A Framework for Secure End-to-End Delivery of Messages in Publish/Subscribe Systems

Shrideep Pallickara, Marlon Pierce, Harshawardhan Gadgil, Geoffrey Fox, Yan Yan, Yi Huang
{spallick, marpierc, hgadgil, gcf, yayan, yihuan}@indiana.edu
Community Grids Lab, Indiana University

Abstract
In the paper we present a framework for the secure end-to-end delivery of messages in distributed messaging infrastructures based on the publish/subscribe paradigm. The framework enables authorized publishing and consumption of messages. Brokers, which constitute individual nodes within the messaging infrastructure, also ensure that the dissemination of content is enabled only for authorized entities. The framework includes strategies to cope with attack scenarios such as denial of service attacks and replay attacks. Finally, we include experimental results from our implementation of the framework outlined in this paper.

Keywords: publish/subscribe systems, security, distributed messaging, middleware

1. Introduction

Entities in distributed systems communicate through the exchange of messages. The underlying messaging framework in these systems could be based on point-to-point communications, queuing systems, peer-to-peer (P2P) based interactions, hardware multicast-based disseminations or publish/subscribe systems. The work described in this paper focuses on publish/subscribe systems.

In publish/subscribe systems typically there are one or more router nodes, referred to as *brokers*, which are responsible for routing messages from the publishers to the subscribers. Individual messages are routed based on the *topic* information contained within these messages. This information is added by the publisher to describe the contents of the message. Subscribing entities need to first register their interests in specific topics, also referred to as *subscriptions*, with the broker. Depending on the expressiveness of the content description, there are two kinds of topics viz. simple and complex. In the case of *simple topics*, this is usually a "/" separated String such as /Sports/Football. *Complex topics* are those that describe the content in a more precise manner. Here, the content could be described using a set of tag=value pairs, a set of properties associated with the message, verbose text or as an XML (eXtensible Markup Language) document. In each of these cases the corresponding subscriptions provided by an entity would be <tag, value> pairs with wild card operators, SQL queries on the properties, regular expressions that should be evaluated against the verbose text and finally XPath or XQuery queries on the XML document describing the content.

Upon receipt of a message (previously issued by a publisher) the broker checks the topic information contained within the message with the list of previously registered subscriptions. The broker then proceeds to route the message to subscribers with *matching* subscriptions. Here matching refers to the process of evaluation the stored subscriptions against the topic information contained within the message.

The work presented in this paper focuses on the secure end-to-end delivery of messages within publish/subscribe systems. This paper is organized as follows. In section 2 we provide a brief overview of the NaradaBrokering system, and the topic discovery scheme which is leveraged within our security framework. Section 3 provides a brief overview of the security framework, while section 4 describes this framework in detail. In section 5 we describe how the framework copes with various attack scenarios. We include benchmarks from our implementation of this security framework in section 6. Related work is included in section 7, and we outline our conclusions and future work in section 8.

2. The NaradaBrokering System

NaradaBrokering (http://www.naradabrokering.org) [1-3] is an open-source, distributed messaging infrastructure based on the publish/subscribe paradigm. The subscription formats supported by the messaging infrastructure include Strings, Regular expressions, SQL queries, XPath queries and comma-separated tag=value pairs. For a given topic the system provides services reliable delivery and ordered delivery. Additional services include Network Time Protocol based synchronized timestamps [2] at distributed entities, buffering services to reduce jitter in multi-media settings, replay and recording services and finally discovery services such as broker discovery and topic discovery. The infrastructure provides support for the Java Message Service specification. More recently, we have incorporated support for Web Service specifications such as WS-Eventing [4] in the area of publish/subscribe messaging and WS-ReliableMessaging [5] and WS-Reliability [6] in the area of reliable messaging.

2.1 The Topic Creation and Discovery Scheme

Interactions between entities in publish/subscribe systems are predicated on the knowledge of the topic that will be used for communications; the publisher will publish over this topic while the subscribe registers a subscription to this topic. The topic discovery and creation scheme in NaradaBrokering facilitates the creation, advertisement and authorized discovery of topics by entities within the system. The discovery process is a distributed process and is resilient to failures that might take place within the system. Topic creators can advertise their topics and can also enforce constraints related to the discovery of these topics. Specifically, a topic creator may require the presentation of appropriate credentials (a X.501 security certificate [7]) prior to being able to discover a topic. This discovery scheme addresses issues such as

1. Provenance –- The system can verify easily the owner of a certain topic.
2. Secure discovery –- A topic owner can restrict the discovery of a topic only to authorized entities or those that possess the valid credentials.

These capabilities are provided by specialized nodes – Topic Discovery Nodes (TDNs) – within the system. Since a given topic advertisement will be stored at multiple TDN nodes, this scheme easily sustains the loss of TDN nodes due to failures or scheduled downtimes. Additional details regarding the topic discovery scheme can be found in Ref [3].

3. The Security Scheme: Overview

This section provides an overview of the objectives and operations involved in the secure delivery of messages. Subsequent sections explain the scheme in greater detail. A message comprises the topic information, message headers and finally the content payload. During secure

communications it is the content payload of message that is secured. A topic over which communications need to be secure is referred to as a *secure topic*; this would involve authorized publishing and subscribing, in addition to message payloads being encrypted and signed for confidentiality and integrity/tamper-evidence respectively. Associated with every secure topic is a secret symmetric key that is maintained at a Key Management Center (KMC). There can be more than one KMC within the system and a given KMC can manage more than one secure topics. The KMC is also responsible for generating security tokens that are presented by entities to facilitate authorized publishing and consumption of messages.

