accessible only to authorised subjects, e.g. the press should know about a car accident on a highway, but the names of the victims should stay confidential to the police and health services. This is achieved by RBAC policy and mechanism at application level, and by encrypting attributes (in publications) and filter expressions (in subscriptions) with symmetric keys at the message service level, as outlined above. Although our approach introduces run-time overhead due to the cryptographic operations on 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. Section 6 shows that attribute encryption can decrease the overall cryptographic overheads.

Event Types with Attribute Encryption in Hermes. To indicate attribute encryption within the Hermes type system, we annotate the event type hierarchy with the keys that are used to encrypt specific attributes, reflecting defined policy. Local brokers of publishers and subscribers implement this security policy; clients are not concerned with encryption. Each attribute of an event type is either *public*, indicated by the empty key (0), or it must be encrypted using one or more keys. Fig. 2 shows annotated type hierarchies for Police and Fire Service domains. The `location` attribute in a `PoliceEvent` may be encrypted using both *police* and $fire$ keys. This would result in two instances of the same attribute in a single event, each instance encrypted with a different key.

The standard inheritance sub-typing relation between event types must still hold: a subtype has to be more specific than its parent type. As a result, encryption keys can only be removed from inherited attributes but not added. This is illustrated in Fig. 2 with the `location` attribute, whose access becomes more restrictive as new event types are derived.

Coverage Relations with Encrypted Filters. In order to take advantage of

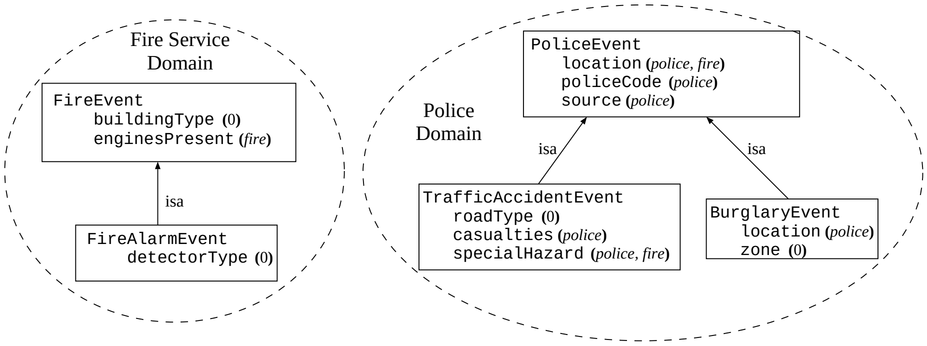

Fig. 2. Per-Domain Event Type Hierarchies with Attribute Encryption

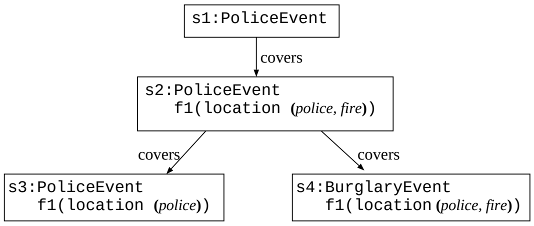

Fig. 3. Subscription Coverage with Attribute Encryption

subscription coverage (described in Section 3), we extend this relation to handle subscriptions that refer to encrypted attributes.

A filter expression encrypted under a particular key is covered by a previous filter expression if this previous filter is the same or more general, and is encrypted under the same key (including the case where neither expression is encrypted). A subscription is then covered by another subscription if all its filter expressions are covered. More formally, if s_1 and s_2 are two subscriptions with a conjunction of filter expressions f^i and g^j encrypted under the keys k_i and l_j, respectively,

$$s_1 = f^1_{k_1} \wedge f^2_{k_2} \wedge \ldots \wedge f^n_{k_n} \tag{1}$$

$$s_2 = g^1_{l_1} \wedge g^2_{l_2} \wedge \ldots \wedge g^m_{l_m}, \tag{2}$$

then s_1 covers ($\sqsupseteq$) s_2 is defined as follows:

$$s_1 \sqsupseteq s_2 \iff \forall i \exists j.\ f^i_{k_i} \sqsupseteq g^j_{l_j} \ \wedge \ k_i = l_j \tag{3}$$

We assume above that each subscription includes empty filters encrypted with all available keys by default.

