Encryption-Enforced Access Control in Dynamic Multi-Domain Publish/Subscribe Networks

Lauri I.W. Pesonen
University of Cambridge,
Computer Laboratory
JJ Thomson Avenue,
Cambridge, CB3 0FD, UK
{first.last}@cl.cam.ac.uk

David M. Eyers
University of Cambridge,
Computer Laboratory
JJ Thomson Avenue,
Cambridge, CB3 0FD, UK
{first.last}@cl.cam.ac.uk

Jean Bacon
University of Cambridge,
Computer Laboratory
JJ Thomson Avenue,
Cambridge, CB3 0FD, UK
{first.last}@cl.cam.ac.uk

ABSTRACT

Publish/subscribe systems provide an efficient, event-based, wide-area distributed communications infrastructure. 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. This paper extends 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. Key refresh allows us to ensure forward and backward security when event brokers join and leave the network. We demonstrate that the time and space overheads can be minimised by careful consideration of encryption techniques, and by the use of caching to decrease unnecessary decryptions. We show that our approach has a smaller overall communication overhead than existing approaches for achieving the same degree of control over security in publish/subscribe networks.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Security, Performance

Keywords

Secure publish/subscribe systems, distributed access control, administrative domains, attribute encryption

1. INTRODUCTION

Publish/subscribe is well suited as a communication mechanism for building Internet-scale distributed event-driven applications. Much of its capacity for scale in the number of participants comes from its decoupling of publishers and subscribers by placing an asynchronous event delivery service between them. In truly Internet-scale publish/subscribe systems, the event delivery service will include a large set of interconnected broker nodes spanning a wide geographic (and thus network) area.

However, publish/subscribe systems that do span a wide geographic area are likely to also span multiple administrative domains, be they independent administrative domains inside a single organisation, multiple independent organisations, or a combination of the two.

While the communication capabilities of publish/subscribe systems are well proved, spanning multiple administrative domains is likely to require addressing security considerations. As security and access control are almost the antithesis of decoupling, relatively little publish/subscribe research has focused on security so far.

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.

In [12] Pesonen et al. propose a multi-domain, capability-based access control architecture for publish/subscribe systems. The architecture provides a mechanism for authorising event clients to publish and subscribe to event types. The privileges of the client are checked by the local broker that the client connects to in order to access the publish/subscribe system. The approach implements access control at the edge of the broker network and assumes that all brokers can be trusted to enforce the access control policies correctly. Any malicious, compromised or unauthorised broker is free to read and write any events that pass through it on their way from the publishers 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 in which organisations share a common infrastructure.

We propose enforcing access control within the broker network by encrypting event content, and that policy dictate controls over the necessary encryption keys. With encrypted event content only those brokers that are authorised to access 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 would need to be enforced in a multi-domain publish/subscribe system when multiple organisations form a shared publish/subscribe system yet run 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. 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 along their paths from publishers to subscribers.

There are two particular benefits to sharing the publish/ subscribe infrastructure, both of which relate to the broker network. First, sharing brokers will create a physically larger network that will provide greater geographic reach. Second, increasing the inter-connectivity of brokers will allow the publish/subscribe system to provide 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 that publish information about the movements of vehicles around the London area. We have included Detective Smith as a subscriber in this domain.

Congestion Charge Service Domain. The charges that are levied on the vehicles that have passed through the London Congestion Charge zone each day are issued by systems within this domain. 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.

PITO Domain. The Police Information Technology Organisation (PITO) is the centre from which Police data standards are managed. It is the event type owner in this particular scenario.

Encryption protects the confidentiality of events should they be transported through unauthorised domains. However encrypting whole events means unauthorised brokers cannot make efficient routing decisions.

Our approach is to apply encryption to the individual attributes of events. This way our multi-domain access control policy works at a finer granularity – publishers and subscribers may be authorised access to a subset of the available attributes. In cases where non-encrypted events are used for routing, we can reduce the total number of events sent through the system without revealing the values of sensitive attributes.

In our example scenario, the Congestion Charge Service would only be authorised to read the numberplate field of vehicle sightings – the location attribute would not be decrypted. We thus preserve the privacy of motorists while still allowing the CCS to do its job using the shared publish/subscribe infrastructure.

Let us assume that a Metropolitan Police Service detective is investigating a crime and she is interested in sightings of a specific vehicle. The detective gets a court order that authorises her to subscribe to numberplate events of the specific numberplate related to her case.

Current publish/subscribe access control systems enforce security at the edge of the broker network where clients connect to it. However this approach will often not be acceptable in Internet-scale systems. We propose enforcing security within the broker network as well as at the edges that event clients connect to, by encrypting event content. Publications will be encrypted with their event type specific encryption keys. By controlling access to the encryption keys, we can control access to the event types. The proposed approach allows event brokers to route events even when they have access only to a subset of the potential encryption keys.

We introduce decentralised publish/subscribe systems and relevant cryptography in Section 2. In Section 3 we present our model for encrypting event content on both the event and the attribute level. Section 4 discusses managing encryption keys in multi-domain publish/subscribe systems. We analytically evaluate the performance of our proposal in Section 5. Finally Section 6 discusses related work in securing publish/subscribe systems and Section 7 provides concluding remarks.

2. BACKGROUND

In this section we provide a brief introduction to decentralised publish/subscribe systems. We indicate our assumptions about multi-domain publish/subscribe systems, and describe how these assumptions influence the developments we have made from our previously published work.

2.1 Decentralised Publish/Subscribe Systems

A publish/subscribe system includes publishers, subscribers, and an event service. Publishers publish events, subscribers subscribe to events of interest to them, and the event service is responsible for delivering published events to all subscribers whose interests match the given event.