The secret key associated with a topic is distributed securely to the interested entities (publishers and subscribers alike). A publisher encrypts the content payload of the message with this secret key. To ensure integrity of the payload, this publisher also signs the encrypted payload; this involves computing the message digest of the encrypted payload and encrypting this hashed value with an asymmetric (e.g. RSA) private personal-key. In our approach we secure messages independently of any transport level security. This provides a fine-grained security structure suitable for distributed systems and multiple security roles.

Upon receipt of the secure message, an authorized subscriber can validate the signature to ensure the message's integrity and then proceed to decrypt the encrypted payload using the previously distributed secret key.

Based on the security tokens associated with the actions initiated by the publisher and subscriber, brokers that are part of the messaging infrastructure can enforce authorization rules and prevent (or restrict) the dissemination of content.

4. The Security Scheme

In this section we present the details pertaining to our security framework.

4.1 The CA within the system

In our scheme we have a Certificate Authority (CA) within the system. The most important function that a CA is responsible for is the issuing of certificates to entities within the system. These certificates can also identify an entity as an authorized KMC or TDN. The CA is also responsible for managing revocation lists pertaining to compromised or rogue entities within the system. The CA also notifies brokers and KMCs within the system about any additions to these revocation lists. A newly added broker or KMC may also request the entire set of revoked certificates from the CA. Please note that the certificate revocations are typically done in an out-of-band fashion.

4.2 KMC

A KMC is a specialized node within the system which is responsible for managing information pertaining to secure topics. There can be more than one KMC within the system, and a given KMC may manage more than one secure topic. However, a given secure topic can be managed by only one KMC. A given KMC performs four core functions. First, the KMC is responsible for the generation of the secret symmetric key that is used for encrypting and decrypting content

payloads. Second, the KMC maintains the list of authorized entities associated with a secure topic. In addition to this, the KMC maintains authorization information related to each of these entities. Some of these entities may be registered to publish, subscribe or both.

The third function performed by the KMC is the generation of security tokens. All actions (such as publishing and subscribing) initiated by entities, related to a specific topic, require the presentation of the security token. The KMC generates a security token for every authorized entity. This security-token establishes an entity's rights (publish, subscribe or both) over a secure topic and the duration for which these rights are valid. This security token comprises the following elements:

a. Client Certificate, including the identifier or the Distinguished Name.
b. Rights – publish, subscribe or both
c. Duration for which these rights, and in fact the token itself, is valid.

Entities are expected to include this security token along with every action they initiate with individual brokers within the messaging infrastructure. To enable individual brokers to detect if the token has been tampered with the contents of this security-token is hashed and signed by the KMC.

Third, the KMC is also responsible for the secure distribution of secret keys and secure tokens associated with a secure topic. To do this the KMC encrypts the contents of the message with a secret key. This secret key is then encrypted using the entity's public personal-key. Only the entity that is in possession of the corresponding private personal-key is able to decrypt the contents of the communications.

Finally, the KMC also sets up a topic over which authorized entities may communicate with it through the exchange of messages. In its topic advertisement to the TDNs, the KMC can also specify restrictions pertaining to the discovery of this topic. Only, entities that are authorized or those that have the right credentials would be authorized to discover the aforementioned topic. No entity will communicate directly with the KMC; no entity, expect the KMC administrator, will be aware of the physical location of the KMC. This provides an additional degree of insulation from denial of service attacks that are targeted at the host and assorted port numbers over which the KMC listens to for communications.

4.3 Registering a secure topic

To ensure secure communications over a topic, the topic owner first needs to register the secure topic with a KMC. To do this, the topic owner first needs to discover the topic over which a KMC, that might host the secure topic, communicates. When the topic owner initiates this discovery request if it had valid credentials there would one or more responses (containing the advertisements) from the TDN identifying the topics over which the corresponding KMCs communicate. Since a KMC may restrict discovery of KMC-Topic based on the presentation of appropriate credentials, it is possible that a topic owner will not be able to retrieve topic information pertaining to some of the KMCs within the system.

Based on these responses, the topic owner decides the KMC that it would communicate with. The topic owner then registers the secure topic with a KMC while specifying parameters about the symmetric secret key that would be associated with the secure topic. These parameters

include the encryption algorithm, the key size and the padding scheme to be used. If there are problems with any of these parameters the KMC will report problems back to the topic owner. Additionally, the topic owner may specify the Hashing scheme to be used for signing the security tokens generated by the KMC. Having a larger digest increases the integrity of the message; in the case of smaller digests a malicious user can exploit vulnerabilities in collisions arising from the hashing function employed to compute the digest. In our scheme by default we use the SHA-1 [8] based message digests which are around 160 bits.

Trade offs between encryption strength and the performance of the encryption algorithms need to be considered while determining the key lengths for encryptions. Short key lifetimes in general tend to mitigate the effects of lost/stolen keys.

The topic owner then proceeds to register the entities authorized to communicate over the managed secure-topic and the rights (publish/subscribe) and the corresponding duration associated with these rights. This information is used to create the security token associated with every authorized entity.

