# A pattern for the WS-SecureConversation standard for web services

Ola Ajaj and Eduardo B. Fernandez
Department of Computer and Electrical Engineering and Computer Science
Florida Atlantic University
777 Glades Road, Boca Raton, Florida 33431-0991 USA
oajaj@fau.edu, ed@cse.fau.edu

***Abstract***:

*When using web services, the involved parties need to address two main concerns: first, each party needs to determine if it can trust the credentials of the other party; and second, how to protect their data and how to provide a secure session between the parties. WS-Trust takes care of the first challenge by defining how to establish trust between interacting parties, while WS-Security is in charge of the second part by providing message integrity, confidentiality, and authentication. However, web services exchange multiple messages that increase the overhead of key establishment and decrease performance, which eventually affects business interactions. By defining a shared context among the communicating parties for the lifetime of a communications session that combines secure communications and trusted relationship, we facilitate the process of communication and increase overall performance. This shared context is implemented by the WS- SecureConversation standard, and we present here a pattern for it.*

## 1. Introduction

Web services interact with users and other web services to conduct business. Those business interactions have different degree of complexity and validity depending on their nature. Some interactions exchange a large number of messages, which adds complexity and overhead to the message exchanges and thus increase cost. In addition, sometimes users and web services are not predefined and known to each other and a trust relationship must be established before any interaction can happen between them. Those users also might have different requirements and their own policy rules, implementing different security constraints. Addressing all these concerns in one abstract and practical solution will facilitate the interaction between web services. This is the motivation behind the WS-SecureConversation standard.

The Web Services Secure Conversation Standard (WS-SecureConversation) is built on top of the WS-Security, WS-Trust, and WS-Policy standards to provide secure interaction and data exchange between web services [IBM]. WS-Security [OASa] describes how to embed existing security mechanisms such as XML Encryption, XML Digital Signature, and Security Tokens into SOAP messages to provide message confidentiality, integrity, authentication, and non-repudiation. WS-Trust [OASb] is a standard to support the establishment of trust relationships between web services. WS-Policy [W3C07] provides specifications that describe the capabilities and constraints of the security (and other business) policies on intermediaries (for example, required security tokens, supported encryption algorithms, and privacy rules) and how to associate policies with services and end points.

To perform its functions, WS- SecureConversation [OASc] defines mechanisms for establishing and sharing Security Context Token (SCTs), and deriving keys from security contexts to authenticate messages between parties. This shared context defines who, how, what and for how long each user is able to conduct business. The involved parties share these SCTs throughout the lifetime of the communication channel [IBM, OASc].

Figure 1 shows a pattern diagram describing the relationships between the patterns for some web services standards. The diagram shows dependencies between the patterns; for example, WS-Security uses policies defined by WS-Policy. Our group has written all these patterns except WS-Federation, which is ongoing work [Fer12].

