Performance of Web Services Security

Hongbin Liu Shrideep Pallickara Geoffrey Fox
Indiana University
Address Line
{holiu, spallick, gcf}@indiana.edu

1 Introduction

Web services enable application integration and data sharing in a platform neutral, language independent environment for both business and science. This increases the degree of exposure of critical resources which poses new challenges to securing data and service. The existing technologies such as VPN, firewall, NAT and SSL are examples of either intra-corporate domain or point-to-point solution, whereas compositional services and complex service invocation chain very often stand across multiple trust domains. In order to cope with the challenges, WS-Security and associated emerging standards define SOAP-level mechanisms to move security related information along with the message content. Designed to achieve end-to-end security, the new standards have also been utilized by the NaradaBrokering [19] messaging infrastructure, a features rich and values added interoperable interface to Web services. These security centered standards, however, have brought about significant overheads to the use of service. Concerns about the operational performance of Web services security are legitimate because the new suite of XML specifications significantly enlarge SOAP size especially its header size. The lately added XML security elements not only make use of more network bandwidth as SOAP transports, they also demand additional CPU cycles at both the assembly-sender side and at the processing-receiver side. Their utilization into the messaging substrate is at debate. Therefore it's desirable to be able to examine the performance issue of Web services security, and it would be considered constructive to examine it based on the specific implementation, based on the actual data gathered from these implementations.

The package we use is produced by WSS4J project under Apache. Written in Java as its name speaks, WSS4J is a fully implementation of WS-Security 1.0 OASIS standard; it also convenes active contributors for WS-SecureConversation and WS-Trust. Originally forked from Web services project, however, WSS4J is closely involved with Axis. For example, the current WS-SecureConversation implementation deals with MessageContext, an Axis construct instead of W3C Document object or Sun standardized SOAPPart. The feature of Axis affinity benefits those developers who build the Web services directly using Axis. We focus on the performance of security operations, hence decided not to answer the question of how much extra time spent on the wire because of added security information or how much time Axis engine uses to do the non-security related SOAP processing. The data presented in this article are all collected from local computation, that is, assembly and processing of SOAP documents residing at the same machine. The increased time of transportation brought about by the increased SOAP size is not WS-Security specific concern but shared by a series of WS standards that are augmenting SOAP headers.

2 Anatomy of SOAP Security Performance

Conceptually speaking, SOAP security activities consist of at least the following two categories, cryptography operations and XML processing. Cryptography operations are byte based, structure neutral and computation intensive. Under this category, we are interested in the performance differences in signing and verification, in encryption and decryption, and among different algorithms. Though all of the implemented cryptography algorithms are well studied, the data will provide a chance to confirm with the well known algorithmic properties.

XML processing is the second area that a WS security performance mind would focus on. Note that the most popular XML packages all comply with W3C Document Object Model (DOM), which provides a set of interfaces to creating, accessing and modifying both the structure and the content of document. Security related XML processing includes parsing, validation, transformation and document tree traversal. Is time spent relatively even among them or is there some which dominates? Some researchers find that canonicalization disproportionately time consuming; we are interested in collecting data to address the issue, too.

WS-Security utilizes existing XML Digital Signature and XML Digital Encryption to specify how to apply these XML general technologies to the SOAP. How to attach security tokens and a set of processing rules are also included in the specification.

We measure WS-Security by measuring time on document signing and encryption. If JCE, short for Java Cryptography Extension, measurement help to explain how different cryptography algorithms applied can make a difference to security performance, WS-Security measurement further isolates the size and the structural complexity of XML as possible independent factors.

WS-SecureConversation is based on the idea of the conversation session, and/or conversation context. It specifies how SecurityContextToken to be accepted at the beginning of and to be used during the session by communication parties. In a scenario where multiple messages are exchanged, use of SecurityContextToken will help to save sending security keys over and over again. SecurityContextToken is stored in both side's memory and referenced by its uniquely generated ID. But WS-SecureConversation recommends not use the SecurityContextToken for the entire session but to derive a series of keys from the initial token and use these keys in each round of message exchange. The communication parties only need to trade the derivation needed information such as nonce value and key derivation algorithm.