4.4 Entity KMC communications:

All communications that the entities and the KMC have with each other need to be secure. To ensure this, all exchanges between the entities and KMC are encrypted using the following rule. First, a secret symmetric key is generated and the payload of the message is encrypted using this key. Second, depending on the direction of the communication this secret key is then secured using the KMC's or the entity's public personal-key. Only the entity or the KMC that is in possession of the corresponding private personal-key is able to decrypt the secret key that was used for encrypting the content payload.

This method leverages both symmetric and asymmetric key encryptions. Specifically, asymmetric encryptions have higher overheads for large payloads. By restricting the use of asymmetric encryptions (and subsequent decryptions) to operate on only the secret key, which would typically be a 256-bit AES key [9], we have worked around the high overhead constraint for asymmetric encryptions/decryptions.

4.5 Distributing keys and security tokens

An entity interested in communications over a secure topic, discovers the topic over which the KMC managing this secure topic communicates. Once this KMC-interaction topic has been discovered the entity issues a request message to the KMC to retrieve the Secret-Key associated with the secure-topic. In this request the entity also includes its credentials and the topic over which any responses should be issued.

Upon receipt of this secret-key request, the KMC first checks to see if the entity is authorized to receive the secret secure-topic key. If the entity is indeed authorized for communications over the secure-topic, the KMC securely routes this secret key to the entity based on the strategy outlined in section 4.4. In addition to the secret key that was routed to the entity, the KMC also routes the corresponding security token (discussed in section 4.2) associated with the requesting entity. This security-token establishes an entity's rights (publish, subscribe or both) and the duration for which these rights are valid.

An entity is expected to include this security token for all actions and exchanges related to this secure topic. These actions include publishing messages or subscribing to the secure-topic. A broker will not perform the expected actions if the entity fails to include this token in exchanges related to this topic. This discussed in greater detail in section 4.8.

4.6 Publishing messages

When a publisher is ready to publish a message, it performs three steps. First, it encrypts the content payload of the message with the secret secure-topic key that it received from the KMC. Second, the entity also includes its security-token within this message. Finally, the publisher then proceeds to sign the message by computing the message-digest of the encrypted content payload and then encrypting this computed message-digest with its private-personal key. This digital signing enables subscribers to verify the integrity of the message by checking to see if the message has been tampered with.

4.7 Subscribing to secure topics

When an entity is ready to subscribe to a secure topic it includes the security token, assigned to it by the KMC, in its subscription request. Failure to include this security token in its subscription request would prevent the messaging infrastructure from routing messages which match the specified subscription constraint.

4.8 Broker Operations

All entities within the system use the broker, which it is connected to, to funnel messages and exchanges into the messaging infrastructure. The broker thus plays a very crucial role in the dissemination of messages. In this section we describe the broker functions in detail.

4.8.1 Keeping track of revoked certificates

In order to verify the credentials and signatures associated with messages and exchanges a broker needs to keep track of revoked certificates. When a broker starts up for the first time, it retrieves the list of revoked certificates (including those issued to KMCs and TDNs). A broker may store (and retrieve) the list of revoked certificates onto (and from) a stable storage that it has access to. This ensures that between successive re-starts a broker only needs to retrieve those certificates that have been revoked in the interim.

4.8.2 Keeping track of secure topics

Brokers (newly added or otherwise) know about secure-topics as soon as they receive messages targeted to a secure-topic and containing valid security tokens. When a broker joins the broker network the newly added broker retrieves information regarding secure-topics from the broker that it connects to.

If a broker receives messages targeted to a secure-topic without a valid security-token it will discard the message without any further processing and concomitant routing. In case a broker is not aware of a secure-topic this broker may propagate the message without a security token (and

without performing steps outlined in the next sub-section). A broker that encounters such a message, on what it knows to be a secure-topic, issues an error-message back to the broker that it received the message from. Upon receipt of such a message, the original broker updates its secure-topics list to include the one that it missed previously.

4.8.3 Handling messages published by a publisher

Upon receiving a message, issued over a secure topic, from an entity the broker checks to see if there is a security token associated with the message. If the topic is a secure topic and there is no security token associated with the message, the broker discards the message without performing any checks and concomitant routing.

If on the other hand there is a security token associated with the message targeted to a secure topic, the broker performs three functions – verifying if the security token is valid, check topic information contained in token and check for rights associated with the token. If any of these three checks fail the message is simply discarded.

To verify that the security token is a valid one the broker first checks to see if the certificate associated with the KMC, which signed this token has been revoked. Next, it checks to see if the token has been tampered with, by verifying the signature associated with the security token.
Next, the broker checks to see if the topic in the token matches the topic contained within the message. In the case of complex topics, the template for the topic contained within the token should match the complex topic associated with the message. For e.g. the topic template is Sports=NBA, Team=* matches the complex topic Sports=NBA, Team=NYK.

Finally, the broker checks to see if the security token indicates that the publisher that signed the message, indeed has publish permission reflected in its security token. If these checks are completed successfully, the broker proceeds to route the message within the distributed messaging infrastructure.

4.8.4 Routing messages to subscribing entities

When a broker receives a subscription request to a secure topic if there is no security token associated with the request, the request is discarded and an error message is issued back to the entity. If there is a security token associated with the subscription request, the broker performs three checks – validity of the security token, topic information and subscription rights. If either of these checks fails the request is discarded and an error message is issued back to the subscribing entity.

A broker will not route messages, corresponding to subscription to a secure topic, under two distinct conditions. In the first scenario the token associated with the subscription is no longer valid. This might be because either the certificate of the KMC that issued the token was revoked or because the duration of validity contained with the security has elapsed.