The event service in a decentralised publish/subscribe system is distributed over a number of broker nodes. Together these brokers form a network that is responsible for maintaining the necessary routing paths from publishers to subscribers. Clients (publishers and subscribers) connect to a local broker, which is fully trusted by the client. In our discussion we refer to the client hosting brokers as *publisher hosting brokers* (PHB) or *subscriber hosting brokers* (SHB) depending on whether the connected client is a publisher or

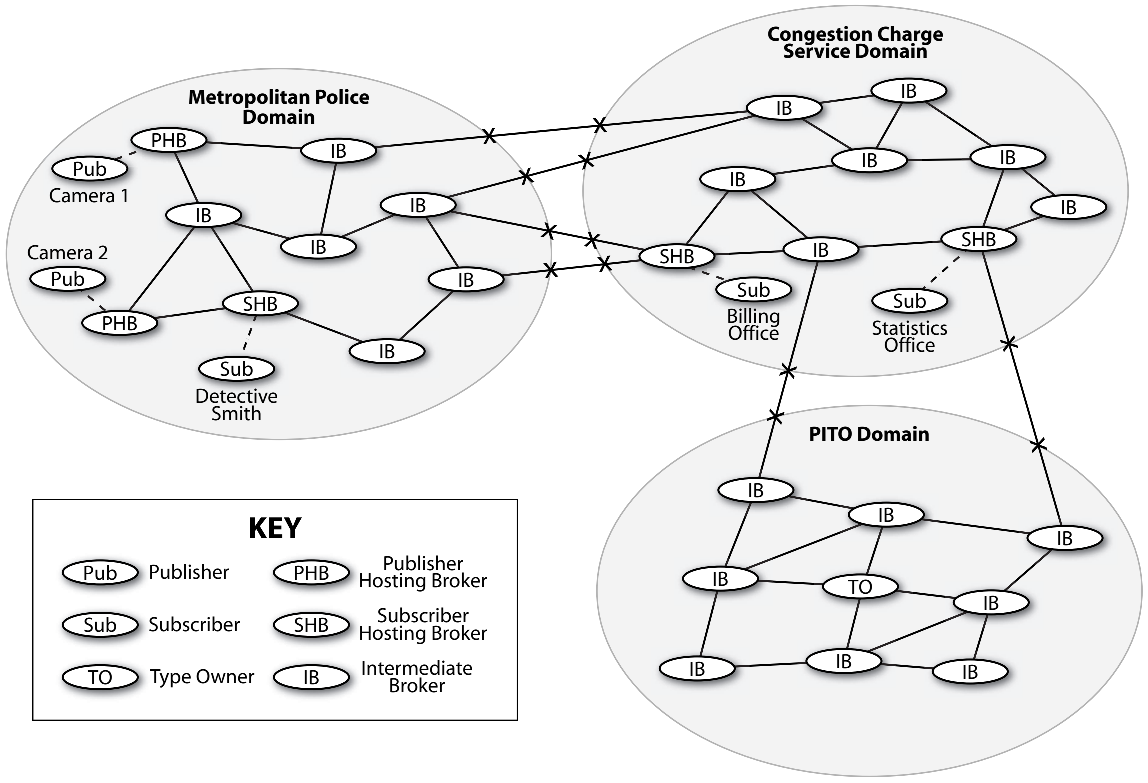

Figure 1: An overall view of our multi-domain publish/subscribe deployment

a subscriber, respectively. A local broker is usually either part of the same domain as the client, or it is owned by a service provider trusted by the client.

A broker network can have a static topology (e.g. Siena [3] and Gryphon [14]) or a dynamic topology (e.g. Scribe [4] and Hermes [13]). Our proposed approach will work in both cases. A static topology enables the system administrator to build trusted domains and in that way improve the efficiency of routing by avoiding unnecessary encryptions (see Sect. 3.4), which is very difficult with a dynamic topology. 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.

Our work is based on the Hermes system. Hermes is a content-based publish/subscribe middleware that includes strong event type support. In other words, each publication is an instance of a particular predefined event type. Publications are type checked at the local broker of each publisher. Our attribute level encryption scheme assumes that events are typed. Hermes uses a structured overlay network as a transport and therefore has a dynamic topology.

A Hermes publication consists of an *event type identifier* and a set of attribute value pairs. The type identifier is the SHA-1 hash of the name of the event type. It is used to route the publication through the event broker network. It conveniently hides the type of the publication, i.e. brokers are prevented from seeing which events are flowing through them unless they are aware of the specific event type name and identifier.

2.2 Secure Event Types

Pesonen et al. introduced *secure event types* in [11], which can have their integrity and authenticity confirmed by checking their digital signatures. A useful side effect of secure event types are their globally unique event type and attribute names. These names can be referred to by access control policies. In this paper we use the secure name of the event type or attribute to refer to the encryption key used to encrypt the event or attribute.

2.3 Capability-Based Access Control

Pesonen et al. proposed a capability-based access control architecture for multi-domain publish/subscribe systems in [12]. The model treats event types as resources that publishers, subscribers, and event brokers want to access. The event type owner is responsible for managing access control for an event type by issuing Simple Public Key Infrastructure (SPKI) authorisation certificates that grant the holder access to the specified event type. For example, authorised publishers will have been issued an authorisation certificate that specifies that the publisher, identified by public key, is authorised to publish instances of the event type specified in the certificate.

We leverage the above mentioned access control mechanism in this paper by controlling access to encryption keys using the same authorisation certificates. That is, a publisher who is authorised to publish a given event type, is also authorised to access the encryption keys used to protect events of that type. We discuss this in more detail in Sect. 4.

2.4 Threat model

The goal of the proposed mechanism is to enforce access control for authorised participants in the system. In our case the first level of access control is applied when the participant tries to join the publish/subscribe network. Unauthorised event brokers are not allowed to join the broker network. Similarly unauthorised event clients are not allowed to connect to an event broker. All the connections in the broker network between event brokers and event clients utilise *Transport Layer Security* (TLS) [5] in order to prevent unauthorised access on the transport layer.

The architecture of the publish/subscribe system means that event clients must connect to event brokers in order to be able to access the publish/subscribe system. Thus we assume that these clients are not a threat. The event client relies completely on the local event broker for access to the broker network. Therefore the event client is unable to access any events without the assistance of the local broker.

The brokers on the other hand are able to analyse all events in the system that pass through them. A broker can analyse both the event traffic as well as the number and names of attributes that are populated in an event (in the case of attribute level encryption). There are viable approaches to preventing traffic analysis by inserting random events into the event stream in order to produce a uniform traffic pattern. Similarly attribute content can be padded to a standard length in order to avoid leaking information to the adversary.

While traffic analysis is an important concern we have not addressed it further in this paper.

3. ENCRYPTING EVENT CONTENT

We propose enforcing access control in a decentralised broker network by encrypting the contents of published events and controlling access to the encryption keys. Effectively we move the responsibility for access control from the broker network to the key managers.

