Achieving Scalability and Security in Publish-Subscribe Web Services

John C. Zablocki

ABSTRACT
Scalability in Internet-scale distributed systems can be achieved through the publish-subscribe communication paradigm. In publish-subscribe systems, interacting parties communicate asynchronously, possibly connected at different times and possibly without knowledge of each other’s identity. Though this decoupling promotes scalability, it presents significant challenges to a security model. Secure publish-subscribe systems, like other distributed systems, need to account for issues such as authentication, information integrity and confidentiality. Addressing these issues tends to require some level of coupling between communicating parties.

Web Services are quickly becoming the preferred middleware for Internet-scale distributed systems. Recently released Web Services specifications, such as WS-Notification and WS-SecureConversation have made the Web Services model a viable approach for building secure, publish-subscribe systems. This paper will examine four Web Services security related specifications and evaluate how they impact scalability in publish-subscribe Web Services.

Keywords
Publish-Subscribe, Web Services, WS-Security, WS-SecureConversation, WS-Trust, WS-Notification

1. INTRODUCTION
Internet-scale distributed systems require a flexible communication model, capable of supporting thousands of entities distributed throughout the world. Traditional point-to-point communication is prohibitive to such systems in that there is a strict coupling of communicating parties [1][4]. This coupling does not represent the nature of communication on the Internet, which is often anonymous and disconnected.

Systems based on the publish-subscribe (pub-sub) communication model support a natural decoupling of communicating parties. In pub-sub systems, a middleware infrastructure often referred to as an “event service” is responsible for facilitating end-to-end communication between publishers and subscribers. Subscribers inform the event service of their interest in events of a particular nature (subscribe). Publishers publish events to the event service, which is then responsible for propagating those events to interested subscribers.

The requirements for securing an Internet-scale pub-sub system are not unlike those for other communication models. Communicating parties may wish to authenticate each other prior to beginning a conversation. Message integrity and confidentiality are also important issues. Pub-sub systems also have the additional requirements of maintaining subscription and publication confidentiality [2]. Traditional security techniques are used to address these security concerns, but often at the expense of scalability.

Pub-sub systems may be implemented in Web Services by utilizing the WS-Eventing [13] or WS-Notification [9] specifications. A number of specifications are used in concert to secure Web Services, such as WS-Security [14] and WS-Trust [15].

The WS-Notification specification recommends the use of WS-Trust, WS-Security, WS-SecureConversation and WS-Policy for defending against attacks common to pub-sub systems. These specifications will be discussed in terms of the protections they offer and to what extent their implementation impacts scalability.

2. PUBLISH-SUBSCRIBE OVERVIEW
2.1 General Architecture
A pub-sub system is a “routing network” delivering “datagrams from publishers to interested subscribers [2].” An information manager receives publications from information producers and subscriptions from information consumers. The information manager is responsible for the delivery of these publications to interested subscribers [1]. The following sections describe two major pieces in a pub-sub system - the application and the infrastructure [2].

2.1.1 The Application
The application consists of publishers and subscribers. Publishers are responsible for producing messages, which are often termed events [1]. These events are consumed by interested subscribers. Interest in an event or certain types of events is established with the subscription. For example, a likely application for pub-sub systems is a stock quote notification service [1][2]. With the subscription message, subscribers might express interest in a set of stock symbols. These subscribers should receive only those publications relating to the specified stock symbols.

2.1.2 The Infrastructure
The infrastructure is the event service, or the information manager, in a pub-sub system. It provides publishers with an interface to raise events and subscribers an interface for subscribing and unsubscribing to these events. It manages these subscriptions and is responsible for the delivery of published events. This event delivery is often termed a notification [1] and pub-sub systems are often referred to as event notification services [3].

Infrastructure implementations can take on one of two architectures. In a centralized architecture, the same entity is responsible for receiving subscriptions and delivering publications. Many commercial message queuing systems are based on this architecture. By contrast, with a distributed architecture there is no centralized communication point. Separate entities are responsible for receiving subscriptions and delivering publications [1]. Figure 1 shows a distributed architecture, including message flow.
2.2 Variations of Publish Subscribe

Much research into pub-sub systems deals with how subscribers express interest in events [3][5], generally termed expressiveness. Expressive pub-sub systems allow subscribers to receive highly relevant event notifications, but require complex filtering algorithms [1]. Three typical subscription-filtering schemes include topic-based, content-based and type-based.