In the second scenario, it might be possible that a subscription without a security token was allowed by a certain broker that was not aware of a certain secure topic. When the broker encounters a message with a valid security token, it determines the matching subscriptions for

the message; if any of these subscriptions do not have a security token associated with them, the subscription is removed and the message is not routed to that subscription.

5. Coping with various attack scenarios

In this section we outline the various attack scenarios that we try to deal with. We do not address (and consider it out of our research scope) cryptographic attacks.

5.1 Denial of service

In denial of service attacks the attacker may try to overload the system resources such as CPU and network cycles by generating a large volume of spurious messages that are processed by the system. Since only authorized entities are allowed to publish messages within the system, messages published by unauthorized entities would be rejected at brokers that receive them. In the event that someone tries to overload the KMC by sending multiple messages to the topic over which the KMC communicates; the KMC will generate a new topic for communications and unsubscribe to messages issued to the old (and compromised) topic. In general it is difficult to "guess" the KMC topic since it based on a randomly generated 128-bit UUID.

No one except the KMC administrator and the broker that the KMC has connected to are aware of the physical network address and ports associated with the KMC. It is thus quite difficult to launch a direct denial of service attack on the KMC. In the unlikely event that this information has been compromised, and a denial of service attack was launched based on the network address and port numbers, this particular vulnerability may be addressed in the implementation by rejecting socket connections from IP addresses that have made multiple bogus attempts.

5.2 Thwarting replay attacks

It is possible for a rogue entity to capture valid messages and continually publish these messages. To thwart this, a broker keeps track of timestamps associated with the messages published over secure-topics by authorized entities. For every authorized entity the broker maintains the timestamp associated with the last secure message published by that entity. If the timestamp associated with a message is less than or equal to the timestamp associated with the last message published by the entity in question, the message is discarded as a duplicate.

Some schemes resort to maintaining a list of identifiers (typically UUIDs) associated with published messages to determine duplicates. This scheme obviates the need to maintain this information. If the timestamps have millisecond resolution, this limits the rate at which secure messages could be published by an entity to 1000 per second. This limitation may be circumvented by also including sequence numbering if the timestamps for two messages are identical. In this case, a broker maintains the timestamp and the sequence numbering within this timestamp. For two messages issued by an entity with identical timestamps a broker will not reject the second message so long as the sequence numbering for the second message is greater than the first one.

This scheme can also be used in tandem with the global Network Time Protocol based timestamp scheme so that a broker can discard an old message based on the timestamps contained within the message.

5.3 Detecting and responding to a security compromise

One of the ways to detect security compromises is to issue authentication challenges at regular intervals along with shorter key lifetimes. Most current systems are designed for fixed user session periods (a few hours) or for long-term key lifetimes (a year or two). In the scheme that we propose entities would need to negotiate and retrieve new keys after the delivery of a set of messages or a period of time. Additionally, entities may be forced to answer queries from a set negotiated between the KMC and the entity during initializations. When it is detected or reported that an entity's security has been compromised the following operations need to be performed –

(a) *Generation of new keys*: New keys need to be generated for the secure topics that the entity can publish and subscribe to.

(b) *Propagation of compromise detection*: A message also needs to be propagated throughout the system (to brokers and entities alike) propagating the invalidity of the affected entity's signature. Entities (currently present in the system) that receive these notifications and are affected by it, renegotiate new keys for the affected topics. We could require disconnected entities to do a check on whether the security keys for any of the topics, to which it publishes/subscribes to have changed. If it has the entity needs to retrieve these new topic keys.

(c) *Encrypting replay of messages with new keys*: Routing of missed messages is done based on the new key.

6. Experimental Results

In this section we describe performance numbers from our implementation of the framework described in this paper. We have measured several aspects of the security framework, so that the reader has a precise idea of the costs involved in secure communications. The implementation of this framework was in Java, and the cryptography package that we used for our benchmarks was BouncyCastle (http://www.bouncycastle.org) Version 1.3. To test our system we used the topology depicted in Figure 1.

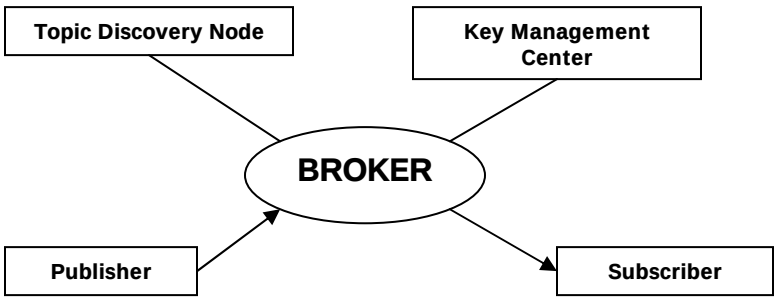

Figure 1 Test Topology

The machines (A, B, C and D) used and their respective configurations are tabulated in Table 1. The KMC, the TDN and the broker were all hosted on different machines (A, B and C respectively). To obviate the need for clock synchronizations and coping with clock skews, the publisher and the subscriber were hosted on the same machine (D). Messages issued by the publisher are time-stamped and upon receipt at the subscriber (after traversal over the network to the broker and then to the subscriber) the transit delay is computed. In all cases we have repeated

the experiment several times for a given message payload size; we then report the mean transit delay and standard deviation associated with a specific payload size.