The conversation timing data items presented in this article cover time used for both sender and receiver processing. But again, the time used for the wire transportation is not included. We are interested in how much time needed to conduct a security equipped conversation without considering the network distance between the two parties.

3 Experimental Results

We introduce the results found in the four experiments intended to investigate the performance of Java cryptography operation, SOAP XML processing, WS-Security implementation and WS-SecureConversation implementation.

3.1 Experimental environment

The data is collected running code on a machine of a single Pentium 4 CPU 2.79 GHz with 768 mega bytes main memory. Operating system is Linux kernel version 2.4.10, Hotspot Client JVM with JRE version J2sdk1.4.2 and JCE provider BouncyCastle. The data is verified at a Pentium 4 Windows XP to be alert at any possible gross differences. Each data entry of the all four datasets described below is obtained as average milliseconds over 100 runs. These numbers are obtained as integer value unless otherwise obviously they are not. In that case, they are floating point doubles.

3.2 Cryptography operations results

Table 1 and 2 list the results of cryptography operations over the byte arrays of increasing size. These byte arrays are filled with randomly generated byte values to neutralize any possible effect that input distribution can have on the experiment. The results shown in the tables confirm with some of well-known properties of algorithms in question.

- In RSA verification is much faster than signing; in DSA signing is faster than verification.
- SHA1 is noticeably slower than MD5.
- Symmetric encryption is faster than asymmetric ones.

The results show that symmetric encryption algorithms are in general faster than to be always recognized by the low resolution timing[1] machinery JVM until data size nears 10 kilo bytes. After the input length reaches 10 kilo bytes, times grow linearly with input data size. Other than these data, we also measured digestion, asymmetric encryption, and phash algorithm. Phash algorithm is being used by Apache WSS4J to derive keys in conversation context. For keys with size as large as several kilo bytes, it takes 2~6 milliseconds to derive a new key. Asymmetric encryption is measured with RSA for data less than 1000 bytes; as it practically is used to encrypt and decrypt keys instead of real data. Digestion takes recognizable times only when the array size goes beyond 1 mega bytes and we found MD5 is faster than SHA1. The latest breakthrough development regarding MD2 and MD5 will probably make the measurement on these two algorithms less meaningful, but these are not benchmarking data; they can be used just as reference.

Table 1: Signing and Verification with Different Algorithms

	MD5WithRSA		**SHA1WithRSA**		**SHA1WithDSA**	
	Sign	**Verify**	**Sign**	**Verify**	**Sign**	**Verify**
50	18	1.47	19	1.19	1.17	14
100	20	1.31	19	1.31	7	12
1000	19	1.62	19	1.47	7	13
10000	19	1.31	19	1.41	7	13
100000	18	2	20	2	9	16
1000000	27	9	37	22	34	41
10000000	109	89	225	210	274	278

[1] It should be noted that where zero appears as data values, it doesn't mean no time is taken; it only means time taken is so small that the resolution scale in the measurement records it as zero.

Table 2: Encryption and Decryption with Symmetric Algorithms

	AES		Blowfish		DES		TripleDES	
	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt
50	0.15	0.31	0.31	0.47	0.31	0.31	0.62	0.47
100	0	0.31	0.47	0.31	0.32	0.32	0.47	0.46
1000	0.16	0.15	0.63	0.31	0.31	0.47	0.78	0.62
10000	0.47	0.63	0.94	0.62	0.77	1.41	2.81	3.12
100000	3	3	5	6	9	9	25	26
1000000	37	40	55	59	90	99	260	270
10000000	426	470	601	668	953	1057	2643	2746

3.3 XML processing results

The hypothesis is that the XML processing time is dependent on both the size and the complexity of document structure. Number 1 to 5 along the X-axis in the Figure 1 represents SOAP documents with identical structure. The only difference among them is the XML of bigger number has a ten times larger text node than the one of smaller number. So document 5 is the largest file of 1,369,230 bytes. XML from 6 to 25 can be divided into two groups. The first group numbered from 6 to 15 contains XMLs that have increasing number of nested elements. XML from 16 to 25 has an increasing number of sibling elements. Documents in both groups start with 500 elements and increase 100 for the next one; the sizes of them range from 7 to 30 kilo bytes. X-axis in Figure 2 represents the same set of SOAP document, that is, from 1 to 5, identical structure but increasing file length; from 6 to 15, increasingly nested nodes and from 16 to 25, increasingly sibling nodes. In order to present timing data effectively, small Y-axis values at XML 1 to 4 are omitted from Figure 2.