It is assumed that all clients have access to a broker that they can trust and that is authorised to access the event content required by the client. This allows us to implement the event content encryption within the broker network without involving the clients. By delegating the encryption tasks to the brokers, we lower the number of nodes required to have access to a given encryption key[1]. The benefits are three-fold: i) fewer nodes handle the confidential encryption key so there is a smaller chance of the key being disclosed; ii) key refreshes involve fewer nodes which means that the key management algorithm will incur smaller communication and processing overheads to the publish/subscribe system; and iii) the local broker will decrypt an event once and deliver it to all subscribers, instead of each subscriber having to decrypt the same event. Delegating encryption tasks to the local broker is appropriate, because encryption is a middleware feature used to enforce access control within the middleware system. If applications need to handle encrypted data in the application layer, they are free to publish encrypted data over the publish/subscribe system.

We can implement encryption either at the event level or the attribute level. Event encryption is simpler, requires fewer keys, fewer independent cryptographic operations, and thus is usually faster. Attribute encryption enables access control at the attribute level, which means that we have a more expressive and powerful access control mechanism, while usually incurring a larger performance penalty.

In this section we discuss encrypting event content both at the event level and the attribute level; avoiding leaking information to unauthorised brokers by encrypting subscription filters; avoiding unnecessary encryptions between authorised brokers; and finally, how event content encryption was implemented in our prototype. Note that since no publish/subscribe client is ever given access to encryption keys, any encryption performed by the brokers is necessarily completely transparent to all clients.

3.1 Event Encryption

In event encryption all the event attributes are encrypted as a single block of plaintext. The event type identifier is left intact (i.e. in plaintext) in order to facilitate event routing in the broker network.

The globally unique event type identifier specifies the encryption key used to encrypt the event content. Each event type in the system will have its own individual encryption key. Keys are refreshed, as discussed in Sect. 4.2.

While in transit the event will consist of a tuple containing the type identifier, a publication timestamp, ciphertext, and a message authentication tag: `<type_id, timestamp, cipher_text, authentication_tag>`.

Event brokers that are authorised to access the event, and thus have access to the encryption key, can decrypt the event and implement content-based routing. Event brokers that do not have access to the encryption key will be forced to route the event based only on its type. That is, they will not be able to make intelligent decisions about whether events need not be transmitted down their outgoing links.

Event encryption results in one encryption at the publisher hosting broker, and one decryption at each filtering intermediate broker and subscriber hosting broker that the event passes through, regardless of the number of attributes. This results in a significant performance advantage compared to attribute encryption.

3.2 Attribute Encryption

In attribute encryption each attribute value in an event is encrypted separately with its own encryption key. The encryption key is identified by the attribute's globally unique identifier (the globally unique event identifier defines a namespace inside which the attribute identifier is a fully qualified name).

[1]The encryption keys are changed over time in response to brokers joining or leaving the network, and periodically to reduce the amount of time any single key is used. This is discussed in Sect. 4.2

The event type identifier is left intact to facilitate event routing for unauthorised brokers. The attribute identifiers are also left intact to allow authorised brokers to decrypt the attribute values with the correct keys. Brokers that are authorised to access some of the attributes in an event, can implement content-based routing over the attributes that are accessible to them.

An attribute encrypted event in transit consists of the event type identifier, a publication timestamp, and a set of attribute tuples: `<type_id, timestamp, attributes>`. Attribute tuples consist of an attribute identifier, ciphertext, and a message authentication tag: `<attr_id, ciphertext, authentication_tag>`. The attribute identifier is the SHA-1 hash of the attribute name used in the event type definition. Using the attribute identifier in the published event instead of the attribute name prevents unauthorised parties from learning which attributes are included in the publication.

Compared with event encryption, attribute encryption usually results in larger processing overheads, because each attribute is encrypted separately. In the encryption process the initialisation of the encryption algorithm takes a significant portion of the total running time of the algorithm. Once the algorithm is initialised, increasing the amount of data to be encrypted does not affect the running time very much. This disparity is emphasised in attribute encryption, where an encryption algorithm must be initialised for each attribute separately, and the amount of data encrypted is relatively small. As a result attribute encryption incurs larger processing overheads when compared with event encryption which can be clearly seen from the performance results in Sect. 5.

The advantage of attribute encryption is that the type owner is able to control access to the event type at the attribute level. The event type owner can therefore allow clients to have different levels of access to the same event type. Also, attribute level encryption enables content-based routing in cases where an intermediate broker has access only to some of the attributes of the event, thus reducing the overall impact of event delivery on the broker network. Therefore the choice between event and attribute encryption is a trade-off between expressiveness and performance, and depends on the requirements of the distributed application.

The expressiveness provided by attribute encryption can be emulated by introducing a new event type for each group of subscribers with the same authorisation. The publisher would then publish an instance of each of these types instead of publishing just a combined event. For example, in our London police network, the congestion control cameras would have to publish one event for the CCS and another for the detective. This approach could become difficult to manage if the attributes have a variety of security properties, since a large number of event types would be required and policies and subscriptions may change dynamically. This approach creates a large number of extra events that must be routed through the network, as is shown in Sect. 5.3.

3.3 Encrypting Subscriptions

In order to fully protect the confidentiality of event content we must also encrypt subscriptions. Encrypted subscriptions guarantee: i) that only authorised brokers are able to submit subscriptions to the broker network, and ii) that unauthorised brokers do not gain information about event content by monitoring which subscriptions a given event matches. For example, in the first case an unauthorised broker can create subscriptions with appropriately chosen filters, route them towards the root of the event dissemination tree, and monitor which events were delivered to it as matching the subscription. The fact that the event matched the subscription would leak information to the broker about the event content even if the event was still encrypted. In the second case, even if an unauthorised broker was unable to create subscriptions itself, it could still look at subscriptions that were routed through it, take note of the filters on those subscriptions, and monitor which events are delivered to it by upstream brokers as matching the subscription filters. This would again reveal information about the event content to the unauthorised broker.

In the case of encrypting complete events, we also encrypt the complete subscription filter. The event type identifier in the subscription must be left intact to allow brokers to route events based on their topic when they are not authorised to access the filter. In such cases the unauthorised broker is required to assume that events of such a type match all filter expressions.

Each attribute filter is encrypted individually, much as when encrypting a publication. In addition to the event type identifier the attribute identifiers are also left intact to allow authorised brokers to decrypt those filters that they have access to, and route the event based on its matching the decrypted filters.

3.4 Avoiding Unnecessary Cryptographic Operations

Encrypting the event content is not necessary if the current broker and the next broker down the event dissemination tree have the same credentials with respect to the event type at hand. For example, one can assume that all brokers inside an organisation would share the same credentials and therefore, as long as the next broker is a member of the same domain, the event can be routed to it in plaintext. With attribute encryption it is possible that the neighbouring broker is authorised to access a subset of the decrypted attributes, in which case those attributes that the broker is not authorised to access would be passed to it encrypted.