Machine	Architecture (uname –a)	JVM Version used
A. gf6.ucs.indiana.edu **B.** gf8.ucs.indiana.edu	Linux 2.4.22-1.2188.nptlsmp #1 SMP	Java HotSpot(TM) Client VM (build 1.4.2-beta-b19, mixed mode)
C. complexity.ucs.indiana.edu	SunOS complexity 5.9 Generic_112233-03 sun4u sparc SUNW,Sun-Fire-880	Java HotSpot(TM) Client VM (build 1.4.2-beta-b19, mixed mode)
D. Trex.ucs.indiana.edu	Linux trex 2.6.5-7.155.29-default i686 i386 GNU/Linux	Java HotSpot(TM) Client VM (build 1.4.2_08-b03, mixed mode)

Table 1 Machine configuration for Distributed Test

6.1 Registering a secure topic

Here we outline the costs involved in the registration of a secure topic. Here, we do not include costs related to the discovery of the topic over which the KMC communicates. The costs for discovery of topics (a one-time operation) is approximately 250 – 500 milliseconds and depends mainly on the number of matching topics discovered; for detailed results please see Ref [3].

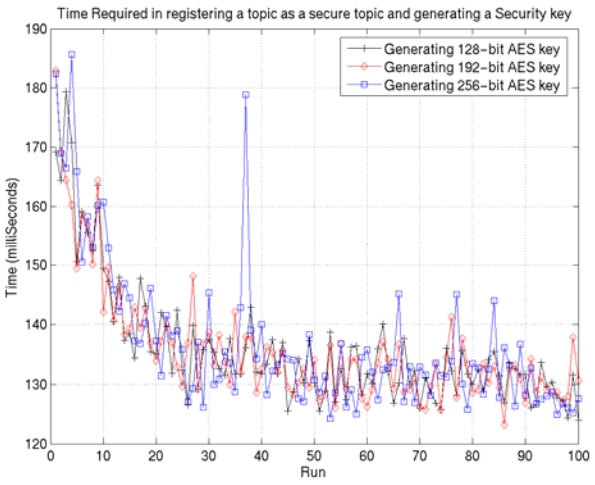

Figure 2 Secure topic registration at KMC

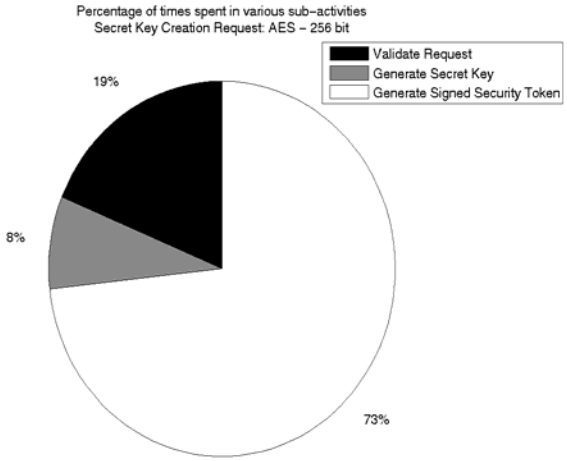

Figure 3 Percentage of time spent in various activities of Topic Registration

Figure 2 depicts the costs involved in generating an AES secret key with different keys sizes – 128, 192 and 256 bits length. The results have been reported for multiple runs. The initial runs tend to be a little more expensive than subsequent ones; this can be attributed to initialization overheads. Figure 3 outlines the percentage of time spent in various parts of the topic registration process.

6.2 Requesting a Secure Topic Key and Security Token

Here we report the costs related to requesting a secret key and the corresponding security token associated with the entity. Upon receipt of this request, the KMC validates the request, and

proceeds to securely distribute the secret key and security token back to the entity. The numbers that we report are for the case where the topic owner had previously set up a 256-bit AES key. This experiment was performed a 100 times, and the overheads are reported below. The mean overhead for this operation was 102.27 milliseconds, with a standard deviation of 7.194 milliseconds with a standard error of 0.7194 milliseconds.

6.3 Sending and receiving secure messages

Here we report the costs involved in the secure end-to-end delivery of messages from the publisher to the subscriber. The costs that we report here include the 3 main components of secure end-to-end delivery –-

1. Publisher costs: This includes the costs related to encrypting the message payload with the topic secret key, signing the message and including the security token assigned to it by the KMC.
2. Broker costs: This includes the costs for validating the security token, verifying the publisher's signature, computing the destinations and finally routing the message to authorized subscribers.
3. Subscriber costs: This includes costs for validating the publisher signature to check for message integrity and decrypting the content payload using the secret topic key that it had previously retrieved from the KMC.

Since a secure topic owner may request different encryption algorithms, key sizes and message digest schemes, the overheads associated would be different depending on the scheme being used. In our measurements for end-to-end delivery of messages we have benchmarked the end-to-end delivery of messages where the AES key sizes have been 128, 192 and 256 bits. For all our measurements we also used SHA1 hashing with a 1024-bit RSA encryption for generating the digital signature. The payloads for messages were varied from 1 byte to 64KB in increments which were powers of 2. For each test case we ran the experiment 15 times, and removed outliers (if any) in the final data set. For each set we measured the mean and the standard deviation within the samples. For every experiment, we have also timed the overheads related to encrypting the payload (at the publisher), performing the security check (at the broker) and decrypting the message payload (at the subscriber).