2.2.1 Topic-Based
Topic-based, sometimes referred to as subject-based, pub-sub systems are the least expressive, but also the least complex to implement. In a topic-based pub-sub system, events are organized into hierarchical topics based on event content [1]. Subscribers register interest in these topics and are subsequently notified of all publications for those topics, including any subtopic in the hierarchy. For example, expressing interest in the topic “Guitar” would lead to all events statically published under this topic – including subtopics – to be delivered to the interested subscriber.

2.2.2 Content-Based
Content-based, sometimes referred to as property-based, pub-sub systems are more expressive than topic-based, but also are more complex. In content-based systems, the actual event content is evaluated against the interest expressed in the subscription. Unlike topic-based systems, content-based systems do not place events into predefined categories [1]. This dynamic evaluation allows for more selective subscriptions. Interested subscribers could subscribe to a topic “Guitar,” but include a query “Brand is Fender,” excluding all Guitar publications not relating to Fender® guitars. This level of selectivity could be achieved in a topic-based pub-sub system, but would require new topics to be defined for each possible refinement of the topic.

2.2.3 Type Based
Type-based pub-sub systems extend both topic-based and content-based systems. In a type-based system, the notion of an event topic is replaced by an event type. Similar to the content-based approach, specific attributes of the event are used to dynamically evaluate whether a subscriber should receive the notification. For example, a subscriber could register interest in events of type “Fender Guitar” with a model property whose value is “Telecaster.”

2.3 Scalability through Decoupling

The loosely coupled nature of pub-sub systems offers scalability at the “abstraction level” [1]. Publishers and subscribers operate independently of each other and can be added or removed from the system with little to no impact. Scalability at the abstraction level does not address specific implementation level issues. In other terms, though the pub-sub model might offer an inherently scalable foundation to applications such as Internet games [4] or airline reservation systems [5], these applications will have specific implementation requirements for scalability in terms of resource consumption (CPU usage, network bandwidth, etc.).

The decoupling of publishers and subscribers occurs on three levels – time, space and synchronization. These three levels of decoupling are the “common denominators” of pub-sub systems [1]. Specific implementation requirements, such as expressiveness [3] and security [2] often compete with abstract scalability. This competition will be explored in later sections.

2.3.1 Space Decoupling
Space decoupling refers to the idea that in a pub-sub system, publishers and subscribers may be communicating anonymously. For example, in a pub-sub auction system, sellers and bidders would likely be “known” by the event service, but might very well have no knowledge of each other.

2.3.2 Time Decoupling
Time decoupling addresses the fact that publishers and subscribers need not be active on the system at the same time. Returning to the pub-sub auction system example (and ignoring publication expiration), a bidder might not be online while a seller has published a new item. But the bidder should still receive these notifications if the auction is still current.

2.3.3 Synchronization Decoupling
Synchronization decoupling requires that publishers and subscribers communicate asynchronously. In other terms, the production and consumption of events should not be required to be the primary application flow for publishers and subscribers.

3. PUBLISH-SUBSCRIBE WEB SERVICES

Internet-scale distributed systems typically involve applications built on different platforms. An e-commerce store may run on the .NET/Windows platform, but could very well consume a service running on a J2EE/Linux platform. Web services are composed of a set of related technologies and specifications (WS-**) that exist for the purpose of connecting these disparate systems. Two recently proposed specifications introduce the pub-sub model to web services – WS-Eventing and WS-Notification. A third
3.1 WS-Eventing
Although WS-Eventing [13] provides a mechanism for establishing and maintaining event subscriptions, it requires tight coupling of event producers and event consumers. With three modes of decoupling common to all Internet-scale pub-sub systems, WS-Eventing must be rejected for use as the foundation of a pub-sub web service. The design of WS-Eventing web services more closely resembles an event-based system, which can be used in non-distributed or distributed systems [8].

Event-based systems behave very much like pub-sub systems and a distinction between the two is not always made [8]. For the purpose of this paper, pub-sub systems will differ from event-based systems in that the decoupling of publishers and subscribers is fully supported in a pub-sub system and optionally supported in an event-based system.

3.2 WS-Notification
WS-Notification [9] is a family of specifications, which may be used in concert to form a full-blown pub-sub system. WS-Notification allows for a WS-Eventing like notification pattern, but also allows for brokered communication between publishers and subscribers. WS-Notification further distinguishes itself from WS-Eventing by referring to events as Situations and event notifications as NotificationMessages. The specifications within WS-Notification are discussed below.

