RT0546 November 30, 2007
Other

IBM Research Report

Implementation and Performance of WS-Security

Satoshi Makino, Kent Tamura, Takeshi Imamura, Yuichi Nakamura
IBM Research Division
Tokyo Research Laboratory
1623-14, Shimo-tsuruma
Yamato, Kanagawa 242-8502
Japan

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Implementation and Performance of WS-Security

Satoshi Makino, Kent Tamura, Takeshi Imamura, and Yuichi Nakamura

{mak0702,tkent,imamu,nakamury}@jp.ibm.com

IBM Tokyo Research Laboratory

1623-14, Shimo-tsuruma, Yamato, Kanagawa 242-8502 Japan

ABSTRACT

This article describes performance improvements for the Web Services Security (WS-Security) specification. In the course of its development and performance measurement, we identified bottlenecks in the XML parsing and public key operations such as RSA signature and encryption. We are working on minimizing the impact of both of these bottlenecks. We implemented a stream-based WS-Security processor and showed its efficiency in XML parsing, in terms of both the processing time and the memory usage. In addition, we introduced the Web Services Secure Conversation (WS-SecureConversation) to avoid expensive public key operations.

KEYWORDS

Web services, security, WS-Security, performance

INTRODUCTION

As Web services become widely accepted, there are strong demands to provide services that are secure and which can be used in a secure manner. However, Web services standardization processes usually take a bottom-up approach, and the Simple Object Access Protocol (SOAP) specification (Simple, 2000), which is used for most Web services, does not define these security-related functions.

Web services often employ intermediary nodes between clients and service providers. In order for the exchanged messages to be protected against altering or eavesdropping even by these intermediaries, it is insufficient to protect only the connections between each node. The message path should be protected all the way through, beginning from the client to the service provider. To do this, we cannot simply rely on the security mechanisms that are provided in the transport layer, and mechanisms need to be added to the SOAP messages themselves. Message-level

security makes it possible for specific parts of the SOAP messages (e.g. credit card numbers) to be digitally signed or encrypted and provides users with fine-grained security control for the messages.

This article describes our implementation of the WS-Security specification that was written to protect SOAP messages. We also point out some performance bottlenecks and propose some possible solutions for the bottlenecks.

WS-SECURITY SPECIFICATION

The Web Services Security (WS-Security) specification ("Web Services Security", 2002) is intended to realize message-level security for the exchange of SOAP messages and is undergoing standardization by the OASIS Web Services Security Technical Committee. This section outlines the specification.

Overview

The WS-Security specification defines mechanisms such as digital signatures and encryption so that the SOAP messages will be protected. This specification also defines the syntax for encoding the arbitrary format of the security tokens that are to be embedded into the messages.

Security tokens

A security token represents the abstract concepts such as name, key, and privileges owned by the message sender. The WS-Security specification defines a token containing the username and password (a UsernameToken) and one containing binary formatted objects such as X.509 certificates and Kerberos tickets (a BinarySecurityToken).

Digital signature and encryption

The syntax for digitally signing and encrypting SOAP messages is based on the W3C Recommendations for XML signature ("XML Signature", 2002) and XML encryption ("XML Encryption", 2002), respectively. The syntax of digital signatures and encryption according to the WS-Security specification is shown in Figure 1.

```
<S:Envelope
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <S:Header>
```

```
        <wsse:Security>
            <wsse:BinarySecurityToken valueType="wsse:X509V3">
                <!-- Security token -->
            </wsse:BinarySecurityToken>
            <xenc:EncryptedKey>
                <!-- The key used for encryption -->
            </xenc:EncryptedKey>
            <ds:Signature>
                <!-- Digital signature on (some part of) message -->
            </ds:Signature>
        </wsse:Security>
    </S:Header>
    <S:Body>
        <xenc:EncryptedData>
            <!-- Encrypted message -->
        </xenc:EncryptedData>
    </S:Body>
</S:Envelope>
```

Figure 1: Syntax of WS-Security

In Figure 1, the <wsse:BinarySecurityToken> element contains an X.509 certificate and it is referred to by the <ds:Signature> element for the signature verification.

Other functions

The WS-Security specification also defines the format of the message timestamps, the method to send the passwords securely with nonces and timestamps, and so on.

IMPLEMENTATION

We implemented some of the functions defined in the WS-Security Specification, such as the digital signature, encryption, UsernameToken, and so on. This implementation is bundled within IBM WebSphere Application Server, V5.0 (WAS V5; http://www.ibm.com/software/info1/websphere/index.jsp) as released in November 2002. This section describes the implementation.

Architecture and procedure

The Web services implementation in WAS V5 is based on the JSR 109 specification (JSR, 2002),

intended for the integration of Web services and J2EE. The SOAP processor is implemented as a servlet, and it interprets the incoming messages and invokes the appropriate Web services implementations. The actual services are implemented as EJBs according to the JSR 109 specification, and our implementation utilizes a “handler” mechanism as also defined in the JSR 109 specification. Our overall conceptual design is shown in Figure 2, where the rectangles with dotted borders represent sub-components.