XML canonical form speaks about documents logically equivalent within an application context can differ in physical representations based on XML permissible syntactic changes. These changes, for example, can be attribute ordering, entity references or character encoding. XML canonicalization is a process of applying standard algorithms to generate a physical representation of XML document. To XML security, now there is a standard mechanism to produce an identical input to digestion procedure prior to both signature generation and signature verification. Given its necessity, the speed of canonicalization will have imprint on the overall performance of SOAP security.

The following observations can be made on the two datasets.

- The size of the document is the most important factor to SOAP process time; but the complexity of structure also matters.
- Subtree canonicalization is substantially faster than XPath canonicalization and is less sensitive to the XML structural changes than XPath is.
- For documents less than 1 mega bytes, parsing takes at most 50~60 milliseconds, whereas canonicalization takes less than 100 milliseconds.

Considering that the implementation we use is still in its active development, it is likely these numbers could be more favorable in the near future. On the other hand, the messaging security may not mandate c14n processing as those of end services do. Several notes can be made to address this issue in messaging systems. Locate and analyze those cases where no change is made to the document along the relay path so canonicalization is unnecessary; use substree instead of XPath canonicalization as much as possible; improve the performance of XPath canonicalization including the performance of XPath implementation itself.

Figure 1: Measurement of SOAP Parsing

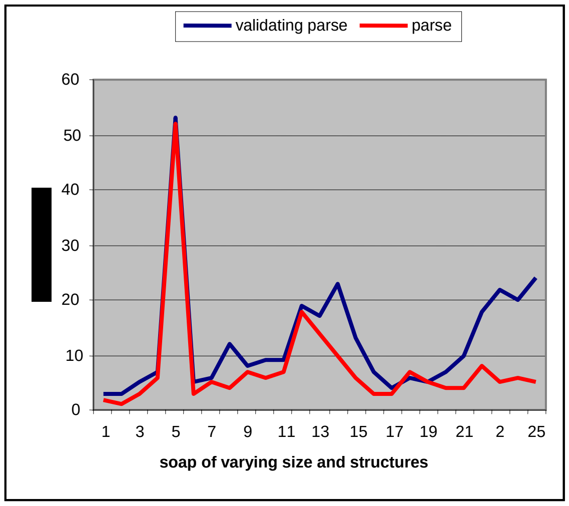
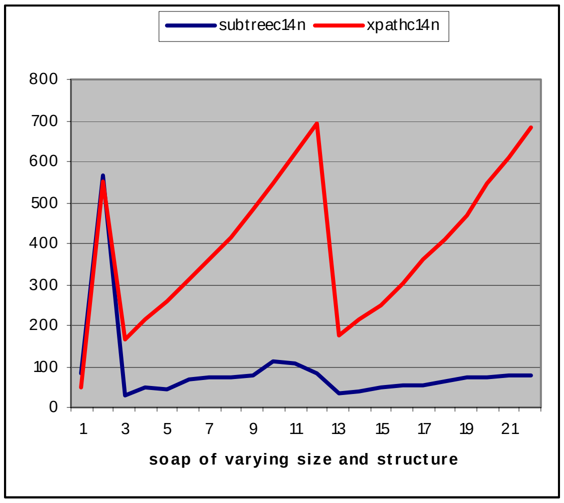