3.2.1 WS-BaseNotification
WS-BaseNotification [10] provides the foundation for messaging in WS-Notification. Similar to WS-Eventing, WS-BaseNotification allows for direct communication between message producers and consumers. In WS-BaseNotification, a NotificationProducer sends a NotificationMessage to a NotificationConsumer.

WS-BaseNotification distinguishes between subscribers and NotificationConsumers. Subscribers register interest in a topic through a Subscribe message. The Subscribe message includes a reference to a NotificationConsumer. This reference is an EndPointReference from the WS-Addressing specification, which provides a Uniform Resource Indicator (URI) for the consumer service. A Subscriber may optionally implement the NotificationConsumer interface.

NotificationConsumers participate in direct messaging with NotificationProducers. The Notify message has been defined for generic exchanges between these entities. The Notify message contains the NotificationMessage, within which are the topic and payload, as well as an optional reference to the NotificationProducer. The NotificationProducer acts as publisher in WS-BaseNotification. In addition to producing NotificationMessages, NotificationProducers must accept Subscribe messages and optionally support the GetCurrentMessage, which allows NotificationConsumers to receive a message published while they were offline. Additionally, NotificationProducers must support a message exchange for advertising information about the topics it supports.

Though the NotificationProducer accepts Subscribe messages, it may optionally delegate subscription management to a Web service that implements the SubscriptionManager interface. A SubscriptionManager allows requestors to pause and resume subscriptions. It also maintains a list of NotificationConsumer references and the topics to which they have subscribed. The NotificationProducer uses the SubscriptionManager to determine which NotificationConsumers should receive a NotificationMessage.

3.2.2 WS-BrokeredNotification
WS-BrokeredNotification extends WS-BaseNotification to allow for decoupled communication between publishers and NotificationConsumers through an entity known as a NotificationBroker. A NotificationBroker implements both the NotificationProducer and NotificationConsumer interfaces and is responsible for facilitating communication between producers and consumers.

In WS-BrokeredNotification, the Publisher role belongs to a Web service that implements the NotificationProducer interface. When a Situation arises, a NotificationProducer exchanges a NotificationMessage with a NotificationBroker through a Notify message exchange. NotificationBrokers are then responsible for sending the NotificationMessage to NotificationConsumers.

3.2.3 WS-Topics
WS-Topics defines a mechanism for subscribers to express interest in groups of Situations with similar attributes. Related NotificationMessages are organized into a single Topic. Topics are hierarchically defined by topic trees. A forest of topic trees may be grouped within a single namespace known as a topic space. TopicExpressions allow subscribers to selectively express interest in particular topics. Though the WS-Topics specification provides three possible grammars for TopicExpressions, custom dialects may be used.

WS-Notification is primarily a topic-based pub-sub specification in that WS-Topics and TopicExpressions are essential and required components of the Subscribe and Notify message exchanges. However, an optional element in the Subscribe message allows WS-Notification to exhibit some content-based behavior. Specifically, the Selector element allows for a QueryExpression to be used to evaluate NotificationMessage content dynamically to determine whether a message should be sent.

4. SECURITY REQUIREMENTS
The WS-Notification specification [9] notes seven classes of security attacks against which pub-sub systems should defend. These attacks are common to many types of distributed systems, not just pub-sub. However, the pub-sub messaging protocol does introduce specific requirements for protecting publishers and subscribers. These attacks and traditional defenses are discussed in more detail below. Attention is given to the specific concerns of pub-sub systems where applicable.

4.1 Message Alteration
Any system in which messages are exchanged must provide communicating parties a guarantee that a message was not altered.
while in transmission. Such a guarantee provides information integrity. Digital signatures are used to protect against message alteration attacks [1]. To digitally sign a message, the sender must first create a message hash or digest. The hash is then encrypted with the sender’s private key. The receiver must have the sender’s public key so that the message hash can be recovered. The receiver hashes the message and compares it to the decrypted hash. If the hashes are different, the message was altered.

Pub-sub systems have the additional requirement that subscription messages be protected from modification after a message has been received by the event service. In a pub-sub system, the event service must maintain all active subscriptions in original form. It is typical that the subscriber will trust the event service and additional subscription integrity guarantees are not required [2].

4.2 Message Disclosure
Confidentiality requires that only intended recipients are able to read the contents of a message. Message confidentiality can be achieved through any number of cryptographic tools, but most commonly involves symmetric (shared-key) encryption or a combination of asymmetric (public-key cryptography) and symmetric encryption. Standard message encryption techniques typically involve endpoints having some shared secret used to encrypt and decrypt messages. While message disclosure is somewhat easily defended against with point-to-point communication, pub-sub systems present unique challenges.