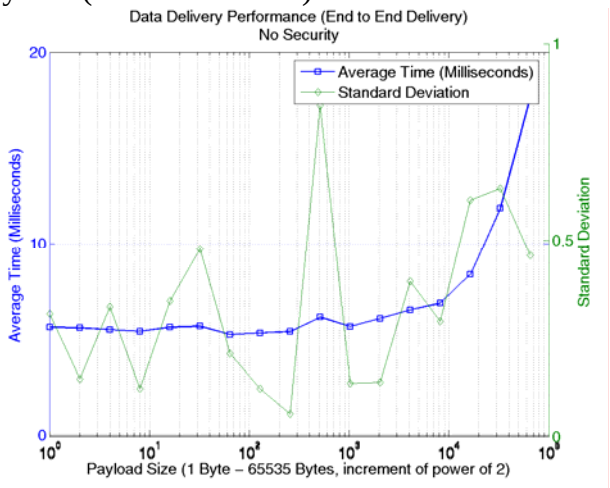

Figure 4 End-to-End delivery in wide area setting with no security

To better understand the overheads introduced by the security scheme we first measured the overheads for communications without any security related overheads. This is depicted in Figure 4.

Figure 5, Figure 7 and Figure 9 depict the costs involved in the secure end-to-end delivery of messages with different AES based topic secret key sizes of 128, 192 and 256 bits respectively. Figure 6, Figure 8, and Figure 10 depicts the overhead introduced by the publisher (encryption of payload), broker (validation of payload and signature) and the subscriber (decryption of payload). We noticed that, in our benchmarks, larger AES key-sizes did not necessarily increase the costs involved in encryptions and decryptions, especially for the payload sizes under consideration in our benchmarks. The costs associated with secure end-to-end delivery are higher than those in the non-secure case, depicted in Figure 4, due to the security related overheads.