Figure 2: Measurement of Canonicalization



3.4 WS-Security results

SOAP documents used in this experiment are the same as in section 3.4; although we remove the largest document 5 to make comparison among series in the plot more effective. Data row for that entry is [8451, 2136, 1859, 1890, 1316]. Five entries represent sign and encrypt together, signing, verification, encryption and decryption. Half of the nested document from 11 to 15 in the above is not shown, either, due to an unexpected StackOverflowError[2]. Interests in the experiment are to investigate differences between signature generation, signature verification, encryption and decryption. WS-Security allows great flexibility with *SecurityToken* and *KeyInfo* representations, which are two XML constructs defined in Web services security specifications, so we test with them, as well as signing and encryption algorithms. The significant differences among them, however, are not found. For example, the choice of X.509 key identifier, issue serial and binary security token doesn't affect encryption decryption speed. Using password, either in text or digest, is much faster than generating the keys on the fly. In the other aspect, algorithmic significance continues to be measurable. The comparison between RSA and DSA in this experiment is in accordance with the knowledge introduced in section 3.2. Signature generation in DSA is faster than signature verification while in RSA algorithm signature verification is much faster than signature generation. The difference should be considered in applications where signing and verification have unequal weights. As shown in Figure 3, signing is more costly than encryption. And if both signing and data encryption are performed, it increases not only security operational

[2] It is thrown on Windows platform when the number of XML nodes reaches near 1200, nested from the first to the last. Speculation is that memory tree of Dom implementation can cause the error.

time but also the complexity of XML. The top curve in Figure 3 shows the time for the same SOAP first signed (verified) and then encrypted (decrypted).

Figure 3: SOAP WS-Security Operations

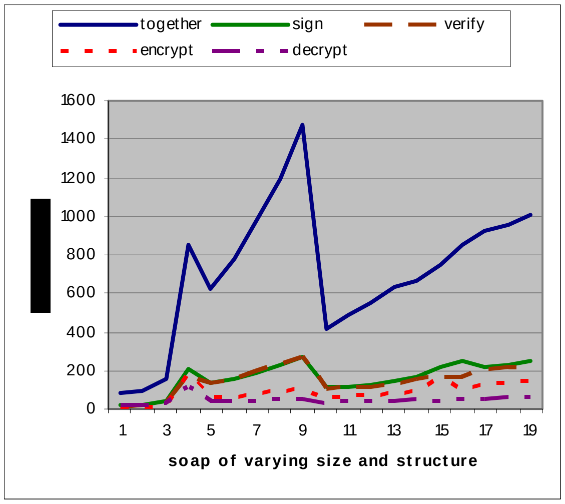
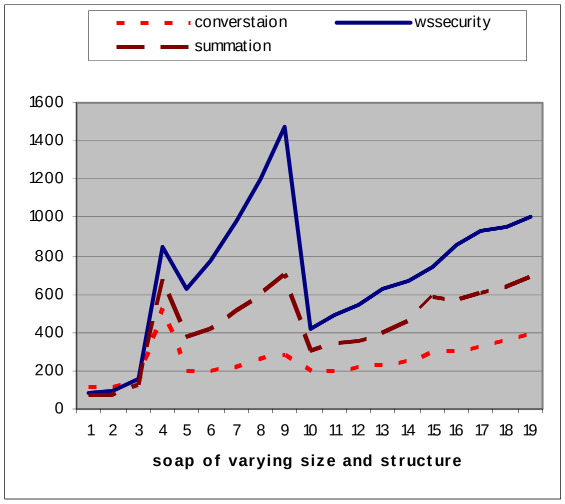


Figure 4: WS-SecureConversation



3.5 WS-SecureConversation results

In Figure 4 we find that WS-SecureConversation is indeed more efficient. With multiple message exchange, in the experiment, as simulated by having the same SOAP continuously undergoing several security operations, WS-SecureConversation is noticeably faster than normal WS-Security as the former doesn't generate symmetric keys every time[3]. The middle curve is the summation of signing, verification, encryption and decryption, using the fast algorithms. We should point out that the superiority of WS-SecureConversation may be a litter exaggerated, because in the open source package used in the experiment both signing and encryption of data use the same *SecurityContextToken* to derive the keys. Different tokens are recommended in WS-SecureConversation specification; but the feature is not currently implemented in the package.

4 Related Work

In [1] several Grid services packages including SSL are compared for the speed of their security modules. Its data items cover both the security processing time and the packet traveling time in the network. Canonicalization is singled out as the performance bottleneck of the message level SOAP security operation. [2] is a formal study of modeling security objects as language primitives within SOAP header. [3] combines parsing techniques, in-memory data structures and transportation paradigm for the design and implementation of SOAP tuned for scientific arrays. [4] is another performance comparison between SOAP, JavaRMI and CORBA using both open source and corporate implementation packages. Both [1], [3] and [4] are attentive to SOAP as a communication protocol; [3] and [4] are not security related investigation; [1] focus on signing but data encryption is not covered.