The coverage relations between the example subscriptions s_1 to s_4 are shown in Fig. 3. Subscription s_1 is the most general because it does not specify any filter expressions. It covers the second subscription s_2, which specifies a filter f_1 over the `location` attribute encrypted under the *police* key or the *fire* key. Subscribers can only provide meaningful filters for encrypted attributes if they have read access, and in addition the broker handling the s_2 subscription must be trusted with both the *police* and *fire* keys. The filter expression in s_3 does not match events with `location` attributes encrypted under the *fire* key and therefore s_2 covers s_3 strictly. According to the event type hierarchy `BurglaryEvent` is a subtype of `PoliceEvent`, hence subscription s_4 is also covered by s_2, since their filter expressions are the same.

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 secure links.

To support cryptographic properties such as key freshness, and forward and backward secrecy [22], the system requires key management service(s). The most suitable key management strategy depends on the broker-network architecture. For EDSAC21 we assume a stable configuration with static, multi-hop, inter-broker connections and are investigating a tree-based approach [22]. However, the dynamic nature of a peer-to-peer routing layer presents special problems, and we are also evaluating an alternative, ad-hoc network based approach [23].

Efficient group key management [24] 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.

4.4 Security Overheads

When compared with basic publish/subscribe, our secured publish/subscribe introduces three types of processing overhead: *one-time only*, *per event*, and *key management related*. (1) One-time only overheads include node authentication and authorisation when new nodes connect to the network, and subscription-filter encryptions. (2) Per event overheads include those caused by encrypting and decrypting attributes, and applying restriction predicates at local brokers. One encryption is required for each instance of a secure attribute in a published event (see Section 4.3), using the appropriate symmetric key; this happens only once at the source, and intermediate brokers can pass the encrypted event to the next node directly. Decryption is required on delivery, and possibly at each routing step, too. The event dissemination tree structure ensures that each new subscription adds no more than two decryptions: once *en route* at a filtering broker, and once on delivery at the subscriber's local broker. (3) Finally, the cost of key management depends on the frequency of key change and the dissemination method, as discussed above. This is likely to occur relatively infrequently,

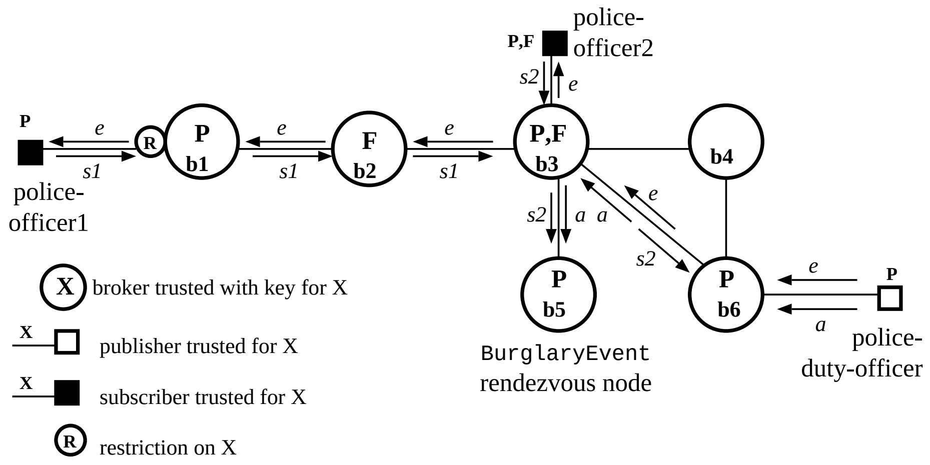

Fig. 4. Notifying two police officers of a BurglaryEvent

as clients never have direct access to encryption keys, and key management is handled at broker-level only.