Content-based pub-sub systems require that the content of a notification be evaluated dynamically in order to determine whether a subscriber should receive that notification. A problem then arises when the publisher wishes to reveal message content only to subscribers and not to the infrastructure. Similar problems exist for subscribers who wish to keep their subscription function secret and publishers who wish to control who receives publications [2].

4.3 Key Integrity
The strength of any shared-key algorithm is in how well the key is protected. Secret keys used to protect message exchanges should be changed frequently using a process known as “re-keying.” Some common re-keying techniques include using a shared-secret and a nonce to derive keys and deriving a series of keys for use in an agreed upon sequence [9].

4.4 Authentication
Authentication is the process of verifying the identity of a system’s user. An authenticated user is one who has provided evidence that an entity is in fact who that entity claims to be. A variety of authentication techniques exist ranging from simple username and password login schemes to digital certificates requiring the existence of a public key infrastructure (PKI). Authentication typically reduces to knowing something or having something that is sufficient to prove identity.

Closely related to authentication is the notion of authorization. Authorization is the process of verifying that an authenticated user has the credentials to access a requested resource. Authorization often involves placing users into various roles and allowing those roles access to various resources.

With pub-sub systems, two flavors of authentication are of interest – end-to-end and point-to-point [2]. End-to-end authentication requires publishers somehow to identify themselves to subscribers using any standard technique. Point-to-point authentication requires authentication between publishers and the event service and the event service and subscribers. In either case, an existing PKI and digital signatures can be used to provide a means for authentication.

4.5 Accountability
Accountability is related to the notion of non-repudiation. In other terms, Alice cannot deny having sent a message to Bob. Often, knowledge of a secret key used to encrypt a message is enough to provide accountability. However, in such a case, it is always possible for Bob to use the secret key to impersonate Alice. To prevent this situation from arising, a PKI and digital signatures may be used [9].

The WS-Notification specification refers to the above notion of accountability. However, another definition exists. Accountability may also refer to the notion of being able to account for resource usage. For example, publishers may wish to charge subscribers for information. However, it is the event service that is responsible for delivering information to interested subscribers. If both publishers and subscribers trust the event service, it can be responsible for billing. Otherwise, the event service must provide a way for publishers or subscribers to audit the system [2].

4.6 Availability
Denial-of-service attacks are a concern for any public-network-based system. Denial-of-service attacks typically involve overloading a system’s resources, rendering it unusable. A number of approaches exist for addressing these attacks.

In addition to standard denial-of-service attacks, pub-sub systems need to protect against malicious publications and subscriptions intended to overload the system. Three possible countermeasures include limiting publication size or frequency, requiring publishers to perform computationally intensive tasks prior to publishing a message or a variation of challenge-response in which subscribers present publishers with a challenge clause to which publishers must respond [2].

4.7 Replay
The resending of a captured message for malicious purposes is a replay attack. A replay attack could be a form of a denial-of-service attack or could even be performed with the intent of causing financial damage. For example, a message exchange that results in the transfer of funds between bank accounts could be replayed causing multiple transactions when only a single one was intended. To prevent against replay attacks, messages should contain timestamps or sequence numbers.

5. SECURITY AND SCALABILITY
Transport Layer Security (TLS) or Secure Sockets Layer (SSL) provides much of the foundation for secure messaging on the Web. However, TLS offers only end-to-end security. All messages routed through intermediaries are exposed. TLS also does not allow for discrete message parts to be secured. The entire message must be secured. Web Services, which are
fundamentally Extensible Markup Language (XML) message exchanges, are well suited to application level security, which can makeup for the shortcomings of TLS [7].

Though the original set of Web services specifications made no mention of security [6], several now exist that may be used to defend against the above-mentioned attacks. Rather than offer novel solutions, the security related Web services specifications adapt existing, well-tested technologies, such as those mentioned above.

A pub-sub Web service’s security architecture is made complicated by the disconnected nature of pub-sub communication. The Web services security related specifications are based on many technologies that presuppose some relationship between communicating parties. However, pub-sub Web services based on WS-Notification allow for brokered communication, which assumes that communicating end-points have no preexisting relationship.

In discussing how to defend against the classes of attack to which WS-Notification is vulnerable, the authors of the WS-Notification specification [9] recommend the use of four security related Web services specifications. The following sections contain an overview of these specifications and a discussion of how scalability is impacted by their implementation.