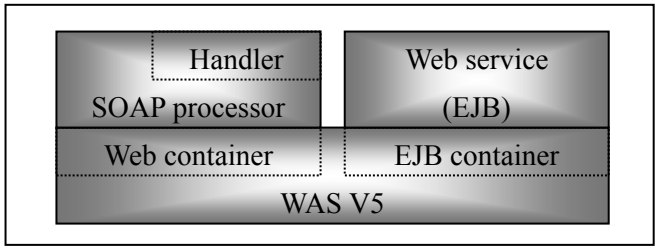

Figure 2: Web services architecture in WAS V5

A handler is a mechanism for performing additional processes on messages. For example logging and caching are possible as additional processes, and the signatures and encryption that we implemented are also packaged in handlers. Handlers are invoked just before a message is sent or just after a message is received. Using this mechanism, messages are signed and/or encrypted just before they are sent, and they are verified and/or decrypted just after they are received. The insertions and verifications of security tokens are also performed by these same handlers. Since this procedure is performed in the same way for request and response messages, our WS-Security handlers are invoked four times in a single message exchange. These procedures are summarized in Figure 3, where arrows and gray rectangles show the message flows and WS-Security handlers, respectively.

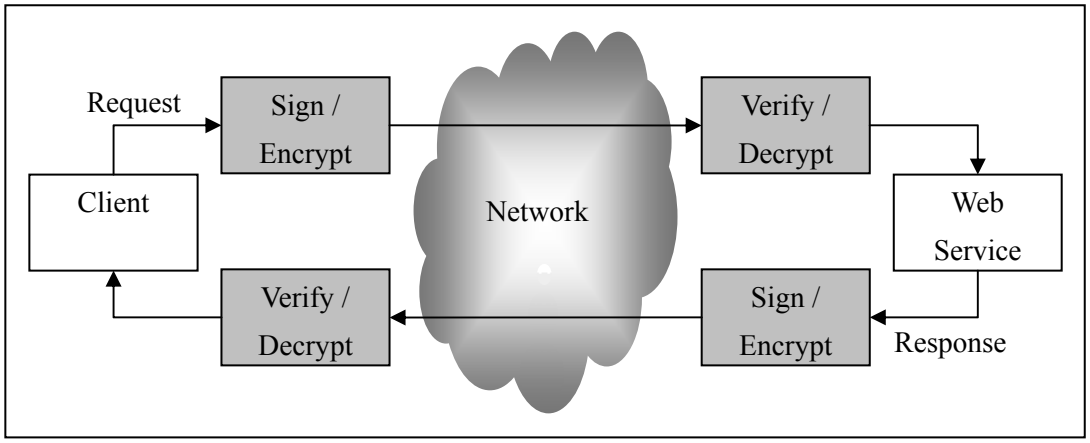

Figure 3: WS-Security handler procedures

Relation to operational environment

The WS-Security handler in WAS V5 passes usernames and passwords in the security tokens to the authentication mechanism in WAS V5. The authentication mechanism validates these values according to the specified user registry such as the local OS user registry or the LDAP directory. If the authentication succeeds, the mechanism attaches the user's ID to the Web service execution thread and the Web service is run under the user's privilege. Otherwise the mechanism throws an exception and the WS-Security handler propagates the exception back to the client in the form of a SOAP fault.

PERFORMANCE IMPROVEMENTS

While creating the WS-Security implementation we found several problems that may lead to performance bottlenecks. This and the following sections discuss some of them.

Motivations

At first, our aim was to provide proof-of-concept implementations of XML- and Web-services-security-related specifications for the market as soon as possible. This was sufficient as long as the Web services were only for early adopters and testers who wanted to make sure that the new concepts actually met their needs. However, as Web services became widely accepted, performance has become an increasing focus of concern. In order for the Web services applications to replace the existing non-Web-services applications such as e-commerce systems, we must show that Web services can achieve security and performance at least equal to the existing applications. While security can be assured to some degree as long as the implementation conforms to the specification, performance largely depends on the way it is implemented. With these concerns we started to measure the performance of our WS-Security implementation to find the bottlenecks.