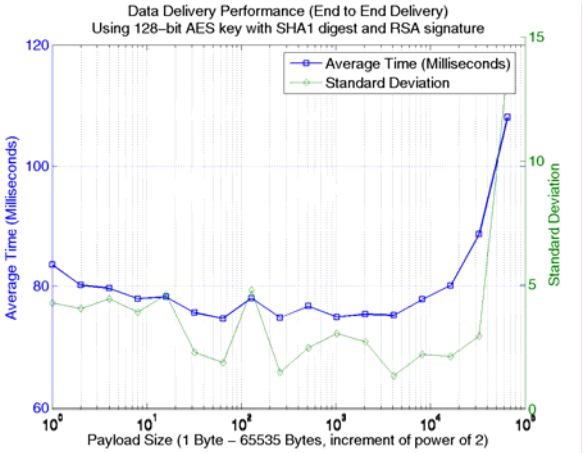

Figure 5 End-to-End delivery using 128 bit AES key

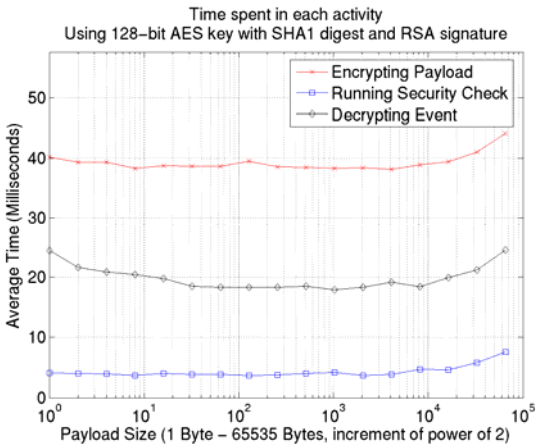

Figure 6 Overhead using 128-bit AES key

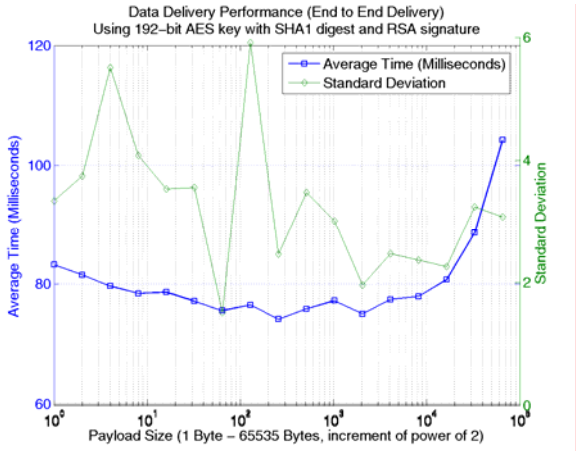

Figure 7 End-to-End delivery using 192 bit AES key

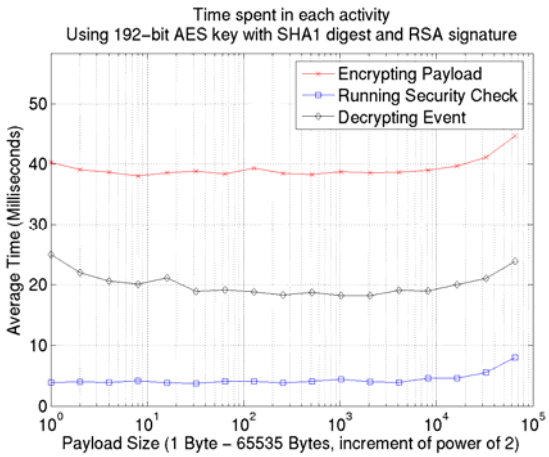

Figure 8 Overhead using 192-bit AES key

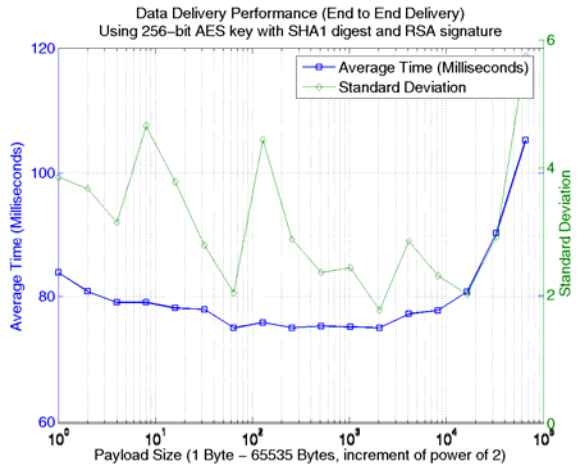

Figure 9 End-to-End delivery using 256 bit AES key

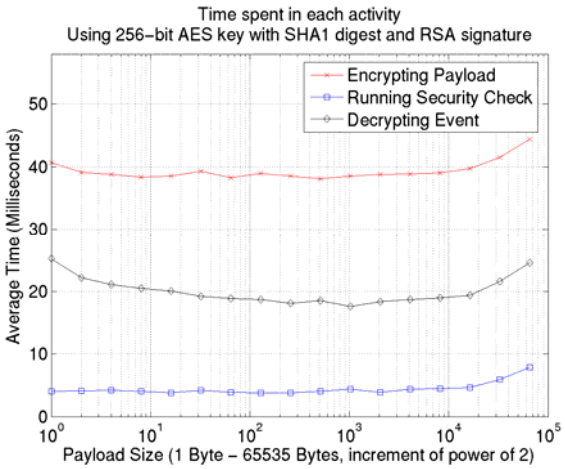

Figure 10 Overhead using 256-bit AES key

7. Related work

GKMP [10] outlines an architecture for the management of cryptographic keys for multicast communications. GKMP creates keys for cryptographic groups and distributes this key securely to the group members while incorporating peer review to incorporate the security policy. GKMP also denies access to known compromised hosts, while monitoring permissions and updating them. Ref [11] outlines strategies for reducing the number of encryptions required to preserve confidentiality between an end-point broker and its subscribing entities in the context of Content based publish-subscribe systems.

P2P systems incorporate several strategies that address secure interchange while incorporating schemes to incorporate trust and reputations. Groove [12] provides excellent P2P security by securing shared spaces, which comprise documents, messages etc. Incremental changes to a shared space object are transmitted to authorized peers in a secure way. Systems such as http://www.advagato.org incorporate trust metrics to support reputations while defeating scenarios where users band together to boost reputation scores. The Free Haven system [13] provides strategies for incorporating accountability while maintaining peer anonymity. Each server in Free Haven maintains values pertaining to reputation and credibility, while broadcasting referrals in some cases.

The Grid Security Infrastructure (GSI) [14] provides a complementary approach that addresses a related problem: a user may need to invoke a particular service through one or more proxy servers. GSI breaks this request into a chain of point-to-point invocations, with the user's initial (proxy) credential being used to create a sequence of proxy key pairs. Each key pair is delegated limited authority to invoke a remote service. Thus the GSI approach treats secure end-to-end connections as a sequence of secure point-to-point connections. We take a complementary approach that enforces security at the endpoints and allows the message to travel securely through insecure intermediaries. The Akenti system [15] addresses the important problem of authorization of resources in a distributed system with multiple stakeholders. Akenti provides an XML access policy language that is transmitted using X.509 policy certificates. This system is complementary to the authentication and message privacy issues that we concentrate on and could potentially be used to govern access to publishing topics. Legion

(http://www.cs.virginia.edu/ ~legion/) is a long-standing research project for building a "virtual computer" out of distributed objects running on various computing resources. Legion objects communicate within a secure messaging framework [16] with an abstract authentication/identity system that may use either PKI or Kerberos. Legion also defines an access control policy on objects.

There are many emerging issues pertaining to security in XML-based Web Services. WS-Security [17] from IBM and Microsoft outlines a proposed architecture to address the gaps between existing security standards and Web Services such as SOAP [18]. By abstracting security services, the WS-security model also serves to unify security technologies such as PKI and Kerberos. Security specifications for Web Services are just starting to emerge, but generally follow the same approach: the message creator adds a signed XML message containing security statements to the SOAP envelope. The message consumer must be able to check these statements and the associated signature before deciding if it can execute the request. Web Services such as those based on SOAP are essentially exchanging XML messages. XML-based message-level security has the additional advantage that it builds upon existing specifications for signing (XML signatures) [19] and encryption (XML encryption) [20], and also allows us to develop a basic security package that can work with both Web Services (communicating with SOAP) as well as peers. The Security Assertion Markup Language (SAML) [21] from OASIS deals with the standard representation of security data – authentication, authorization and attribute – which would be recognized by different application security services irrespective of the security technology or policy that they deploy. SAML is designed to work with W3C specifications such as XML Signature and SOAP. XKMS (XML Key Management Specification) [22] specifies the language for key based trust services and includes protocols for registering, locating and validating keys. Finally, XACML (XML Access Control Markup Language) [23] specifies a vocabulary for expressing XML-formatted rules for making authentication and authorization assertions. XACML uses SAML to define subjects and associated actions. These specifications can be used in tandem with our security infrastructure.