In order to know when it is safe to pass the event in plaintext form, the brokers exchange credentials as part of a handshake when they connect to each other. In cases when the brokers are able to verify each others' credentials, they will add them to the routing table for future reference. If a broker acquires new credentials after the initial handshake, it will present these new credentials to its neighbours while in session.

Regardless of its neighbouring brokers, the PHB will always encrypt the event content, because it is cheaper to encrypt the event once at the root of the event dissemination tree. In Hermes the rendezvous node for each event type is selected uniformly randomly (the event type name is hashed with the SHA-1 hash algorithm to produce the event type

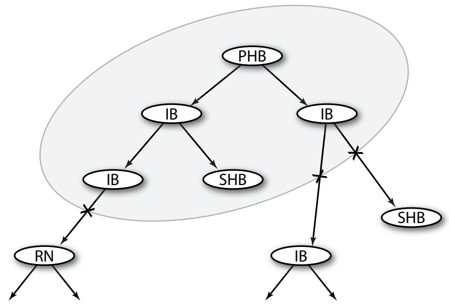

Figure 2: Node addressing is evenly distributed across the network, thus rendezvous nodes may lie outside the domain that owns an event type

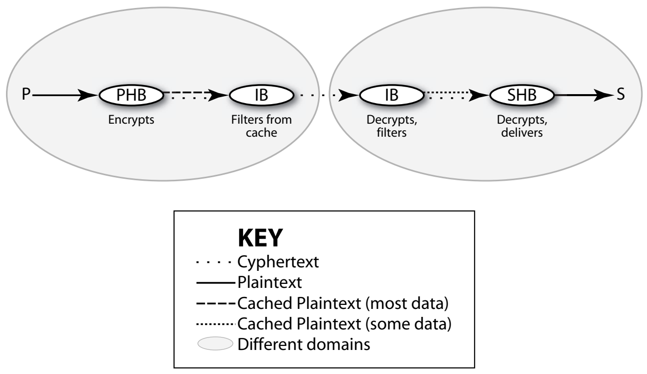

Figure 3: Caching decrypted data to increase efficiency when delivering to peers with equivalent security privileges

identifier, then the identifier is used to select the rendezvous node in the structured overlay network). Therefore it is probable that the rendezvous node will reside outside the current domain. This situation is illustrated in the event dissemination tree in Fig. 2. So even with domain internal applications, where the event can be routed from the publisher to all subscribers in plaintext form, the event content will in most cases have to be encrypted for it to be routed to the rendezvous node.

To avoid unnecessary decryptions, we attach a plaintext content cache to encrypted events. A broker fills the cache with content that it has decrypted, for example, in order to filter on the content. The cache is accessed by the broker when it delivers an event to a local subscriber after first seeing if the event matches the subscription filter, but the broker also sends the cache to the next broker with the encrypted event. The next broker can look the attribute up from the cache instead of having to decrypt it. If the event is being sent to an unauthorised broker, the cache will be discarded before the event is sent. Obviously sending the cache with the encrypted event will add to the communication cost, but this is outweighed by the saving in encryption/decryption processing. In Fig. 3 we see two separate cached plaintext streams accompanying an event depending on the inter-broker relationships in two different domains.

We show in Sect. 5.2 that the overhead of sending encrypted messages with a full plaintext cache incurs almost no overhead compared to sending plaintext messages.

3.5 Implementation

In our implementation we have used the EAX mode [2] of operation when encrypting events, attributes, and subscription filters. EAX is a mode of operation for block ciphers, also called an *Authenticated Encryption with Associated Data* (AEAD) algorithm that provides simultaneously both data confidentiality and integrity protection. The algorithm implements a two-pass scheme where during the first pass the plain text is encrypted, and on the second pass a *message authentication code* (MAC) is generated for the encrypted data.

The EAX mode is compatible with any block cipher. We decided to use the *Advanced Encryption Standard* (AES) [9] algorithm in our implementation, because of its standard status and the fact that the algorithm has gone through thorough cryptanalysis during its existence and no serious vulnerabilities have been found thus far.

In addition to providing both confidentiality and integrity protection, the EAX mode uses the underlying block cipher in *counter mode* (CTR mode) [21]. A block cipher in counter mode is used to produce a stream of key bits that are then XORed with the plaintext. Effectively CTR mode transforms a block cipher into a stream cipher. The advantage of stream ciphers is that the ciphertext is the same length as the plaintext, whereas with block ciphers the plaintext must be padded to a multiple of the block cipher's block length (e.g. the AES block size is 128 bits). Avoiding padding is very important in attribute encryption, because the padding might increase the size of the attribute disproportionally. For example, a single integer might be 32 bits in length, which would be padded to 128 bits if we used a block cipher. With event encryption the message expansion is not that relevant, since the length of padding required to reach the next 16 byte multiple will probably be a small proportion of the overall plaintext length.

In encryption mode the EAX algorithm takes as input a *nonce* (a number used once), an encryption key and the plaintext, and it returns the ciphertext and an authentication tag. In decryption mode the algorithm takes as input the encryption key, the ciphertext and the authentication tag, and it returns either the plaintext, or an error if the authentication check failed.

The nonce is expanded to the block length of the underlying block cipher by passing it through an OMAC construct (see [7]). It is important that particular nonce values are not reused, otherwise the block cipher in CTR mode would produce an identical key stream. In our implementation we used the PHB defined event timestamp (64-bit value counting the milliseconds since January 1, 1970 UTC) appended by the PHB's identity (i.e. public key) as the nonce. The broker is responsible for ensuring that the timestamps increase monotonically.

The authentication tag is appended to the produced cipher text to create a two-tuple. With event encryption a single tag is created for the encrypted event. With attribute encryption each attribute is encrypted and authenticated separately, and they all have their individual tags. The tag length is configurable in EAX without restrictions, which allows the user to make a trade-off between the authenticity guarantees provided by EAX and the added communication overhead. We used a tag length of 16 bytes in our implementation, but one could make the tag length a publisher/subscriber defined parameter for each publication/subscription or include it in the event type definition to make it a type specific parameter.

EAX also supports including unencrypted *associated data* in the tag calculation. The integrity of this data is protected, but it is still readable by everyone. This feature could be used with event encryption in cases where some of the event content is public and thus would be useful for content-based routing. The integrity of the data would still be protected against changes, but unauthorised brokers would be able to apply filters. We have included the event type identifier as associated data in order to protect its integrity.