Measurements

As a first step we measured the performance of the WS-Security handler with various signature and encryption algorithms. The test code accepts a Java InputStream object that represents a SOAP message as its input, processes the WS-Security (i.e., signature and encryption) on the message and discards the result. The sample SOAP message used in the measurements is shown in Figure 4. The performance measurements throughout this article were conducted using a Pentium-4 2 GHz workstation with 2 GB of memory.

```
<SOAP-ENV:Envelope
```

```
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <SOAP-ENV:Body>
        <exxxxxx>
            <e0xxxxx>
                <e00xxxx>
                    <e000xxx>t000xxx</e000xxx>
                    <e001xxx>t001xxx</e001xxx>
                </e00xxxx>
                <e01xxxx>
                    <e010xxx>t010xxx</e010xxx>
                    <e011xxx>t011xxx</e011xxx>
                </e01xxxx>
            </e0xxxxx>
            <e1xxxxx>
                <e10xxxx>
                    <e100xxx>t100xxx</e100xxx>
                    <e101xxx>t101xxx</e101xxx>
                </e10xxxx>
                <e11xxxx>
                    <e110xxx>t110xxx</e110xxx>
                    <e111xxx>t111xxx</e111xxx>
                </e11xxxx>
            </e1xxxxx>
        </exxxxxx>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 4: Sample SOAP message structure

The results of the measurement on the WS-Security sender and receiver are shown in Figure 5 and Figure 6 respectively. Here X- and Y-axes denote the combination of signature and encryption algorithms and the processing time in milliseconds, respectively. “RSA_3DES” encryption means that the XML content is encrypted with a randomly generated 3DES key and

the key is in turn encrypted with the RSA key.

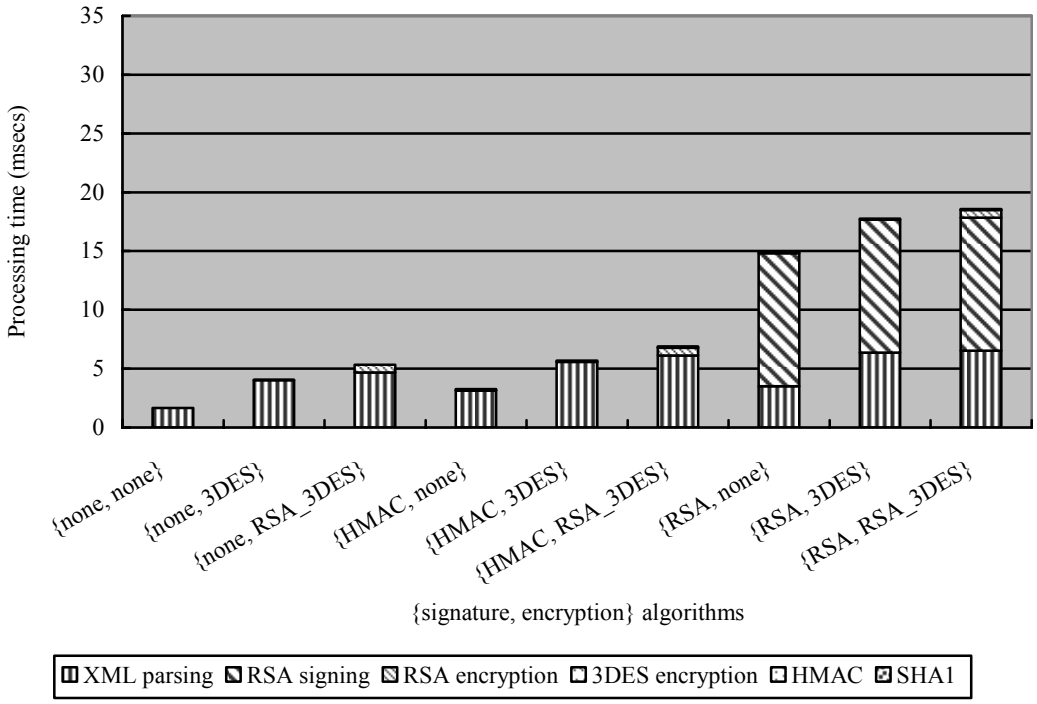

Figure 5: Sender-side WS-Security performance

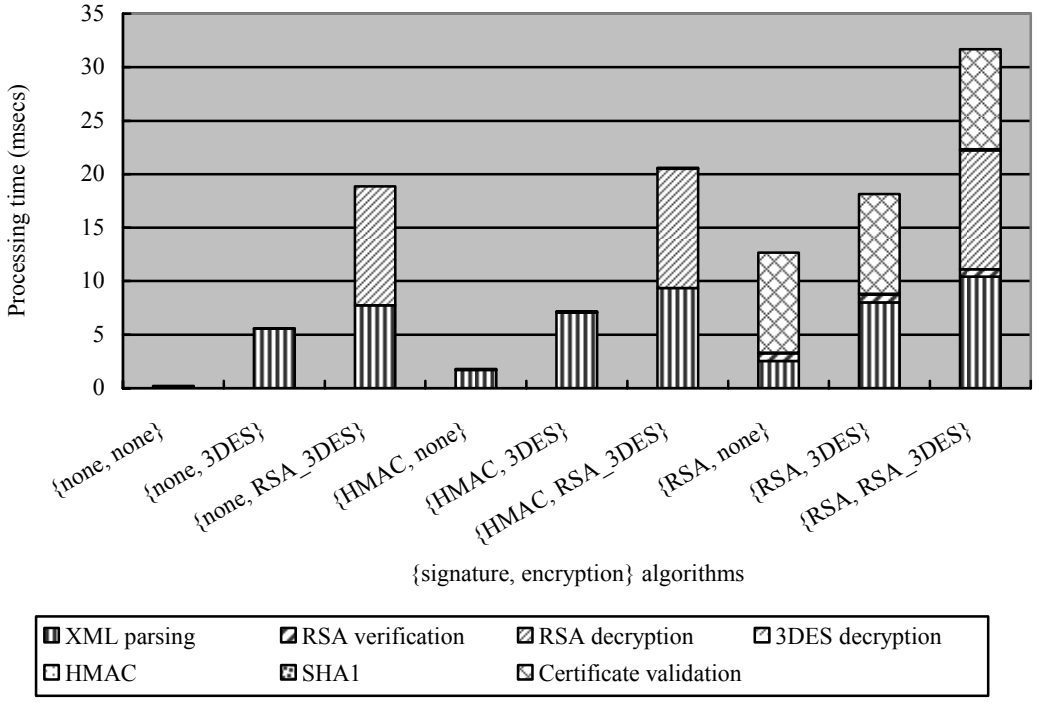

Figure 6: Receiver-side WS-Security performance

Observations

We can see that most of the processing time is occupied with a set of several operations. Here we discuss each of them.

First, XML parsing requires a certain amount of time for both the sender and the receiver. This is mainly because our code constructs a Document Object Model (DOM) tree (Document, 2000) of the entire SOAP message in memory. While DOM is easily manipulated for signature and encryption, it is inefficient in terms of both time and memory usage. It is also inefficient when the message is very large. In order to solve these problems, we are implementing a stream-based WS-Security processor and trying to avoid this inefficiency. The stream-based processor is described in detail in the next section.