In addition to processing overheads, attribute encryption increases the size of events in two ways: (1) a single attribute value encrypted with multiple keys results in multiple instances of that value, each encrypted with a different key; (2) encryption algorithm mechanisms dictate that the encrypted data must be at least of some minimum length, depending on the encryption algorithm. Common minimum lengths would be 64 bits and 128 bits. Thus, a single 8 bit attribute value encrypted with three keys grows in size to 192 bits because of padding and multiple attribute instances. This might be avoided by using a stream cipher, which operates on a stream of data one bit at a time, rather than a block cipher.

5 Case Study: Public Services within a City

We now illustrate our architecture for a city in which the publish/subscribe systems of different emergency services interoperate securely and efficiently. We use a break-in to a university building as an example. Fig. 4 shows the principals, brokers and messages discussed below. We assume that equipment failure has left the police network partititioned, and that broker **b1** is connected only through the fire network.

1) We focus on two police officers on night shift; part of their duty is to respond to notifications of burglaries. We assume that the event-type BurglaryEvent is already advertised when the officers come on duty. This means that a rendezvous node **b5** is assigned for the type and subscriptions can be made. We shall see that further advertisements, and subsequent publications, can be made as burglaries are detected in different areas.

We assume that both officers authenticate with their local OASIS service on coming on duty and, assuming that their credentials are valid, acquire the role with associated privilege to send subscription messages: $s1$ and $s2$ respectively.

Officer 1 is a probationary officer, who moves between different parts of the city. Officer 2 is located in West Cambridge. Suppose that at the start of her shift officer 2 subscribes to `BurglaryEvent(location = 'West Cambridge')`. Since this subscription requires filtering on the `location` attribute, and this attribute is encrypted with the *police* key (recall the event type hierarchy shown in Fig. 2), the officer knows that her local broker must be trusted with the *police* key, i.e. a **P** broker.

Officer 1 tries to subscribe to all burglary events with a police code less than 4, `BurglaryEvent(polCode < 4)`, but the request is only partially granted. Instead, the subscription is restricted, as described in Section 4, to deliver only those events that occur in the officer's current location. This restriction, which is based on a dynamically checked environmental constraint, is shown in Fig. 4, attached to his broker connection.

2) Any broker through which $s1$ and/or $s2$ travel (towards their rendezvous node and then along the reverse path of advertisements) will update its internal routing state appropriately. Note that our security architecture augments standard Hermes subscription setup behaviour when we reach broker **b2**. Whilst $s1$ travels through this broker, the broker is not part of the police network, and thus will not have access to the *police* key. Therefore this broker will be forced to degrade routing efficiency by ignoring police officer 1's filter on the `polCode` attribute, which it cannot decrypt, and routing all events forward.

3) We show a duty-officer at a police station who must notify police officers of reported burglaries. Like officers 1 and 2, the duty-officer authenticates himself with his local OASIS service, and acquires privileges to advertise `BurglaryEvents`. Again, his local broker needs access to the *police* key. The consequent advertisement message is shown as a in Fig. 4. This step could occur in parallel with a subscription, see Step 1. If a broker notices that an existing subscription matches a new advertisement, it will resend the subscription message along the reverse path of the new advertisement towards the publisher.

All this occurs at the start of the officers' sessions, a long time (in publish/subscribe terms) before the actual burglary occurs.

4) Now suppose our example burglary is reported to the duty-officer. He publishes an event e, in this case:
`BurglaryEvent(location = 'West Cambridge', premises =`
`'William Gates Building', polCode = 3, ..., zone = 'university')`.