[3] A key is derived from a session key each time message is processed.

5 Discussion and Future Work

SOAP level security involves computational intensive operations of cryptography and memory demanding XML DOM processing. This work of investigation intends to understand the performance of SOAP security in terms of its dependability on both the implemented cryptography in programming language Java and the properties of input documents, which we focus on the size and the complexity of structure. Different algorithms of cryptography and various kinds of key references in XML are introduced into comparison, too.

There are some works that can be considered in the future. For example, SOAP in the experiments uses simple data types. When new and complex data types and serializers converting XML value to Java object are introduced, it would be interesting to see if there will be performance problems and what kind. Secondly, we don't set the XML parts to be encrypted or signed as parameter; SOAP body is used in all the XML experiments. To further fine control in that aspect is as well meaningful. Lastly and probably the most importantly, the tests we conduct in this article point to several hopeful places in implementation regard to boost Web services security performances. We suggest Document Object Model should be among the first to look at. XPath affects canonicalization speed; the overall security speed will increase if XPath implementation improves.

6 Conclusion

It takes less than 10 milliseconds to sign or encrypt up to an array of 100 kilo bytes, but it takes about 100~200 to perform the security operations for SOAP. The length of the XML matters, so does the complexity of XML structure.

7 References

[1] S. Shirasuna, A. Slominski, L. Fang, and D. Gannon. *Performance Comparison of Security Mechanisms for Grid Services*. Proceedings in *5th IEEE/ACM International Workshop on Grid Computing*. 2004.

[2] A. Gordon and R. Pucella. *Validating a Web Service Security Abstraction by Typing*. Proceedings in *the 2002 ACM Workshop on XML Security*. 2002.

[3] K. Chiu, M.Govindaraju and R. Bramley. *Investigating the Limits of SOAP Performance for Scientific Computing*. Proceedings in *the 11 th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*. 2002.

[4] D. Davis and M. Parashar. *Latency Performance of SOAP Implementations*. Proceedings in *the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. 2002.

[5] G. Fox, S. Pallickara, and S. Parastatidis. *Towards Flexible Messaging for SOAP Based Services*. Proceedings in *the IEEE/ACM Supercomputing Conference*. 2004. Pittsburgh, PA.

[6] J. Kangasharju, S.Tarkoma, and K. Raatikainen. *Comparing SOAP Performance for Various Encodings, Protocols, and Connections*. Proceedings in *Personal Wireless Communications*. 2003. Venice, Italy.

[7] S. Pallickara, M. Pierce., G. Fox, Y. Yan, andY. Huang. *A Security Framework for Distributed Brokering Systems*. Technical Report, 2002, Community Grid Lab, Indiana University.

[8] V. Welch, F. Siebenlist., I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. *Security for Grid Services*. Proceedings in *the Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*. 2003.
[9] http://java.sun.com/j2se/1.4.2/docs/guide/security/CryptoSpec.html.
[10] *Java Cryptography Extension*. http://java.sun.com/products/jce/doc/guide/API_users_guide.html.
[11] *Apache WSS4J Project*. http://ws.apache.org/ws-fx/wss4j.
[12] *SOAP 1.1*. http://www.w3.org/TR/2000/NOTE-SOAP-20000508.
[13] *Document Object Model Level 2*. http://www.w3.org/TR/DOM-Level-2-Core.
[14] *Document Object Model Level 3*. http://www.w3.org/TR/DOM-Level-3-Core.
[15] *SOAP 1.2*. http://www.w3.org/TR/soap12-part1.
[16] *BouncyCastle*. http://www.bouncycastle.org.
[17] *Exclusive XML Canonicalization*. http://www.w3.org/TR/xml-exc-c14n/.
[18] *GT3 security performance evaluation*. http://www-unix.globus.org/toolkit/3.1/ogsa/docs/security_performance.html.
[19] *NaradaBrokering*. http://www.naradabrokering.org.