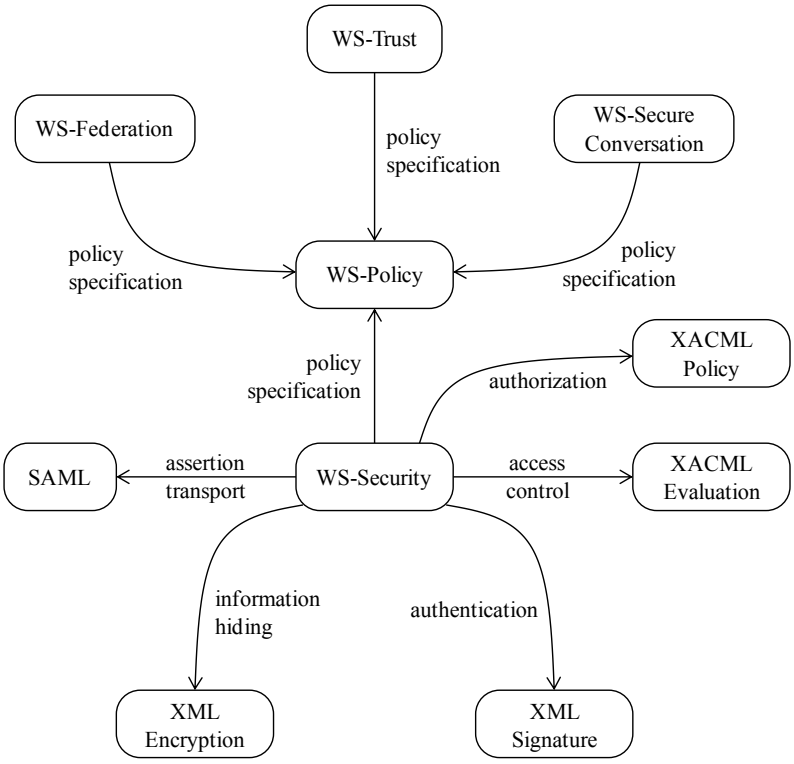


Figure 1: Pattern diagram for web services security standards

Web services standards are rather complex and verbose and it is not easy for designers and users to understand their key points. Our approach is to express web services security mechanisms and standards as patterns. In this way we can verify if an existing product implementing a given security mechanism supports some specific standard [Fer06]. Inversely, a product vendor can use the standards to guide the development of a product. By expressing standards as patterns, we can compare them and understand them better. For example, we can discover overlapping and inconsistent aspects between them. A standard defines a generic architecture and this is a basic feature of any pattern; it can then be confirmed as a best practice by looking at products that implement the standard (and implicitly the pattern).

Section 2 shows a pattern that describes the WS-SecureConversation standard while section 3 ends the paper with some conclusions. Our description is intended for users and designers of business workflow systems that use web services.

## 2. WS- SecureConversation

**Intent**

This pattern describes a standard to allow security context establishment and use through the lifetime of the communication session between web services. This security context is used to provide secure communication between web services by extending the mechanisms of WS-Security, WS-Trust, and WS-Policy.

**Example**

The *Ajiad* travel agency implements several business portals to offer services for tickets, hotel and car rental to its customers. *Ajiad* needs to keep channels that provide secure and trusted relationships with its partners, and needs to be able to determine which travel services to invoke for which customer. Without a well-defined structure, *Ajiad* will not be able to determine if the communication channel is secure or not, if a partner should be trusted or not, or to automate the business relationships securely with its partners. This situation may lead to losing a valuable business goal of offering integrated travel services.

**Context**

Distributed applications using web services that need to collaborate with each other to perform business workflows using insecure networks, e.g. the Internet.

**Problem**

Before initiating a conversation with people, we need to know with whom we are talking and what do we want to talk about. We initiate conversations with different persons, based on our roles and interests. The same is true for web services.

The functions of WS-Security which include integrity, confidentiality and authentication of messages are useful for simple or one-way messages; but this solution is impractical and could cause a problem in case of the necessity of parties to exchange multiple messages [OASa]. If there are multiple message exchanges between service provider and consumer, then the overhead of XML signature and XML encryption are significant.

Further, establishing security relationships is fundamental for the interoperation of distributed systems. Applying relevant policies is needed to make it clear for the users what is allowed or which conditions apply to the use of web services. Without applying relevant policies and trust relationships between the involved parties, web services have no means to assure security and interoperability in their integration and may lose their ability to provide service.

The possible solution to the problem of establishing a secure context is constrained by the following forces:

- **Securing Context Tokens:** While communicating using context tokens, web services exchange multiple messages containing sensitive data; we need to provide message protection for this exchange.

- **Time Restrictions**: Any interactions or means of communications between web services may be restricted in time. We should be able to amend, renew, or cancel those interactions properly, as needed.

- **Policy:** A web service uses policy(ies) to define all the required conditions and constraints that should be met before using that web service. We should reference this policy for verification and proper use.

- **Overhead**: Web services exchange multiple messages that add complexity, increase the overhead of key establishment, and decrease performance; we need to keep overhead at a minimum.