5) The event e leaves the duty-officer's local broker, through the publish/subscribe network, under control of the Hermes routing algorithm. Note that *en route*, each broker decodes and filters the event in so far as it can. In this particular case, only **P** brokers will be able to filter based on the `location` and/or `polCode` attributes, but all brokers will be able to filter on the `zone` attribute (see Fig. 2).

6) As e travels along the reverse path of the subscriptions, it passes through broker **b3**, which is police officer 2's local broker. The broker uses the *police* key to decrypt the `location` and `polCode` attributes before delivering the event to officer 2 over the secure 'final hop' set up as described in Step 1 above.

7) In order to reach police officer 1, e needs to be routed through **b2**. While this is not the most desirable mode of operation, since the event passes through a broker that does not have access to the *police* key, it is crucially better than the situation in which the police network remains partitioned.

As mentioned in Step 2, since broker **b2** cannot decrypt *police* encrypted attributes, it cannot apply filtering on fields such as `location`. Thus for routing e, the event appears as `BurglaryEvent(location = ?, polCode = ?, ..., zone = 'university')`, and it is passed on to **b1** regardless of its `location` value.

8) Finally, the local broker **b1** of police officer 1, which is trusted with the *police* key, will receive e from **b2**, and decrypt the `location` attribute. It will then apply the restriction, checking whether officer 1 is currently in West Cambridge. If so, **b1** will decrypt the entire event and pass it over the secure 'final hop' to officer 1. We have assumed for simplicity that although officer 1 is mobile he remains connected to the same local broker. The alternative is that he creates a new OASIS session whenever he needs to connect to a different broker.

6 Evaluation

The EDSAC21 project is substantial and still at an early stage. We have carried out the following simulation studies to validate the approach. Fig. 5 and Fig. 6 compare the performance of our attribute encryption implementation with the more common approach (such as [4]) that encrypts multiple instances of entire events with each of the relevant keys. Each figure shows the average result of three simulations for each data point.

These experiments all used the Hermes publish/subscribe system for message routing, running over a simulated network topology of 1000 IP routers (organised into ten autonomous subnetworks), with fifty randomly chosen event brokers. In this overlay network we randomly introduced ten event publishers, who, in each iteration of the test, published a total number of 1000 events.

We used the case study scenario of Section 5 as a basis for the simulation, generating events of type PoliceEvent. There were three groups of subscribers: (1) public information services (S0) that filtered only on a single unencrypted attribute (severity); (2) police officers (S1) who filtered on a single *police* encrypted attribute (location); (3) police trainers (S2) who held both *police* and *policeTraining* keys and filtered on isDrill and location. For the attribute encrypted case, publishers encrypted the individual message attributes as shown in Fig. 2. The implementation that encrypts whole events had to send up to three instances of each event, one for each of the independent security domains covered by a message.