Other AEAD algorithms include the *offset codebook mode* (OCB) [17] and the *counter with CBC-MAC mode* (CCM) [22]. Contrarily to the EAX mode the OCB mode requires only one pass over the plaintext, which makes it roughly twice as fast as EAX. Unfortunately the OCB mode has a patent application in place in the USA, which restricts its use. The CCM mode is the predecessor of the EAX mode. It was developed in order to provide a free alternative to OCB. The EAX was developed later to address some issues with CCM [18]. Similarly to EAX, CCM is also a two-pass mode.

4. KEY MANAGEMENT

In both encryption approaches the encrypted event content has a globally unique identifier (i.e. the event type or the attribute identifier). That identifier is used to determine the encryption key to use when encrypting or decrypting the content. Each event type, in event encryption, and attribute, in attribute encryption, has its own individual encryption key. By controlling access to the encryption key we effectively control access to the encrypted event content.

In order to control access to the encryption keys we form a *key group* of brokers for each individual encryption key. The key group is used to refresh the key when necessary and to deliver the new key to all current members of the key group. The key group manager is responsible for verifying that a new member requesting to join the key group is authorised to do so. Therefore the key group manager must be trusted by the type owner to enforce the access control policy. We assume that the key group manager is either a trusted third party or alternatively a member of the type owner's domain.

In [12] Pesonen et al. proposed a capability-based access control architecture for multi-domain publish/subscribe systems. The approach uses capabilities to decentralise the access control policy amongst the publish/subscribe nodes (i.e. clients and brokers): each node holds a set of capabilities that define the authority granted to that node. Authority to access a given event type is granted by the owner of that type issuing a capability to a node. The capability defines the event type, the action, and the attributes that

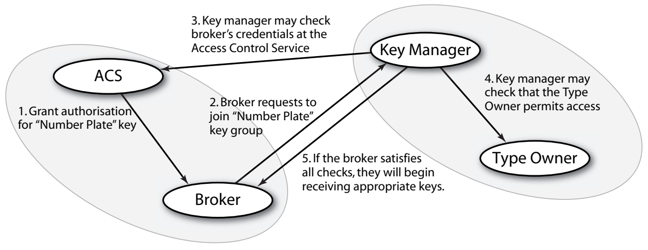
3. Key manager may check broker's credentials at the Access Control Service
Key Manager
ACS
4. Key manager may check that the Type Owner permits access
2. Broker requests to join "Number Plate" key group
1. Grant authorisation for "Number Plate" key
5. If the broker satisfies all checks, they will begin receiving appropriate keys.
Type Owner
Broker

Figure 4: The steps involved for a broker to be successful in joining a key group

the node is authorised to access. For example, a tuple `<NP, subscribe, *>` would authorise the owner to subscribe to *Numberplate* events with access to all attributes in the published events. The sequence of events required for a broker to successfully join a key group is shown in Fig. 4.

Both the client hosting broker and the client must be authorised to make the client's request. That is, if the client makes a subscription request for *Numberplate* events, both the client and the local broker must be authorised to subscribe to *Numberplate* events. This is because from the perspective of the broker network, the local broker acts as a proxy for the client.

We use the same capabilities to authorise membership in a key group that are used to authorise publish/subscribe requests. Not doing so could lead to the inconsistent situation where a SHB is authorised to make a subscription on behalf of its clients, but is not able to decrypt incoming event content for them. In the *Numberplate* example above, the local broker holding the above capability is authorised to join the *Numberplate* key group as well as the key groups for all the attributes in the *Numberplate* event type.

4.1 Secure Group Communication

Event content encryption in a decentralised multi-domain 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 clients, clients 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 are a number of scalable key management protocols for secure group communication [15]. We have implemented the *One-Way Function Tree* (OFT) [8] protocol as a proof of concept. We chose to implement OFT, because of its relatively simplicity and good performance. 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.

4.2 Key Refreshing

Traditionally in group key management schemes the encryption key is refreshed when a new member joins the group, an existing member leaves the group, or a timer expires. Refreshing the key when a new member joins provides backward secrecy, i.e. the new member is prevented from accessing old messages. Similarly refreshing the key when an existing member leaves provides forward secrecy, i.e. the old member is prevented from accessing future messages. Timer triggered refreshes are issued periodically in order to limit the damage caused by the current key being compromised.

Even though the state-of-the-art key management protocols are efficient, refreshing the key unnecessarily introduces extra traffic and processing amongst the key group members. In our case key group membership is based on the broker holding a capability that authorises it to join the key group. The capability has a set of validity conditions that in their simplest form define a time period when the certificate is valid, and in more complex cases involve on-line checks back towards the issuer. In order to avoid unnecessary key refreshes the key manager looks at the certificate validity conditions of the joining or leaving member. In case of a joining member, if the manager can ascertain that the certificate was valid at the time of the previous key refresh, a new key refresh can be avoided. Similarly, instead of refreshing the key immediately when a member leaves the key group, the key manager can cache their credentials and refresh the key only when the credentials expire. These situations are both illustrated in Fig.5. It can be assumed that the credentials granted to brokers are relatively static, i.e. once a domain is authorised to access an event type, the authority will be delegated to all brokers of that domain, and they will have the authority for the foreseeable future. More fine grained and dynamic access control would be implemented at the edge of the broker network between the clients and the client hosting brokers.

When an encryption key is refreshed the new key is tagged with a timestamp. The encryption key to use for a given event is selected based on the event's publication timestamp. The old keys will be kept for a reasonable amount of time in order to allow for some clock drift. Setting this value is part of the key management protocol, although exactly how long this time should be will depend on the nature of the application and possibly the size of the network. It can be configured independently per key group if necessary.

5. EVALUATION

In order to evaluate the performance of event content encryption we have implemented both encryption approaches running over our implementation of the Hermes publish/subscribe middleware. The implementation supports three modes: plaintext content, event encryption, and attribute encryption, in a single publish/subscribe system.

We ran three performance tests in a discrete event simulator. The simulator was run on an Intel P4 3.2GHz workstation with 1GB of main memory. We decided to run the tests on an event simulator instead of an actual deployed system in order to be able to measure to aggregate time it takes to handle all messages in the system.

The following sections describe the specific test setups and the results in more detail.