- **Interoperability:** Web services and requesters should interact seamlessly despite differences in domains and platforms.

**Solution**

We define explicitly an artifact that uses a Security Context Token (SCT). The SCT defines what kinds of assertions are required to be satisfied by any interaction between the involved web services and encapsulates the claims and information sent by the requester in order to obtain the required SCT. Once initiated, this SCT can be used to conduct secure communications. All entities involved share a key that has been agreed in order to establish a communication session with their target partners.

*Structure*

Figure 2 describes the structure of this pattern. Pink classes describe a logical web service connection; blue classes describe the token management structure, while yellow classes describe security tokens and claims.

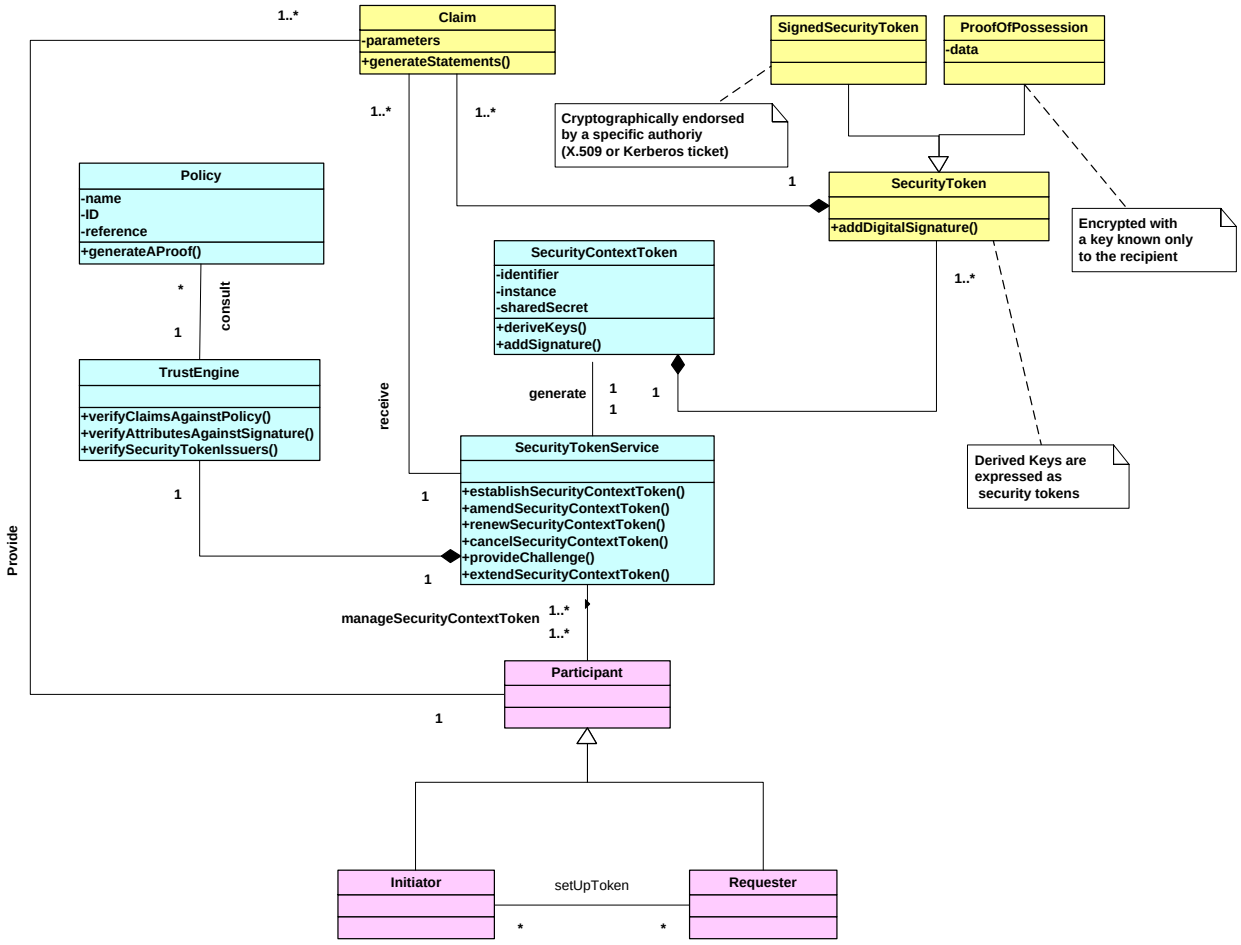


Figure 2: Class Diagram for the WS-SecureConversation Pattern

**Participant** represents an entity (e.g., human, computer, message, an endpoint, interaction, resource) that is in charge of managing SCTs. It can has the role of: **Initiator**, the one creates a SCT or asks a STS to create one for her, or **Requester**, who asks for a SCT to conduct business and use web services.

A **Claim** is a statement about a client, service or other resource (e.g. name, identity, key, group, privilege, capability, etc.). Claims are assertions such as "I am Adnan", "I am an authenticated user and I am authorized to print." A **Security Token** is a collection of claims (such as X.509 certificate, Kerberos ticket, and username).It is responsible for adding signatures to tokens. Security Token also is a generalization of: **Signed Security Token**, that is cryptographically endorsed by a specific authority (e.g. an X.509 certificate or a Kerberos ticket), and **Proof-of-Possession (PoP) Token** which contains a secret *data* parameter that can be used to prove authorized use of an associated security token. Usually, the proof-of-possession information is encrypted with a key known only to the recipient of the POP token.