On the other hand, RSA signing and RSA decryption consumes much more time than the other signature and encryption algorithms. This is because of the nature of the RSA algorithm, which involves many modulus calculations. Since the parameters in the RSA private keys are usually much larger than the ones used for the public keys, RSA private key operations (i.e. signing and decryption) require much more time than the public key operations (i.e. verification and encryption). In order to avoid this overhead, we can establish a secure context using the public key pairs of the parties in the message exchange, derive a shared secret from the secure context, and use that for signature and/or encryption. As shown in Figure 5 and Figure 6, shared key operations such as 3DES and HMAC cost very little, and can achieve good performance. The specification named WS-SecureConversation is proposed to meet these requirements. This specification is described after the section on stream-based processing.

In addition, certificate validation also consumes a lot of time at the receiver side. When an RSA signature is used, an X.509 certificate is embedded in the message for the use in the signature verification, as shown in Figure 1. In order to trust the certificate, it is not enough to validate the certificate alone but the trusted chain of the certificates including the incoming certificate must be established. Our WS-Security code uses Java CertPath API to validate the incoming certificates. Inevitably the validation process involves many cryptographic operations and should be avoided as much as possible. In order to meet this goal, WS-SecureConversation can be utilized as well, because it can eliminate the usage of the RSA signing (and therefore the certificate validation).

STREAM-BASED PROCESSOR

We have implemented a prototype of a stream-based WS-Security processor in order to avoid the inefficiency of DOM. This section describes the details of the processor, including the achieved performance improvements.

Overview

We designed the stream-based processor based on the new event model, instead of relying on DOM. By adopting a stream-based architecture, we expected to achieve efficiency in terms of both time and memory even when the message is very large.