5.1 End-to-End Overhead

The end-to-end overhead test shows how much the overall message throughput of the simulator was affected by event content encryption. We formed a broker network with two brokers, attached a publisher to one of them and a subscriber to the other one. The subscriber subscribed to the advertised event type without any filters, i.e. each publication matched the subscriber's publication and thus was delivered to the subscriber. The test measures the combined time it takes to publish and deliver 100,000 events. If the content is encrypted this includes both encrypting the content at the PHB and decrypting it at the SHB.

In the test the number of attributes in the event type is increased from 1 to 25 (the x-axis). Each attribute is set to a 30 character string. For each number of attributes in the event type the publisher publishes 100,000 events, and the elapsed time is measured to derive the message throughput. The test was repeated five times for each number of attributes and we use the average of all iterations in the graph, but the results were highly consistent so the standard deviation is not shown. The same tests were run with no content encryption, event encryption, and attribute encryption.

As can be seen in Fig. 6, event content encryption introduces a large overhead compared to not using encryption. The throughput when using attribute encryption with an event type with one attribute is 46% of the throughput achieved when events are sent in plaintext. When the number of attributes increases the performance gap increases as well: with ten attributes the performance with attribute encryption has decreased to 11.7% of plaintext performance.

Event encryption fares better, because of fewer encryption operations. The increase in the amount of encrypted data does not affect the performance as much as the number of individual encryption operations does. The difference in performance with event encryption and attribute encryption with only one attribute is caused by the Java object serialisation mechanism: in the event encryption case the whole attribute structure is serialised, which results in more objects than serialising a single attribute value. A more efficient implementation would provide its own marshalling mechanism.

Note that the EAX implementation we use runs the nonce (i.e. initialisation vector) through an OMAC construct to increase its randomness. Since the nonce is not required to be kept secret (just unique), there is a potential time/space trade-off we have not yet investigated in attaching extra nonce attributes that have already had this OMAC construct applied to them.

5.2 Domain Internal Events

We explained in Sect. 3.4 that event content decryption and encryption can be avoided if both brokers are authorised to access the event content. This test was designed to show that the use of the encrypted event content mechanism between two authorised brokers incurs only a small performance overhead.

In this test we again form a broker network with two brokers.

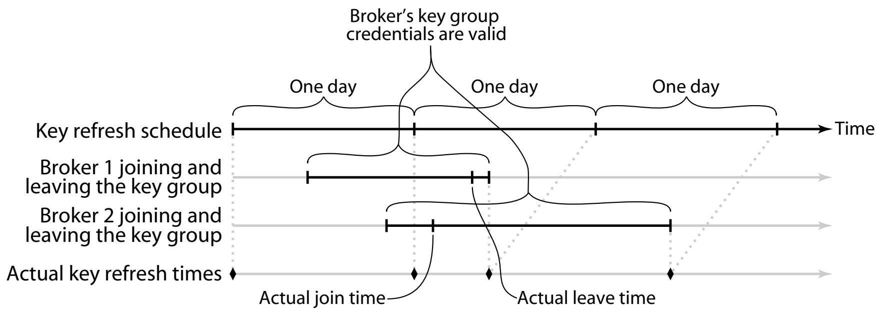
Broker's key group credentials are valid
One day
One day
One day
Key refresh schedule
Time
Broker 1 joining and leaving the key group
Broker 2 joining and leaving the key group
Actual key refresh times
Actual join time
Actual leave time

Figure 5: How the key refresh schedule is affected by brokers joining and leaving key groups

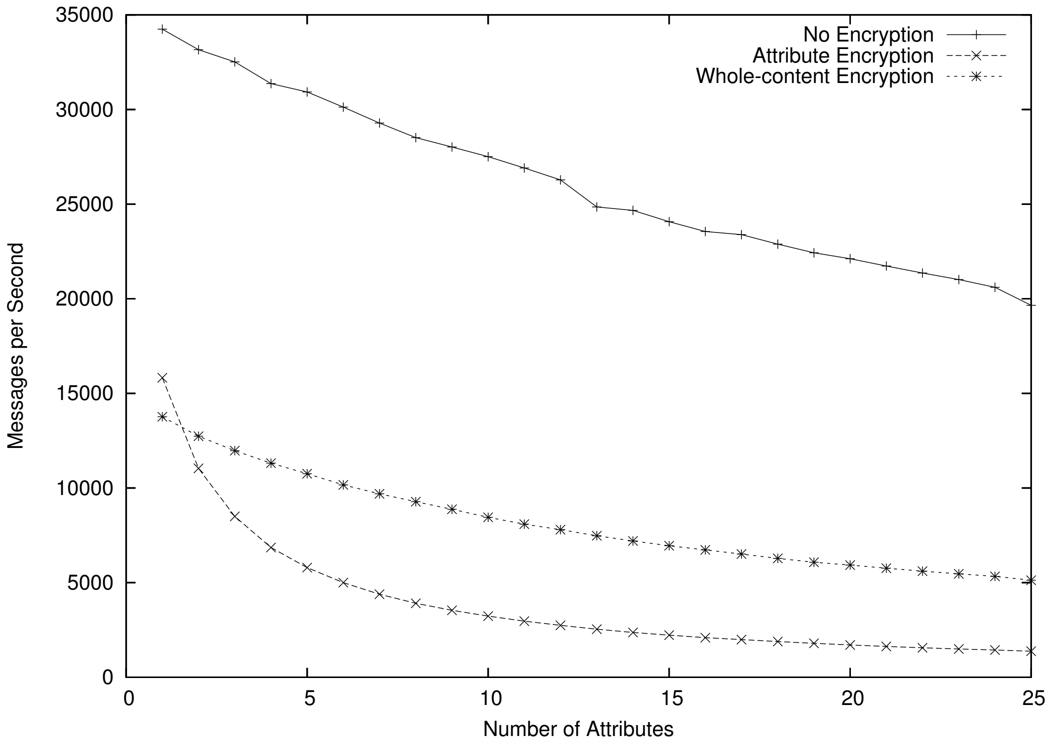

Figure 6: Throughput of Events in a Simulator

Both brokers are configured with the same credentials. The publisher is attached to one of the brokers and the subscriber to the other, and again the subscriber does not specify any filters in its subscription.

The publisher publishes 100,000 events and the test measures the elapsed time in order to derive the system's message throughput. The event content is encrypted outside the timing measurement, i.e. the encryption cost is not included in the measurements. The goal is to model an environment where a broker has received a message from another authorised broker, and it routes the event to a third authorised broker. In this scenario the middle broker does not need to decrypt nor encrypt the event content.