A **Security Context Token** is a representation of a security context, which in turns refers to an established authentication state with negotiated keys that may have additional security-related properties. Requestors can use SCTs to sign and/or encrypt a series of SOAP messages, known as a conversation, between a message sender and the target web service.

**Security Token Service (STS)** is a web service that issues SCTs by itself, or relies on another STS to do so using its own trust statement. It produces assertions based on evidence that it trusts, provides challenge for requesters to ensure message freshness (the message has not been replayed and is currently active), verifies authorized use of a security token and establishes, extends trust among a domain of services. Each **STS** has a **Trust Engine** that evaluates the security-related aspects of a message using security mechanisms and implies a policy to verify the requester's assertions. The **Trust Engine** is responsible for verifying security tokens and verifying claims against policies. A **Policy** is a collection of policy assertions that have their own name, references, and ID.

***Dynamics***

We describe the dynamic aspects of the WS-SecureConversation using sequence diagrams for the use cases "*Establish a SCT to create a context*" and "*Amend a SCT*".

*Establish a SCT to create a context (Figure 3):*

Summary: STS creates a SCT using the claims provided by the initiator.

Actors: Initiator, requester.

Precondition: The STS has the required policy to verify the requester claims and the requester provides parameters in form of *claims* and *RequestType* signed by a *signature*.