On the receiver side, an event-driven XML parser is used and our processor performs WS-Security processing on the message as a parsing event (i.e. start or end of the element) is received from the parser, without constructing a DOM. The processed event-stream is passed to the underlying SOAP engine.

On the sender side, the processor is integrated into the serialization code in the SOAP engine that writes the generated SOAP message object into the network. Our processor intercepts the serialization event (i.e. start or end of the element) and performs WS-Security processing on the message as the event is received, without constructing a DOM. The serialization code in the SOAP engine writes out the WS-Security-processed SOAP message to the network.

In the course of designing the event-based API, we found a problem in character code conversions. The SOAP messages exchanged via networks are in the format of byte arrays with encoding indicators. This format is inconvenient for manipulation and UTF-16 formatted characters and strings are used inside the Java Virtual Machine (JVM). However, the XML canonicalization (Canonical, 2001) used for XML signatures requires its output to be UTF-8. Therefore, SOAP messages are converted from byte arrays to UTF-16 strings by the XML parser, and then converted to UTF-8 by the XML signature processor. Character code conversion generally requires memory allocation proportional to the size of the original data and this may cause performance degradation, especially when the message is large. We concluded that just using the Simple API for XML (SAX; http://www.saxproject.org/) was not enough because it uses Java UTF-16 strings, and so we implemented the processor to avoid the unneeded character code conversions.

Measurements

We measured the performance of our stream-based WS-Security processor under the same conditions as the measurements shown in Figure 5 and Figure 6. For the sender-side measurement, we prepared a dummy serializer that emits serialization events from an InputStream object. The results of the measurements on the sender and receiver are shown in Figure 7 and Figure 8 respectively.

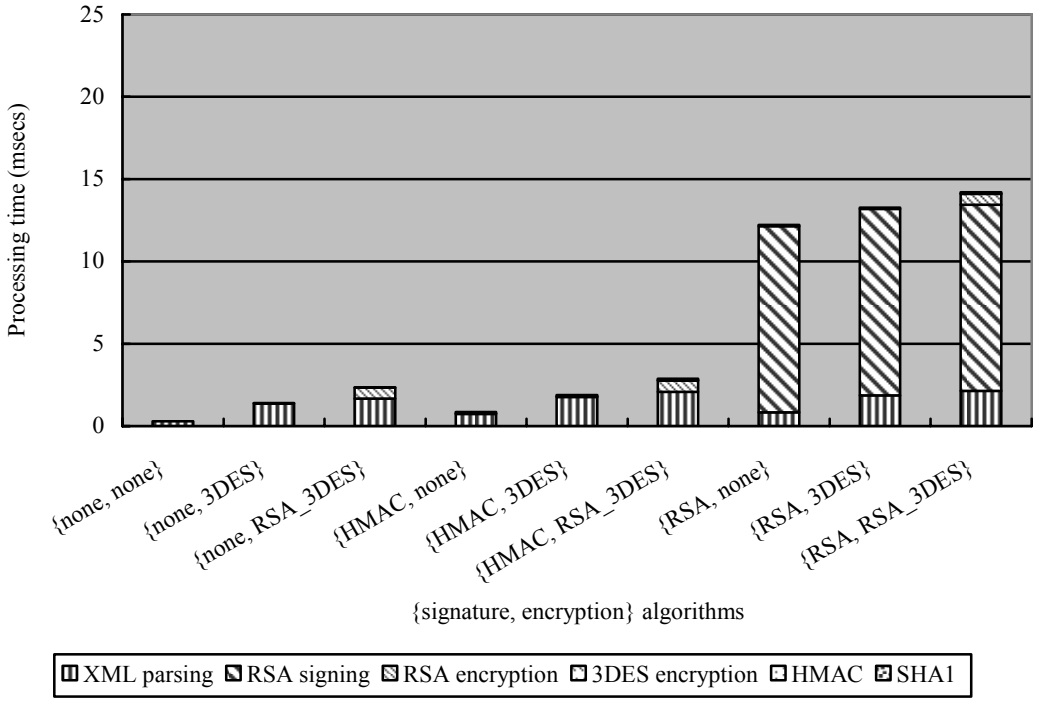

Figure 7: Sender-side stream-based WS-Security performance

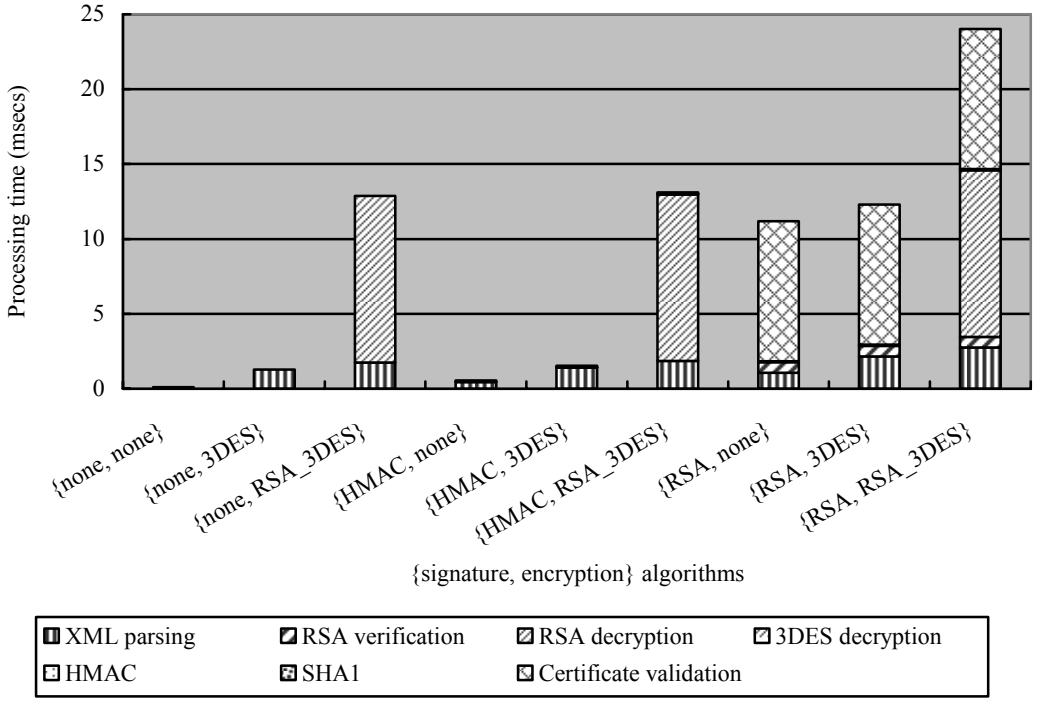

Figure 8: Receiver-side stream-based WS-Security performance

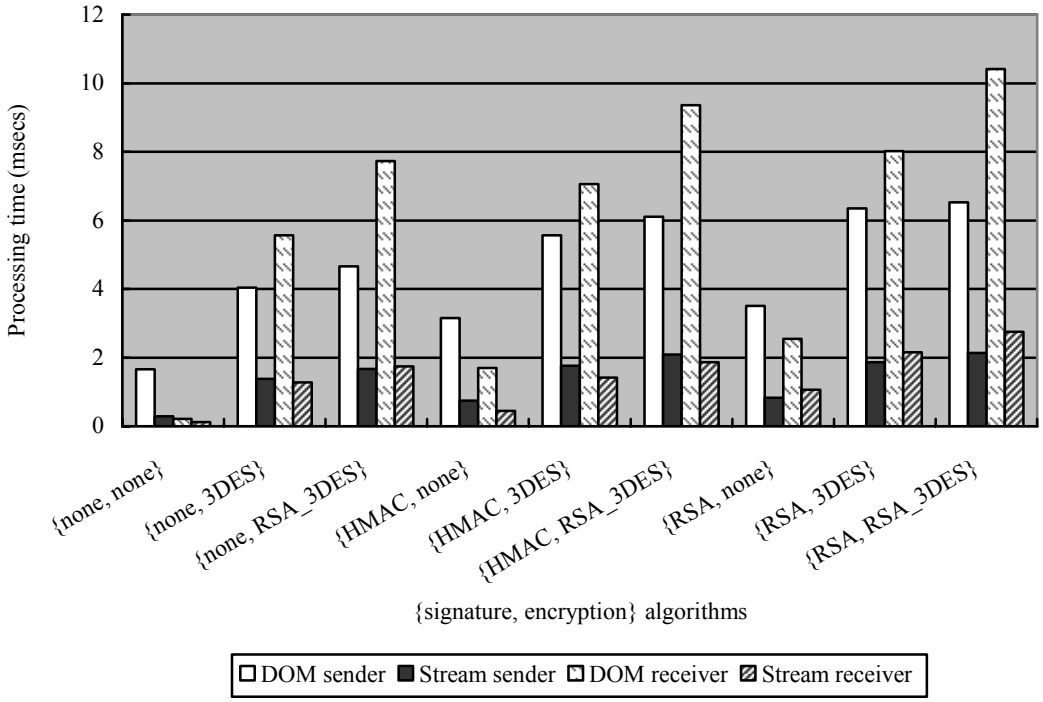

Figure 9: Comparison of the times to parse XML messages