The Open Grids Services Architecture (OGSA) extends the Web Service Description Language to address necessary issues such as service metadata and service state. OGSA services, like Web Services, ultimately comprise a message-based architecture. Entity request messages may be encoded for example in SOAP and passed to hosting environments for consumption and execution. The security issues of this system have been reviewed in [24]. Besides client-server-service style invocations, OGSA proposes to define an interface language that would allow services to communicate their state changes with each other through a notification framework. This notification scheme would in practice be bound to various messaging implementations, such as NaradaBrokering. The security scheme described herein may be used to secure the service-to-service messaging layer.

8. Conclusions & Future Work

In this paper we presented our scheme for enabling secure end-to-end delivery of messages in publish/subscribe systems. We also presented performance numbers from our implementation of the framework. We believe that these benchmarks demonstrate that the costs introduced by our security scheme are acceptable. Successive messages would all be delayed by the amount depicted in our graphs. However, the inter-message spacing for all messages in the stream would

be preserved in our scheme. This would make this scheme suitable for several applications that can sustain this initial delay.

In our future work we plan to investigate the deployment of this scheme in the context of audio/video conferencing systems. We expect that this will raise several interesting issues and problems that would need to be addressed.

9. References

[1]. S. Pallickara and G. Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003. pp 41-61. Lecture Notes in Computer Science 2672 Springer 2003, ISBN 3-540-40317-5.

[2]. Hasan Bulut, Shrideep Pallickara and Geoffrey Fox. Implementing a NTP-Based Time Service within a Distributed Brokering System. Proceedings of the ACM International Conference on the Principles and Practice of Programming in Java. pp 126-134.

[3]. Shrideep Pallickara, Geoffrey Fox and Harshawardhan Gadgil. On the Creation & Discovery of Topics in Distributed Publish/Subscribe systems. (To appear) Proceedings of the IEEE/ACM GRID 2005. Seattle, WA.

[4]. Web Services Eventing. Microsoft, IBM & BEA. http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf

[5]. Web Services Reliable Messaging Protocol (WS-ReliableMessaging) ftp://www6.software.ibm.com/software/devel oper/library/ws-reliablemessaging200403.pdf

[6]. Web Services Reliable Messaging TC WS-Reliability. http://www.oasis-open.org/committees/download.php/ 5155/WS-Reliability-2004-01-26.pdf

[7]. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. S. Chokhani and W. Ford, RFC 2527. March 1999

[8]. The Secure Hash Algorithm (SHA-1) specified in FIPS 180-1. Available from http://www.itl.nist.gov/fipspubs/fip180-1.htm

[9]. "AES Proposal: Rijndael", J. Daemen, V. Rijmen, Available from http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf

[10]. H. Harney and C. Muckenhirn. Group Key Management Protocol (GKMP) Specification. IETF RFC 2093. July 97.

[11]. L. Opyrchal and A. Prakash. "Secure Distribution of Events in Content-Based Publish Subscribe Systems." In Proceedings of the 10th USENIX Security Symposium, pages 281--295, August 2001.

[12]. Groove Networks Inc. Desktop Collaboration Software. http://www.groove.net/

[13]. Roger Dingledine, Michael J. Freedman, David Hopwood, David Molnar. A Reputation System to Increase MIX-net Reliability. Proceedings, Information Hiding Workshop, Mar 2001 (LNCS 2137).

[14]. A Security Architecture for Computational Grids," I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998

[15]. "Certificate-based Access Control for Widely Distributed Resources" Mary Thompson, William Johnston, Srilekha Mudumbai, Gary Hoo, Keith Jackson, Proceedings of the Eighth Usenix Security Symposium, Aug. `99.

[16]. "A Flexible Security System for Metacomputing Environments" (HPCN Europe 99), April 1999.

[17]. "Web Services Security (WS-Security) Version 1.0 05 April 2002," B.Atkinson, et al. Available from http://www-106.ibm.com/developerworks/webservices/library/ws-secure/.

[18]. XML based messaging and protocol specifications SOAP. http://www.w3.org/2000/xp/.

[19]. "XML-Signature Syntax and Processing, W3C Recommendation 12 February 2002", M. Bartel, J. Boyer,B Fox, et. al. Available from http://www.w3.org/TR/xmldsig-core/

[20]. "XML Encryption Syntax and Processing, W3C Recommendation 10 December 2002", T. Imamura, B. Dillaway, E. Simon Available from http://www.w3.org/TR/xmlenc-core/

[21]. "Assertions and Protocol for the OASIS Security Assertion Markup Language," P. Hallam-Baker and E. Maler, eds. Available from http://www.oasis-open.org/ committees/security/docs/ cs-sstc-core-01.pdf.

[22]. "XML Key Management Specification (XKMS 2.0), W3C Working Draft 18 March 2002", Edited by P Hallam-Baker, Available from http://www.w3.org/TR/xkms2/

[23]. "OASIS eXtensible Access Control Markup Language (XACML)" edited by S. Godik, T. Moses, Available from http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-01.pdf

[24]. http://www.globus.org/ogsa/Security/OGSA-SecArch-v1-07192002.pdf (Global Grid forum document).