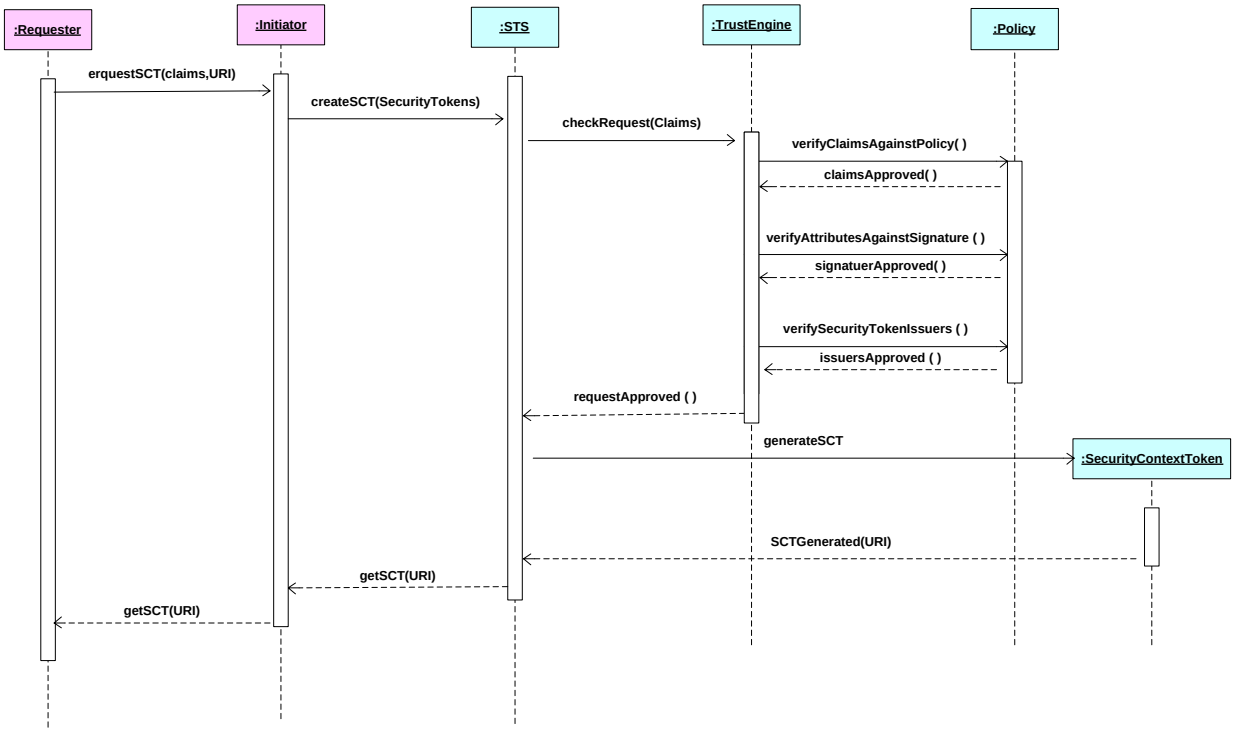


*Figure 3: Sequence Diagram for establishing a SCT to create a context*

Description:

a. The requester asks for a SCT to use a web service.
b. The initiator requests a SCT by sending the required parameters of *claims* signed by a *Signature* to the STS in forms of security tokens.
c. The STS uses WS-Trust mechanisms of Trust Engine and WS-Policy to check the initiator's claims.
d. Once approved, the STS creates a new SCT in the form of an URI and sends it back to the initiator, who in turns sends it back to the requester.

Post condition: The initiator has a SCT that can be used to communicate with other web services.

*Amend a SCT (Figure 4):*

Summary: A STS will amend an existing SCT to carry additional claims upon the initiator's request.
Actors: Initiator.
Precondition: The initiator owns a SCT.