The time for the cryptographic operations such as signing and encryption can be considered to be constant regardless of the type of the processor (i.e. DOM- or stream-based), because they just accept a raw byte array as an input and return a signed or encrypted byte array as an output in both cases. Therefore we compare the "XML parsing" part from Figures 5, 6, 7 and 8. The results are shown in Figure 9.

In order to show the scalability of the stream-based WS-Security processor, we also measured its processing time and memory usage with various message sizes. Figure 10 and Figure 11 show the processing time and the memory usage respectively. Here the X-axis denotes the size of SOAP message to be processed, in bytes. The HMAC signature and 3DES encryption are used, and the memory usage data is collected from the garbage collection log in the JVM.

Observations

Figure 9 tells us that a stream-based processor can achieve good performance with all of the combinations of signature and encryption algorithms, on both the sender and receiver sides. The ratio of the improvement varies, from 64-82% on the sender side and from 46-80% on the receiver side. We believe these improvements come from avoiding the DOM construction and character code conversions. Therefore it can be said that the relative significance of parsing the XML messages in the overall processing time was greatly decreased, and the overhead in the

cryptographic operations is becoming dominant.

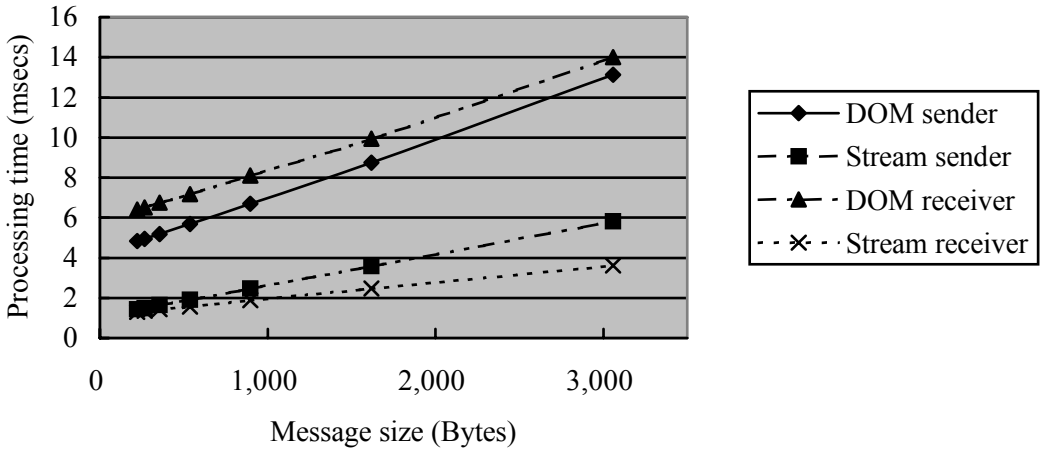

Figure 10: Processing time with various message sizes

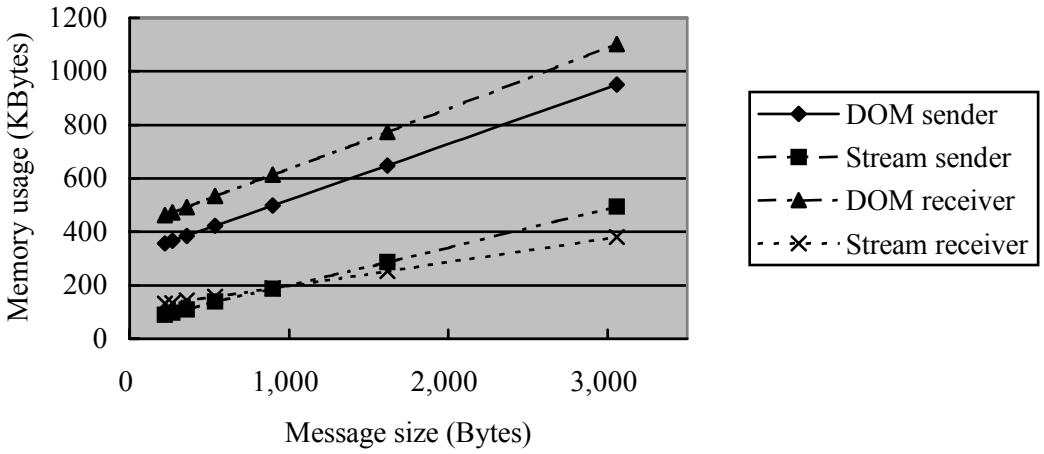

Figure 11: Memory usage with various message sizes

As for the scalability, the results shown in Figure 10 and Figure 11imply that the stream processing achieves a better scaling factor in terms of both the processing time and the memory usage. This is summarized in Table 1. The improvements at the receiver side are more significant than on the sender side, because the DOM-based receiver has to parse the XML from a byte array and construct a DOM twice, when the SOAP message is received and when the encrypted message is decrypted. While the DOM-based sender parses the XML and constructs a DOM once, the stream-based receiver does not construct a DOM at all. That is the reason why

the stream-based receiver can offer better scalability.

Table 1: Percentage of improvements in terms of their scaling factors

	Processing time	Memory usage
Sender	46.9%	31.9%
Receiver	69.7%	61.1%

In general, it can be concluded that the stream-based approach is very promising in terms of both performance and scalability, although the current implementation is a prototype and it leaves much room for refinement and tuning (Tamura et al., 2003).