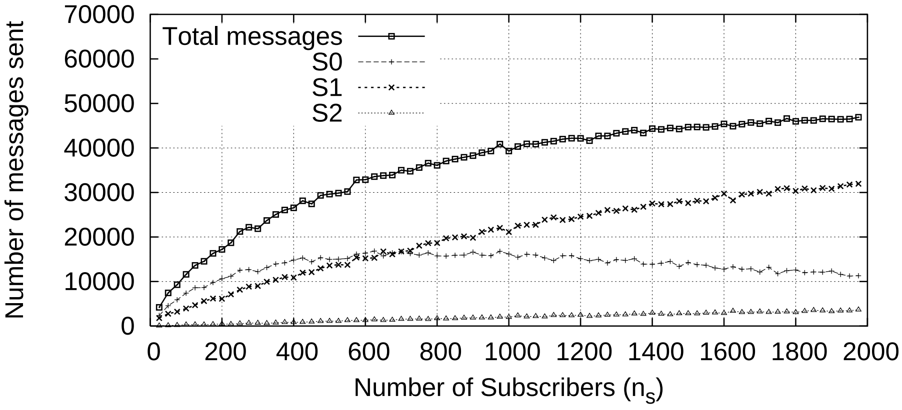

Fig. 5. Total number of messages with attribute encryption

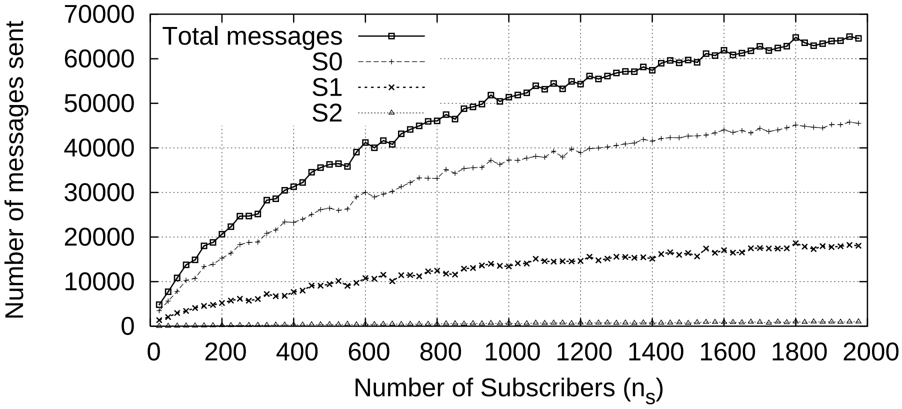

Fig. 6. Total number of messages without attribute encryption

The events were delivered to the subscribers, whose number (n_s) we gradually increased from 25 to 2000 in steps of 25. Subscribers set random filters on event attribute values. Five per cent of all subscriptions were S2 subscriptions that filtered on two encrypted attributes, one encrypted with the *police* key, and the other with the $policeTraining$ key. Thirty five per cent of all subscriptions were S1 subscriptions that filtered on one attribute encrypted with the *police* key, while the rest were S0 subscriptions filtering on a single unencrypted attribute. Note that subscriptions with filtering on encrypted attributes may also include filters on unencrypted attributes. The total number of events sent within the broker network is also shown in the graphs, as Total messages $=$ S0 $+$ S1 $+$ S2.

Our performance results show that with 1000 subscribers, only about 39300 messages needed to be sent when using attribute encryption, while 51400 were sent when events were encrypted atomically with one key at a time – a 24%

saving in bandwidth. For 2000 subscribers, the savings had increased further to 27%.

As the number of subscribers increases, the network with attribute encryption eventually becomes saturated by complex filtering; this is because it becomes increasingly likely that there is a local S1 or S2 subscriber at each broker for any given event. Thus the number of events that need to be decrypted (S1 and S2) grows in Fig. 5, initially because of new event dissemination routes, but later also because events previously counted under S0 now need at least one attribute decrypted; they contribute instead to S1 or S2, which explains the eventual fall-off of the S0 tally as the number of subscribers increases. However, even with 2000 subscribers there were over 11000 event hops for which no attribute decryptions were needed.

Attribute encryption slightly increases the number of times that events need to be decrypted for filtering. However, this is largely compensated by the fact that we then need fewer point-to-point encryptions and decryptions within a TLS connection ($\mathsf{total\ decryptions} = 2 \times \mathsf{S2} + \mathsf{S1}$). For 2000 subscribers, whole event encryption needed about 87400 decryptions, while attribute encryption required a grand total of approximately 88500, – an increase of 1.2%. However, for a less saturated network with 1000 subscriptions, overall encryptions decreased by 2.7%.

Note that generally the overall load on event brokers is decreased still further in our approach, since less data needs to be decrypted at each filtering decryption step (a few attributes, as opposed to the whole event).

7 Conclusions and Future Work

Security is a crucial concern for the development of scalable messaging systems, particularly those for the public sector where data is often highly confidential and privacy must be guaranteed. Publish/Subscribe communication is recognised as appropriate for large-scale systems, yet most research on it excludes security. This paper presents our architecture for a secure publish/subscribe middleware. Our system builds on the performance and fault-tolerance of publish/subscribe messaging, and augments it with scalable security administration based on decentralised Role-Based Access Control. We assume a multi-domain architecture for administration of roles, message types and policies.