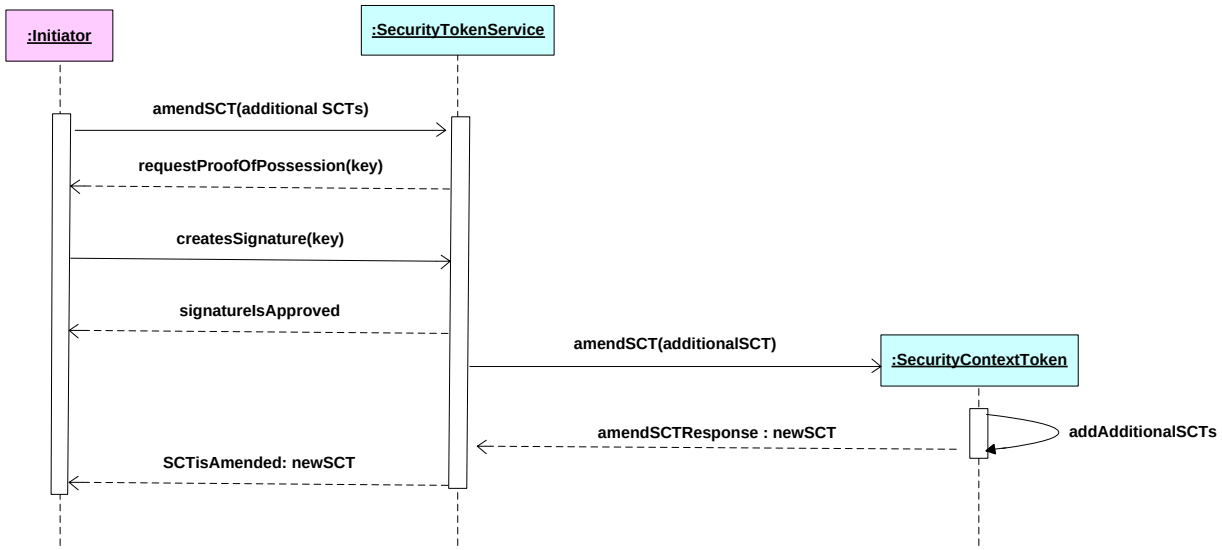


*Figure 4: Sequence Diagram amending an existing SCT*

Description:

a. The initiator asks to change a SCT she owned to carry additional SCTs.
b. The STS asks for the key associated with the SCT as a proof of possession.
c. The initiator replies back with a signature.
d. Once approved, the STS adds those additional SCTs to the SCT to form a new SCT and sends it back to the initiator.

Postcondition: The initiator has an amended SCT that can be used to communicate with other web services.

**Implementation**

When it comes to encrypt a message, SCTs use a symmetric key rather than an asymmetric key, which makes them faster and more efficient. A STC allows a context to be named by a URI, also for reference purposes. The participating web services determine a shared secret to use as the basis of key generation. In other words, shared secrets are in form of derived keys. The derived keys are expressed as security tokens.

In order to assure effective implementation, we need to take in consideration the following:

- Using a service requires a signature to prove knowledge of a security token or set of security tokens. Three possible scenarios are presented to establish a SCT; by one of the communicating parties, by a negotiation/exchange process between the participants, or by a separate STS.

- Although the messages exchanged between the involved entities are protected by WS-Security; still three possible issues related to security tokens are handled: security token format incompatibility, security token trust and namespace differences. The WS-Trust pattern can address these issues by: Defining a

request/response protocol (in which the client sends *RequestSecurityToken* and receives *RequestSecurityTokenResponse* and introducing a Security Token Service (STS).

This pattern could be used to guide the development of a product. Users can use this pattern to ask for certain security mechanisms that fit their business goals. Developers and product vendors should be aware that no complete security solution for web services is guaranteed through WS-SecureConversation by itself. WS–Secure Conversation should be used in conjunction with other web services standards such as WS-Security, WS-Trust, and WS-Policy for an optimal solution [OASc]. Implementing various security mechanisms through those standards will lead to an optimal solution. Implementations of those WS standards are beyond the goals of this pattern and were covered in [Aja10a], [Aja10b] and [Has09].

**Example Resolved**

*Ajiad* now has the ability to automate the business relationships with its partners by assuming that all partners are registered and by issuing customers unique IDs. In this case, *Ajiad* provides an intermediate link between the customers and its partners and plays the role of negotiator as well as third-party player who is looking to satisfy both sides. *Ajiad* now can offer a SCT Service for its business partners, who may find useful ways to take advantage of credit processing and other of its services, *Ajiad* now has new business opportunities.

**Known uses**

- WS-SecureConversation is used in the Microsoft Web Services Enhancement 2.0 toolkit [Gud04].
- WS-SecureConversation support in Apache's CXF builds upon the WS-SecurityPolicy implementation to handle the SecureConversationToken policy assertions [Apa].
- Java applications support in IBM products includes support for WS-SecureConversation [Sos10].
- SAP is using the security context primarily to allow WS-ReliableMessaging to reuse a security context, so that the server can contact the client [SAP].