WS-SECURECONVERSATION

In order to avoid expensive public key operations, the Web Services Secure Conversation (WS-SecureConversation) specification ("Web Services Secure", 2002) can be used. This section quickly outlines the specification and shows its effectiveness in improving WS-Security performance.

Motivations

WS-Security is often compared to SSL (Freier et al., 1996) in that both are designed to provide security in the data exchange. As stated in the Introduction, although there is a big difference in that WS-Security is at the message level while SSL is at the transport level, we have to convince customers that WS-Security can be a replacement for SSL at a reasonable cost in order to encourage the wide adoption of WS-Security.

However, the overhead of SSL data transfer is on the order of 0.1 ms (Apostolopoulos et al., 1999). As shown in Figure 5, this is far lower than the overhead of WS-Security with RSA, which is an algorithm often used for XML signature and encryption. This is because shared key encryption is employed in SSL data transfer. The SSL protocol is divided into two parts: handshake and data transfer. The handshake establishes a secure context containing a shared secret value using the public keys of the sender and the receiver. Then the data transfer uses the shared key which has been derived from the secure context. Therefore, the expensive public key operations are needed only in the handshake phase and the data transfer can be performed with a very low cost, using the shared key.

WS-SecureConversation overview

WS-SecureConversation was proposed to define a message format for the secure communication under the secure context. Specifically, it defines the methods for establishing a

Security Context which corresponds to the secure context in SSL and deriving secret keys from the context. This specification was submitted on December 18, 2002 by IBM, Microsoft, RSA Security, and VeriSign. It will next be submitted to a standards body such as OASIS

The Security Context is represented as a kind of security token and can be established by one of the following three methods, according to the specification.

- The context can be created by a trusted third party (called Security Token Service), or
- It can be created by one of the communicating parties and propagated to the other parties, or
- It can be created through the negotiation between the communicating parties.