5.1 WS-Security

5.1.1 Overview

A stated requirement of the WS-Security specification [14] is to offer “end-to-end message level security and not just transport-level security.” In this way, WS-Security can be seen as the first attempt to address the inadequacy of TLS as a foundation for securing Web services. WS-Security is a far-reaching, composite specification that provides protection against many common security threats. Two World Wide Web Consortium (W3C) recommendations provide the core of WS-Security – XML Encryption and XML Signature. WS-Security adapts these specifications for optimal use in Web services.

XML Encryption is used to provide message confidentiality. With XML Encryption, specific XML document elements or entire XML documents may be encrypted. Again, this granularity is not offered by TLS. XML Encryption uses symmetric key encryption algorithms, such as the Triple Data Encryption Standard (3DES) and the Advanced Encryption Standard (AES). Communicating parties may exchange keys real-time by providing a pointer to the key location or by including the key in the message, encrypted with the recipient’s public key.

Message integrity and accountability are provided by XML Signature, which is a schema for formatting a digital signature in XML. Traditional message digest algorithms, such as SHA-1 and MD5 provide the basis for XML Signature. Just as XML Encryption allows for partial message confidentiality, XML Signature allows for partial message integrity. In other terms, all or part of an XML document may be digitally signed.

Authentication is also addressed by WS-Security. The most basic form of authentication offered by WS-Security schema is the use of the UsernameToken element. Within this element a username and password (or password hash) may be provided. The BinarySecurityToken element provides support for Kerberos tickets and X.509 certificates. Kerberos tickets and X.509v3 certificates are means for allowing a trusted third-party to authenticate a service requestor. A SecurityTokenReference allows a service requestor to specify the location of security credentials through a URI.

Finally, WS-Security offers protection against replay attacks through the combined use of a UsernameToken and a PasswordDigest element. This scheme involves the client hashing a shared password with a timestamp and a nonce (arbitrary set of bytes). The server is responsible for caching the nonce and rejecting any repeated hash in which the same nonce is used (over an agreed upon period of time).

5.1.2 Impact to Scalability

WS-Security relies on shared secrets and public key cryptography to provide its various protections. A publisher wishing to keep a publication confidential must share a secret with subscribers. Similarly, a subscriber wishing to verify a publisher’s identity will likely need that publisher’s public key. Such exchanges require direct interaction between publishers and subscribers.

Space decoupling is achieved in WS-Notification through a brokered communication model and assumes that publishers and subscribers have no knowledge of each other. WS-Security alone does not allow for space decoupling since some keys must be exchanged between publishers and subscribers. Additionally, WS-Security alone does not allow for space-decoupled authentication.

Time decoupling is also a concern for WS-Security. Keys used in XML Encryption and authentication tokens have expiration dates. A publication encrypted with a certain key must be decrypted before the key expires. This forces the pub-sub system to explicitly define the duration for which time decoupling is allowed.

WS-Security and its dependent technologies place no constraints on publishers or subscribers such that asynchronous communication would not be allowed. Therefore synchronization decoupling is not impacted.

5.2 WS-Trust

5.2.1 Overview

Security tokens are the foundation of authentication in WS-Security. Again, supported security token types include Kerberos tickets or X.509v3 certificates. However, WS-Security does not address the need to validate that a security token was issued by a trusted source. Mechanisms for providing this validation are provided by WS-Trust [15].

WS-Trust specifies a brokered-trust model through a specialized Web service known as Security Token Service (STS). Prior to requesting a security token from an STS, a service requestor must have a pre-existing relationship with the STS. This requirement is so that the service requestor can authenticate itself to the STS. If a service requestor is authenticated and proven to be an authorized user of a service, the STS issues a signed security token. The security token is then used as described in the WS-Security specification. If the Web service trusts the STS, it can now trust the service requestor.
5.2.2 Impact to Scalability
It was previously stated that WS-Security alone does not allow for space decoupling. However, when used with WS-Trust, WS-Security may offer this feature. WS-Trust specifically allows for a brokered trust model, which is not found in WS-Security.

In such a service, the NotificationBroker would have to be trusted by publishers and subscribers, as would the STS. Shared secrets and public keys are maintained between publishers and NotificationBrokers and NotificationConsumers. It should be noted that maintaining confidentiality from the NotificationBroker is impractical and would require computationally expensive algorithms [2] that would be prohibitive to implementation level scalability.