**Consequences**

The WS-SecureConversation pattern presents the following advantages:

- Policy providers now can use mechanisms provided by other web services specifications such as WS-Security, XML Digital Signature [W3C08], and WS-Metadata Exchange [W3C09] to protect the messages needed to create a secure context.
- **Time restriction**: We can specify time constraints in the parameters of SCTs, which can specify how long that token is active. Upon expiration, the SCT's holder may amend, renew, or cancel it.
- **Policy**: We can implement WS-Policy to support trusted partners by expressing and exchanging their statements of trust expressed as a trust policy.
- **Overhead**: For the communication channels that require end-to-end security and have frequent message exchanges, the WS-SecureConversation may reduce the overhead. Using either encryption or signing is better than using both, since combining both produces significantly lower performance [Liu05].
- **Interoperability**: STS satisfies the capabilities and constraints of the security (and other business) policies on intermediaries which at the end increase the interoperability between web services. By implementing STS, the WS-SecureConversation framework will be more comprehensive and can carry out secure conversations between parties in different trust domains.

The WS-SecureConversation pattern presents the following liabilities:

- The WS-SecureConversation standard is a lengthy document with a lot of details that were left out to avoid making the pattern too complex. Somebody interested in addressing more details can check the WS-SecureConversation Standard web page [OASc].

- No complete security solution for web services is guaranteed through WS-SecureConversation by itself. WS–Secure Conversation should be used in conjunction with other web services standards such as WS-Security, WS-Trust, and WS-Policy for an optimal solution [OASc].

**Related Patterns**

- A Pattern for WS-Security [Has09] defines how to secure SOAP messages applying XML security standards such as XML Encryption and XML Signature.

- A pattern for the WS-Trust standard of web services [Aja10a] provides a framework for requesting and issuing security tokens, and to broker trust relationships. It uses WS-Security to transfer the required security tokens, using XML Signature and Encryption to provide confidentiality. This standard may use WS-Policy to specify which security tokens are required at the target.

- A pattern for the WS-Policy standard [Aja10b] describes how to express requirements that are needed or supported by a web service. For instance, it can indicate that a specific signature algorithm must be used when signing a document.

## 3. Conclusion

We presented a pattern for the WS-SecureConversation that describes how a web service can authenticate requester messages, how requesters can authenticate services, and how to establish mutually authenticated security contexts.

Message authentication is useful for simple or one-way messages; parties intending to exchange multiple messages can create a secure session. A security context is shared among the communicating parties for the lifetime of a communications session. These security context-token-issuance services build on WS-Security, WS-Trust and WS-Policy to transfer the requisite security tokens in a manner that ensures the integrity and confidentiality of those tokens.

In our future work, we will write a pattern for WS-Federation standard, which enables appropriate sharing of identity, authentication, and authorization data using different or similar mechanisms. WS-Federation depends on WS-SecureConversation to perform its functions. This will complete the pattern diagram of Figure1, and will give us the opportunity to investigate and analyze how WS-Federation fits with other web services standards and how it could facilitate implementing these standards in business applications.

## Acknowledgements

We thank our shepherd Kiran Kumar for his useful comments that significantly improved the paper.

## References

[Aja10a] Ajaj, O., and Fernandez E.B., “A pattern for the WS-Trust standard of web services”, *Procs. of the 1st Asian Conference on Pattern Languages of Programs (AsianPLoP 2010)* , Tokyo, Japan, March 16-17, 2010, http://patterns-wg.fuka.info.waseda.ac.jp/asianplop .

[Aja10b] Ajaj, O., and Fernandez E.B., "A pattern for the WS-Policy standard", *Procs. of the 8th Latin American Conference on Pattern Languages of Programs (SugarLoafPLoP 2010),* Salvador, Bahia, Brazil, Sept 23-26, 2010,

http://wiki.dcc.ufba.br/SugarLoafPlop

[Apa] Apache CXF, *WS-SecureConversation*, http://cxf.apache.org/docs/ws-secureconversation.html

[Fer06] Fernandez, E.B., and Delessy, N. 2006. "Using patterns to understand and compare web services security products and standards", *Proceedings of the Int. Conference on Web Applications and Services (ICIW'06)*, Guadeloupe, February 2006. IEEE Comp. Society, 2006.