```
<S:Envelope xmlns:S="..." xmlns=".../secext" xmlns:wsu=".../utility">
    <S:Header>
        ...
    </S:Header>
    <S:Body>
        <RequestSecurityTokenResponse>
            <RequestedSecurityToken>
                <wsse:SecurityContextToken>
                    <wsu:Identifier>uuid:...</wsu:Identifier>
                </wsse:SecurityContextToken>
            </RequestedSecurityToken>
            <RequestedProofToken>
                <xenc:EncryptedKey Id="newProof">
                    ...
                </xenc:EncryptedKey>
            </RequestedProofToken>
        </RequestSecurityTokenResponse>
    </S:Body>
</S:Envelope>
```

Figure 12: A sample SOAP message containing a Security Context token

Error! Reference source not found. Figure 12 shows a sample message that carries an established Security Context token. The <wsse:SecurityContextToken> element represents the Security Context and the <RequestedProofToken> element contains the secret data for the Security Context. After this Security Context token is shared between the parties, they can

communicate securely using the derived secret key from the Security Context.

The WS-SecureConversation framework can be used for our purpose of improving the performance of message exchanges. That is, first the communicating parties can exchange their information such as their public keys, and then they create a Security Context from the exchanged information. After the Security Context is established, they can derive the shared key for the context and securely communicate with each other. In this way, the expensive public key operations are performed only once in the Security Context establishment phase, and high performance shared key operation are used afterwards. This approach enables WS-Security operations based on the shared key signature and encryption, and its performance is expected to be as high as the "{HMAC, 3DES}" column in Figure 7. We are now implementing the WS-SecureConversation as an add-on to our WS-Security processor. Once the implementation is done, the ratio of WS-Security performance to the normal Web services performance will be closer to the ratio of SSL performance to the normal HTTP performance (He, 2003). We recognize that the performance is critical in product environments and customers will not adopt the system and switch from the existing applications to Web services unless they can be sure that the Web services can achieve security and performance as good as the existing applications. In that sense, we consider the implementation of WS-SecureConversation to be very important and we firmly believe it will contribute greatly to the wide deployment of Web services.

CONCLUSION

We showed performance improvements for a WS-Security processor. Through the performance measurements on the processor, we found the bottlenecks in the XML parsing and the public key operations. In order to reduce the overhead of XML parsing, we implemented a prototype of a stream-based WS-Security processor which is event-based and avoids the DOM construction and character code conversions. In the best cases, the stream-based processor offers about 80% performance improvement in comparison to the current DOM-based processor, for both the processing time and the memory usage. In order to avoid the expensive public key operations, we also introduced WS-SecureConversation to establish a secure context between the communicating parties. This can be compared to the SSL approach, where the actual data transfer is done with very high performance using shared keys, after the secure context is established using the public key information. With WS-SecureConversation it is expected that WS-Security will be utilized with reasonable performance and contribute to the wide deployment of Web services.

REFERENCES

Apostolopoulos, G., Peris, V., & Saha, D. (1999). Transport Layer Security: How much does it really cost? *Proc. of the IEEE INFOCOM 1999*.

Canonical XML Version 1.0 (2001), Retrieved August 22, 2003, from http://www.w3.org/TR/2001/REC-xml-c14n-20010315

Document Object Model (DOM) Level 2 Core (2000). Retrieved August 22, 2003, from http://www.w3.org/TR/DOM-Level-2-Core/

Freier, A. O., Karlton, P., & Kocher, P. C. (1996). The SSL Protocol Version 3.0, Retrieved August 22, 2003, from http://wp.netscape.com/eng/ssl3/draft302.txt

He, X. (2003). *A Performance Analysis of Secure HTTP Protocol*. STAR Lab Technical Report, Department of Electrical and Computer Engineering, Tennessee Tech University.

JSR 109: Implementing Enterprise Web Services (2002). Retrieved August 22, 2003, from http://www.jcp.org/en/jsr/detail?id=109

Simple Object Access Protocol (SOAP) 1.1 (2000). Retrieved August 22, 2003, from http://www.w3.org/TR/SOAP/

Tamura, K., Makino, S., Imamura, T., & Nakamura, Y. (2003). Latency performance analysis of a Web Services Security Implementation. *Proc. of the 2003 ACM Workshop on XML Security*, to appear.

Web Services Secure Conversation (WS-SecureConversation) (2002). Retrieved August 22, 2003, from http://www.ibm.com/developerworks/library/wss-secon

Web Services Security (WS-Security) (2002). Retrieved August 22, 2003, from http://www.ibm.com/developerworks/library/ws-secure/

XML Encryption Syntax and Processing (2002). Retrieved August 22, 2003, from http://www.w3.org/TR/xmlenc-core/

XML Signature Syntax and Processing (2002). Retrieved August 22, 2003, from http://www.w3.org/TR/xmldsig-core/