Although our implementation uses Hermes and OASIS, our design is applicable to publish/subscribe systems in general. To secure a topic-based publish/subscribe system, whole event encryption would be used, with given events being sent multiple times, encrypted under different keys. Our simulation takes this approach as a basis for comparison. To secure a content-based publish/subscribe system, whole event encryption could be used, but we have shown that it is practicable to encrypt the different attributes of an event separately.

Using an "Active City" example, we show how various public-sector, emergency service notifications can be captured in an event type hierarchy, and how access control and attribute encryption can facilitate secure and efficient communication. If a type hierarchy is not available, our design equally well supports separate services using a shared publish/subscribe system with a flat message type-space.

We have simulated attribute encryption and whole-event encryption for a scenario based on the case study in Section 5. We show that our approach reduces the number of events sent in the system, as well as the processing required for decryptions performed by brokers. Efficiency was not the main focus of our design; rather, we were concerned to demonstrate that the expressiveness of fine-grained access control need not incur undue implementation overhead.

Current and Future Work. This research is part of a project, EDSAC21, to provide secure middleware for large-scale, widely distributed applications. The system mechanisms themselves are used to maintain role membership rules and push changes of policy, thus facilitating immediate response to changes in security predicates.

In [25] we present current work on ensuring the system-wide uniqueness and integrity of message type names and versions, and [26] discusses how a broker network is assembled securely and maintained. We are currently integrating active databases and publish/subscribe. Database message types are defined as described in Section 4.2. Database instances can then advertise the events they are prepared to publish, and subscribers use the standard subscription mechanism [27].

We are also working on how to support communication patterns other than the anonymous multicast of publish/subscribe, while retaining the efficiency and resilience of a broker network. Natural requirements are for an individual member of a role to be selected on publication, and for any recipient to be able to reply to a publication, either anonymously (as in voting) or identified.

We shall continue to assume stationary rather than mobile brokers. Since OASIS is session-based we have so far assumed that mobile clients will remain connected to a single broker during their period of subscription. We envisage natural extensions that allow detached operation while a subscription persists, where a local broker (or a separate service) will buffer messages on behalf of detached clients. Future work is to investigate how best to support client mobility during a period of subscription.

In this paper we have demonstrated the synergy between roles and publish/subscribe communication within and between domains, and have shown the feasibility of expressing and enforcing fine-grained security policy.

Acknowledgements
We acknowledge the contributions of Peter Pietzuch, Brian Shand and András Belokosztolszki. The EDSAC21 project builds on their research as graduate students and they were involved in the design of the architecture presented here. EPSRC GR/T28164 supports Lauri Pesonen and EPSRC GR/S94919 supports David Eyers.