As shown in Fig. 2, the elapsed time was measured as the number of attributes in the published event was increased from 1 to 25. The attribute values in each case are 30 character strings. Each test is repeated five times, and we use the average of all iterations in the graph. The same test was then repeated with no encryption, event encryption and attribute encryption turned on.

The encrypted modes follow each other very closely. Predictably, the plaintext mode performs a little better for all attribute counts. The difference can be explained partially by the encrypted events being larger in size, because they include both the plaintext and the encrypted content in this test. The difference in performance is 3.7% with one attribute and 2.5% with 25 attributes.

We believe that the roughness of the graphs can be explained by the Java garbage collector interfering with the simulation. The fact that all three graphs show the same irregularities supports this theory.

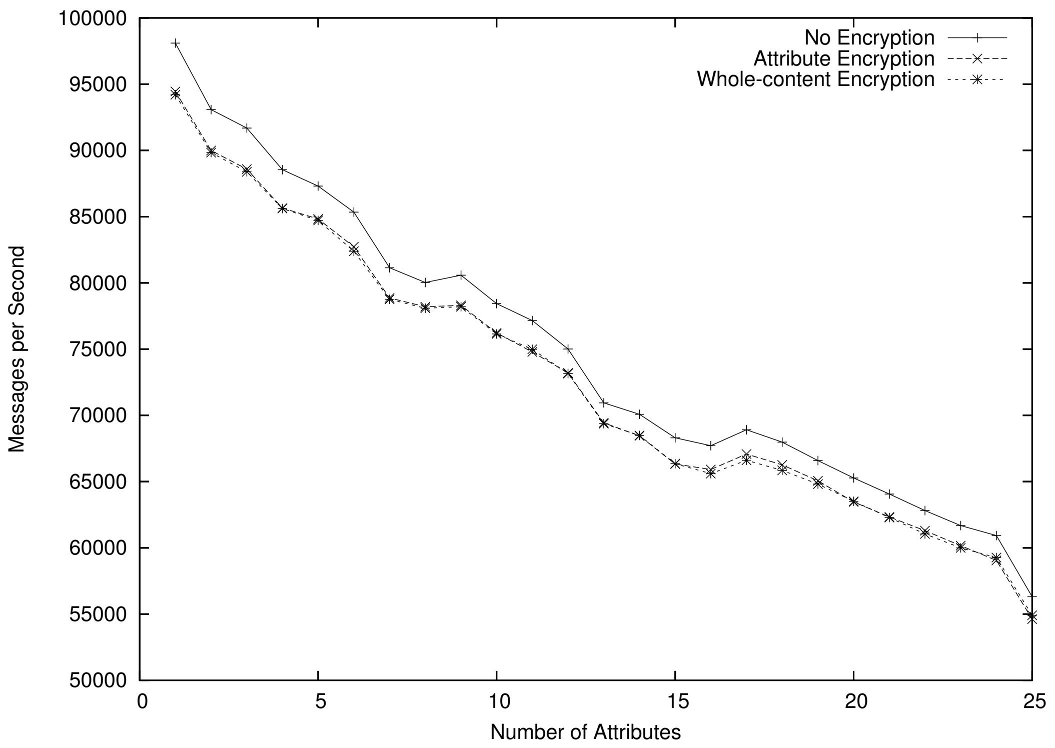

Figure 7: Throughput of Domain Internal Events

5.3 Communication Overhead

Through the definition of multiple event types, it is possible to emulate the expressiveness of attribute encryption using only event content encryption. The last test we ran was to show the communication overhead caused by this emulation technique, compared to using real attribute encryption.

In the test we form a broker network of 2000 brokers. We attach one publisher to one of the brokers, and an increasing number of subscribers to the remaining brokers. Each subscriber simulates a group of subscribers that all have the same access rights to the published event. Each subscriber group has its own event type in the test.

The outcome of this test is shown in Fig. 8. The number of subscriber groups is increased from 1 to 50 (the x-axis). For each n subscriber groups the publisher publishes one event to represent the use of attribute encryption and n events representing the events for each subscriber group. We count the number of hops each publication makes through the broker network (y-axis).

Note that Fig. 8 shows workloads beyond what we would expect in common usage, in which many event types are likely to contain fewer than ten attributes. The subscriber groups used in this test represent disjoint permission sets over such event attributes. The number of these sets can be determined from the particular access control policy in use, but will be a value less than or equal to the factorial of the number of attributes in a given event type.

The graphs indicate that attribute encryption performs better than event encryption even for small numbers of subscriber groups. Indeed, with only two subscriber groups (e.g. the case with *Numberplate* events) the hop count increases from 7.2 hops for attribute encryption to 16.6 hops for event encryption. With 10 subscriber groups the corresponding numbers are 24.2 and 251.0, i.e. an order of magnitude difference.

6. RELATED WORK

Wang et al. have categorised the various security issues that need to be addressed in publish/subscribe systems in the future in [20]. The paper is a comprehensive overview of security issues in publish/subscribe systems and as such tries to draw attention to the issues rather than providing solutions.

Bacon et al. in [1] examine the use of role-based access control in multi-domain, distributed publish/subscribe systems. Their work is complementary to this paper: distributed RBAC is one potential policy formalism that might use the enforcement mechanisms we have presented.

Opyrchal and Prakash address the problem of event confidentiality at the last link between the subscriber and the SHB in [10]. They correctly state that a secure group communication approach is infeasible in an environment like publish/subscribe that has highly dynamic group memberships. As a solution 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 SHB to a set of matching subscribers. We assume in our work that the SHB is powerful enough to man-