[Fer12] Fernandez, E.B., Ajaj, O., Buckley, I., Delessy-Gassant, N., Hashizume, K., and Larrondo-Petrie, M.M., "A Survey of Patterns for Web Services Security and Reliability Standards", *Future Internet* **2012**, 4(2), 430-450. http://www.mdpi.com/1999-5903/4/2/430, - Last Access on August, 04, 2012

[Gud04] Gudgin, M., "Using WS-Trust and WS-Secure Conversation", MSDN 2004, http://msdn.microsoft.com/en-us/library/ms996521.aspx, Last accessed on August 16, 2012.

[Has09] Hashizume, K.; Fernandez, E.B. A Pattern for WS-Security. *First IEEE Int. Workshop on Security Eng. Environments*, Dec. 17-19, 2009, Shanghai, China.

[IBM] IBM, Security in a Web Services World: A Proposed Architecture and Roadmap, http://download.boulder.ibm.com/ibmdl/pub/software/dw/library/ws-secmap.pdf - Last accessed on March 30, 2012

[Liu05] Liu H., Pallickara S., and Fox G., " Performance of Web Services Security" *Proceedings of the 13th Annual 13th Mardi Gras Conference*, Baton Rouge, Louisiana, February 3-5, 2005. http://grids.ucs.indiana.edu/ptliupages/publications/WSSPerf.pdf

[OASa] OASIS Standard, *Web Services Security: (WS-Security 2004)*, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf - Last accessed on December 15, 2009

[OASb] OASIS Standard, *WS-Trust 1.4*, http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.pdf - Last accessed on March 07, 2012

[OASc] OASIS Standard, *WS-SecureConversation 1.4*. http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.pdf - Llast Accessed on February, 29, 2012

[SAP] SAP AG, WS-SecureConversation, http://help.sap.com/saphelp_nwpi711/helpdata/en/48/aea404ac5a3206e10000000a42189c/content.htm

[Sos10] Sosnoski, D., Java Web Services: WS-Trust and WS-Secure Conversation, IBM May 2010. http://www.ibm.com/developerworks/java/library/j-jws15/index.html. Last accessed August 17, 2012.

[W3C07] W3C, *Web Services Policy 1.5 – Framework*, 4 September 2007, http://www.w3.org/TR/ws-policy/- Last accessed on March 15, 2012

[W3C08] W3C, *XML Signature Syntax and Processing (Second Edition)*, 10 June 2008, http://www.w3.org/TR/xmldsig-core/ - Last Accessed on February, 29, 2012

[W3C09] W3C, *Web Services Metadata Exchange (WS-MetadataExchange)*, W3C Working Draft 17 March 2009, http://www.w3.org/TR/2009/WD-ws-metadata-exchange-20090317/ - Last Accessed on February, 29, 2012

## Appendix I: Glossary

**Claim**: a statement about a client, service or other resource (e.g. name, identity, key, group, privilege, capability, etc.).

**Participant:** represents an entity (e.g., human, computer, message, an endpoint, interaction, resource) that is in charge of managing SCTs.

**Policy**: a collection of policy assertions that have their own name, references, and ID.

**Proof-of-Possession (PoP) Token**: a security token which contains a secret *data* parameter that can be used to prove authorized use of an associated security token.

**Security Context Token (SCT)**: a representation of a security context, which in turns refers to an established authentication state with negotiated keys that may have additional security-related properties.

**Security Token**: a collection of claims (such as X.509 certificate, Kerberos ticket, and username).It is responsible for adding signatures to tokens.

**Security Token Service (STS)**: a web service that issues SCTs by itself, or relies on another STS to do so using its own trust statement

**Signed Security Token**: a security token that cryptographically endorsed by a specific authority (e.g. an X.509 certificate or a Kerberos ticket)

**Trust Engine**: responsible for verifying security tokens and verifying claims against policies. It evaluates the security-related aspects of a message using security mechanisms