References

1. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Computing Surveys **35** (2003) 114–131
2. Belokosztolszki, A., Eyers, D.M., Pietzuch, P.R., Bacon, J.M., Moody, K.: Role-based access control for publish/subscribe middleware architectures. In: 2nd International Workshop on Distributed Event-Based Systems (DEBS'03). ICDCS, ACM SIGMOD (2003)
3. Baldoni, R., Contenti, M., Virgillito, A.: The evolution of publish/subscribe communication systems. In: Future Directions of Distributed Computing. Volume 2584 of LNCS., Springer (2003) 137–141
4. Yan, Y., Huang, Y., Fox, G., Pallickara, S., Pierce, M., Kaplan, A., Topcu, A.: Implementing a prototype of the security framework for distributed brokering systems. In: Proceedings of the International Conference on Security and Management (SAM'03). (2003) 212–218
5. Wang, C., Carzaniga, A., Evans, D., Wolf, A.: Security issues and requirements in internet-scale publish-subscribe systems. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02), IEEE (2002) 303
6. Miklós, Z.: Towards an access control mechanism for wide-area publish/subscribe systems. In: 1st International Workshop on Distributed Event-Based Systems (DEBS'02). ICDCS, IEEE (2002) 516–524
7. Opyrchal, L., Prakash, A.: Secure distribution of events in content-based publish subscribe systems. In: 10th USENIX Security Symposium. (2001)
8. Dierks, T., Allen, C.: The TLS protocol, version 1.0, RFC-2246. Internet Engineering Task Force (1999)
9. Campbell, R., Al-Muhtadi, J., Naldurg, P., Sampemane, G., Mickunas, M.D.: Towards security and privacy for pervasive computing. In: Software Security – Theories and Systems, Mext-NSF-JSPS International Symposium, ISSS 2002. Volume 2609 of LNCS., Springer (2002) 1–15
10. Beresford, A., Stajano, F.: Location privacy in pervasive computing. IEEE Pervasive Computing **2** (2003) 46–55
11. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems **19** (2001) 332–383
12. Banavar, G., Kaplan, M., Shaw, K., Strom, R.E., Sturman, D.C., Tao, W.: Information flow based event distribution middleware. In: Middleware Workshop at the International Conference on Distributed Computing Systems 1999. (1999)
13. Pietzuch, P.R., Bacon, J.M.: Peer-to-peer overlay broker networks in an event-based middleware. In: 2nd International Workshop on Distributed Event-Based Systems (DEBS'03). ICDCS, ACM SIGMOD (2003)
14. Pietzuch, P.R., Bacon, J.M.: Hermes: A distributed event-based middleware architecture. In: 1st International Workshop on Distributed Event-Based Systems (DEBS'02). ICDCS, IEEE Press (2002) 611–618
15. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In: Middleware '01, IFIP/ACM International Conference on Distributed Systems Platforms. (2001) 329–350
16. Zhao, B.Y., Kubiatowicz, J.D., Joseph, A.D.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley (2001)

17. Sandhu, R., Coyne, E., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer **29** (1996) 38–47
18. Bacon, J., Moody, K., Yao, W.: Access control and trust in the use of widely distributed services. In: Middleware '01, IFIP/ACM International Conference on Distributed Systems Platforms. Volume 2218 of LNCS., Springer (2001) 295–310
19. Bacon, J., Moody, K., Yao, W.: A model of OASIS role-based access control and its support for active security. ACM Transactions on Information and System Security (TISSEC) **5** (2002) 492–540
20. Hombrecher, A.B.: Reconciling Event Taxonomies Across Administrative Domains. PhD thesis, University of Cambridge Computer Laboratory, Cambridge, UK (2002)
21. ITU-T (Telecommunication Standardization Sector, International Telecommunication Union): ITU-T Recommendation X.509: The Directory – Authentication Framework. (2000)
22. Kim, Y., Perrig, A., Tsudik, G.: Tree-based group key agreement. ACM Transactions on Information and System Security **7** (2004) 60–96
23. Hietalahti, M.: Efficient key agreement for ad-hoc networks. Master's thesis, Helsinki University of Technology, Department of Computer Science and Engineering, Espoo, Finland (2001)
24. Rafaeli, S., Hutchison, D.: A survey of key management for secure group communication. ACM Computing Surveys **35** (2003) 309–329
25. Pesonen, L., Bacon, J.: Secure event types in content-based, multi-domain publish/subscribe systems. In: Fifth International Workshop on Software Engineering and Middleware (SEM05). (2005) To appear.
26. Pesonen, L., Eyers, D., Bacon, J.: A capability-based access control architecture for multi-domain publish/subscribe systems. (2006) Submitted for publication.
27. Vargas, L., Bacon, J., Moody, K.: Integrating databases with publish/subscribe. In: 4th International Workshop on Distributed Event-Based Systems (DEBS'05). ICDCS, IEEE Press (2005) 392–397