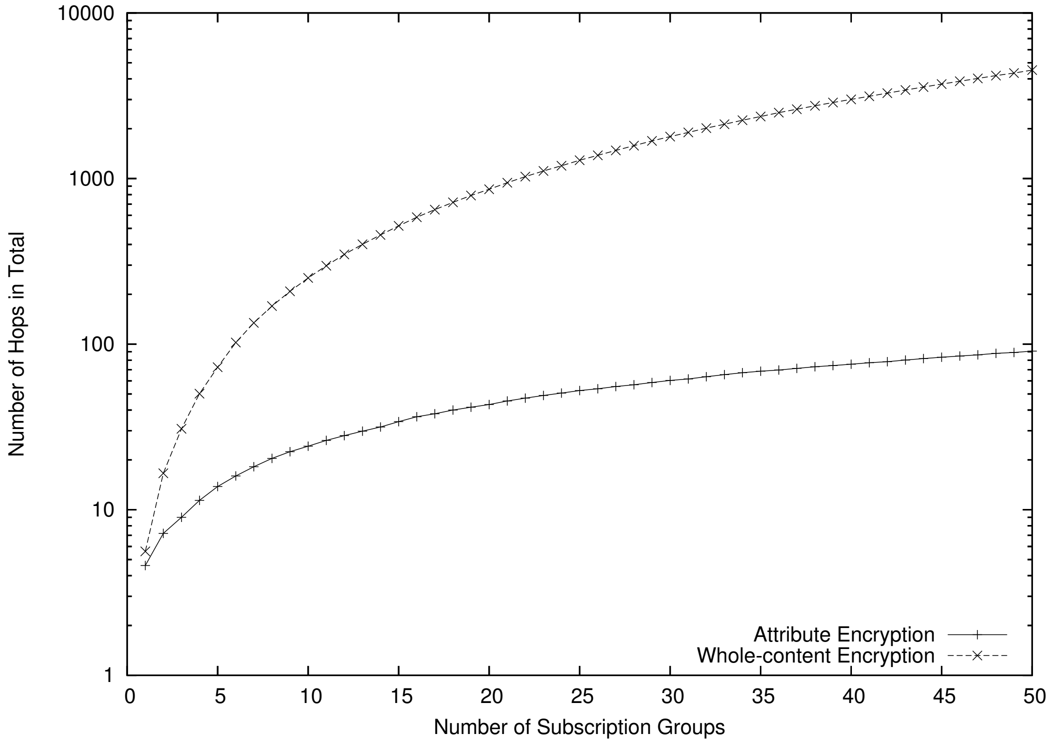

Figure 8: Hop Counts When Emulating Attribute Encryption

age a TLS secured connection for each local subscriber.

Both Srivatsa et al. [19] and Raiciu et al. [16] present mechanisms for protecting the confidentiality of messages in decentralised publish/subscribe infrastructures. Compared to our work both papers aim to provide the means for protecting the integrity and confidentiality of messages whereas the goal for our work is to enforce access control inside the broker network. Raiciu et al. assume in their work that none of the brokers in the network are trusted and therefore all events are encrypted from publisher to subscriber and that all matching is based on encrypted events. In contrast, we assume that some of the brokers on the path of a publication are trusted to access that publication and are therefore able to implement event matching. We also assume that the publisher and subscriber hosting brokers are always trusted to access the publication. The contributions of Srivatsa et al. and Raiciu et al. are complementary to the contributions in this paper.

Finally, Fiege et al. address the related topic of event visibility in [6]. While the work concentrated on using scopes as mechanism for structuring large-scale event-based systems, the notion of event visibility does resonate with access control to some extent.

7. CONCLUSIONS

Event content encryption can be used to enforce an access control policy while events are in transit in the broker network of a multi-domain publish/subscribe system. Encryption causes an overhead, but i) there may be no alternative when access control is required, and ii) the performance penalty can be lessened with implementation optimisations, such as passing cached plaintext content alongside encrypted content between brokers with identical security credentials. This is particularly appropriate if broker-to-broker connections are secured by default so that wire-sniffing is not an issue.

Attribute level encryption can be implemented in order to enforce fine-grained access control policies. In addition to providing attribute-level access control, attribute encryption enables partially authorised brokers to implement content-based routing based on the attributes that are accessible to them.

Our experiments show that i) by caching plaintext and ciphertext content when possible, we are able to deliver comparable performance to plaintext events, and ii) that attribute encryption within an event incurs far less overhead than defining separate event types for the attributes that need different levels of protection.

In environments comprising multiple domains, where event-brokers have different security credentials, we have quantified how a trade-off can be made between performance and expressiveness.

Acknowledgements

We would like to thank the anonymous reviewers for their very helpful comments. Lauri Pesonen is supported by EPSRC (GR/T28164) and the Nokia Foundation. David Eyers is supported by EPSRC (GR/S94919).

8. REFERENCES

[1] J. Bacon, D. M. Eyers, K. Moody, and L. I. W. Pesonen. Securing publish/subscribe for multi-domain systems. In G. Alonso, editor, *Middleware*, volume 3790 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2005.

[2] M. Bellare, P. Rogaway, and D. Wagner. Eax: A conventional authenticated-encryption mode. Cryptology ePrint Archive, Report 2003/069, 2003. `http://eprint.iacr.org/`.

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

[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)*, 20(8):1489–1499, Oct. 2002.

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

[6] L. Fiege, M. Mezini, G. M uhl, and A. P. Buchmann. Engineering event-based systems with scopes. In *ECOOP '02: Proceedings of the 16th European Conference on Object-Oriented Programming*, pages 309–333, London, UK, 2002. Springer-Verlag.

[7] T. Iwata and I. A. Iurosawa. OMAC: One-key CBC MAC, Jan. 14 2002.

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

[9] National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). Federal Information Processing Standards Publication (FIPS PUB) 197, Nov. 2001.

[10] 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.

[11] 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*, pages 98–105, New York, NY, USA, Sept. 2005. ACM Press.

[12] 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)*, pages 222–228, Phoenix, AZ, Jan. 2006. IEEE.

[13] 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)*, pages 611–618, Vienna, Austria, July 2002. IEEE.

[14] P. R. Pietzuch and S. Bhola. Congestion control in a reliable scalable message-oriented middleware. In M. Endler and D. Schmidt, editors, *Proc. of the 4th Int. Conf. on Middleware (Middleware '03)*, pages 202–221, Rio de Janeiro, Brazil, June 2003. Springer.

[15] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. *ACM Computing Surveys*, 35(3):309–329, 2003.

[16] C. Raiciu and D. S. Rosenblum. Enabling confidentiality in content-based publish/subscribe infrastructures. In *Securecomm '06: Proceedings of the Second IEEE/CreatNet International Conference on Security and Privacy in Communication Networks*, 2006.

[17] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205, 2001.

[18] P. Rogaway and D. Wagner. A critique of CCM, Feb. 2003.

[19] 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*, pages 289–298, New York, NY, USA, 2005. ACM Press.

[20] 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, 2002. IEEE.

[21] D. Whitfield and M. Hellman. Privacy and authentication: An introduction to cryptography. In *Proceedings of the IEEE*, volume 67, pages 397–427, 1979.

[22] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). RFC 3610, Internet Engineering Task Force, Sept. 2003.