The issue facing WS-Security regarding time decoupling is also addressed by WS-Trust. Again, the problem is changed to requiring time coupling of publishers and NotificationBrokers and NotificationConsumers. Synchronization decoupling is not impeded.

5.3 WS-SecureConversation

5.3.1 Overview
For Web services that involve multiple messages, WS-Notification for example, the process of retrieving a security token from an STS could become burdensome. Additionally, re-sending credentials to the STS increases the likelihood of having those credentials compromised [18].

WS-SecureConversation [16] builds upon WS-Security and WS-Trust by allowing for security across several messages – a conversation. This conversation takes place in what is called a security context. The context is maintained through a special security token called a Security Context Token (SCT). WS-SecureConversation does not alone provide protections against specific threats, but allows for WS-Trust and WS-Security to be used more efficiently.

With the first message in the conversation, an encrypted secret key is sent. Subsequent messages contain only the SCT. The communicating parties maintain the secret key for the duration of the conversation – or until the SCT expires. The SCT is distributed in one of three ways – an STS may be used, one of the communicating parties (if trusted) may create the SCT or it may be negotiated.

5.3.2 Impact to Scalability
WS-SecureConversation relies heavily on and achieves space and time decoupling through the mechanisms described by the WS-Trust brokered trust model. To achieve both space and time decoupling, the creation of the SCT requires the use of a STS. The other SCT creation options require direct communication between communicating parties and the issues that apply to WS-Security without WS-Trust would apply here as well.

5.4 WS-Policy

5.4.1 Overview
Web service consumers learn the basic requirements of a particular service through the Web Services Description Language (WSDL). WSDL exposes information such as the name of a service and the input and output values. However, WSDL does not provide a mechanism to specify other important communication parameters [7]. For example, with WSDL it is not possible to specify how long a requestor should wait for a response before re-sending a message.

WS-Policy [17] is a family of specifications that augment WSDL. Of specific interest to this discussion is WS-SecurityPolicy. With WS-SecurityPolicy, a Web service is able to communicate to consumers which WS-Security elements are required or supported. For example, a Web service could assert that message confidentiality is required and 3DES and AES are supported algorithms. Additional assertions may be made on authentication and message integrity requirements. WS-SecurityPolicy also supports assertions for mandating that specific message elements not be encrypted, which is useful when a message needs to be passed through intermediaries.

While WS-SecurityPolicy alone does not offer protection against security attacks, the WS-Notification specification recommends that strict policies are asserted to guarantee that the strongest algorithms be used to maintain key integrity. Any service consumer would not be allowed to participate in a message exchange with the service if such strong algorithms were not supported.

5.4.2 Impact to Scalability
WS-Policy does not provide security directly at run time and therefore does not have a direct impact on scalability of pub-sub Web services. However, the set policies may very well impact scalability as discussed in the previous sections.

6. SUMMARY
The following table summarizes the protections offered by the four security specifications recommended for use in a WS-Notification based pub-sub Web service. The level of decoupling allowed by each specification is also included.

<table>
<thead>
<tr>
<th>Table 1. Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WS Specification</strong></td>
</tr>
<tr>
<td>WS-Security</td>
</tr>
<tr>
<td>WS-Policy</td>
</tr>
<tr>
<td>WS-Trust</td>
</tr>
</tbody>
</table>
service is trusted by both publishers and subscribers. between publishers and the event service and the event service and subscribers.

<table>
<thead>
<tr>
<th>WS-SecureConversation</th>
<th>Efficiency of Authentication</th>
<th>Yes, but SCT must be created by a trusted STS.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes, but expiration dates on shared encryption keys may be a limiting factor, due to the dependency on WS-Security.</td>
<td>No impact.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WS-Policy</th>
<th>Key Integrity</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. CONCLUSIONS
This paper has examined how implementing security in a pub-sub Web service imposes limits on the level of decoupling allowable in the service. Such impositions are prohibitive to scalability at the abstraction level. In deciding how to secure a pub-sub Web service, one must evaluate matters of privacy and trust versus the ability of the system to scale to new publishers and subscribers.

The WS-Notification specification provides the framework for building scalable publish-subscribe Web services, in which decoupling in space, time and synchronization may be achieved. WS-Notification allows for this decoupling by way of a brokered communication model, in which publishers and subscribers do not communicate directly.

WS-Security and its constituent technologies offer several protections to a WS-Notification based web service, but often by coupling publishers and subscribers. WS-Trust provides a brokered trust model that when used with WS-Security and WS-SecureConversation allows for scalable and secure publish-subscribe Web services.

8. REFERENCES


