Software Engineering Institute

Performance Analysis of WS-Security Mechanisms in SOAP-Based Web Services

Marc Novakouski
Soumya Simanta
Gunnar Peterson
Ed Morris
Grace Lewis

November 2010

TECHNICAL REPORT
CMU/SEI-2010-TR-023
ESC-TR-2010-023

Research, Technology, and System Solutions
http://www.sei.cmu.edu

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about SEI publications, please visit the library on the SEI website (www.sei.cmu.edu/library).

Table of Contents

List of Figures

List of Tables

Abstract

Identity management (IdM) solutions in web services environments are often compared on the levels of performance and security they provide. Selecting the appropriate IdM solution for a given system or application often requires making tradeoffs between security and performance, while also considering the system's contextual and environmental requirements and constraints. This paper presents the results of a series of experiments targeted at analyzing the performance impact of adding WS-Security, a common security standard used in IdM frameworks, to SOAP-based web services. The goal of this work is to establish a baseline of performance data that can be used to explore performance/security tradeoffs in environments with complex attributes, such as resource or bandwidth limitations.

1 Introduction

Security and performance are two primary quality attributes that shape the architecture of any software-based system. These two quality attributes are often in tension [5]. This tension is amplified in systems that use XML-based web services. While using XML as the communication medium for web services promotes quality attributes such as interoperability, flexibility, modifiability, and shorter development time [11], its use to support security significantly affects performance—primarily due to its verbosity [4]. The standard means of adding security is to increase the content of each XML message, which adds to the burden of an already heavyweight communication medium and magnifies the tension between security and performance.

This paper describes a series of experiments that focus on understanding the performance impact of different security-level mechanisms based on WS-Security.[1] The collected data represents a starting point for understanding tradeoffs between security and performance and forms a basis for making engineering and architectural decisions. Section 2 presents related work. Section 3 presents an overview of the security mechanisms that are being examined and the experimental methodology. Section 4 presents the experimental results and provides some analysis. Section 5 presents overall conclusions based on the experiment results. Finally, Section 6 presents possible future work to leverage the baseline established in this paper.

[1] WS-Security has been defined as "a standard of SOAP extensions that can be used when building secure Web services to implement message content integrity and confidentiality" [6].

2 Related Work

There has been considerable effort to benchmark various aspects of web service performance. Work by Head and associates [15] primarily focuses on comparing the performance of popular SOAP 1.1 toolkit implementations circa 2005 and thus is out of date with the most recent SOAP engines. Further research led by Head [16] expands that work to compare the performance of a wide variety of XML parsers and updates to SOAP 1.2 but is still orthogonal to our interests in security. Additional protocol comparison with no security considerations is found in "Performance of Web Services" by Jeckle and Melzer [21]. Finally, the work reported in "Comparing Web Service Performance" [22] demonstrates a performance comparison of two popular application servers running web service benchmark tests.

Early discussions of the cost of implementing security included the blog posts in "Fat protocols slow Web services" [23] and "Fat Protocols" [24]. The work in "Characterizing Secure Dynamic Web Applications Scalability" [17] takes the first step in examining the cost of security in web services, but restricts its scope to channel-based (SSL) security. "Comparison of performance of Web services, WS-Security, RMI, and RMI-SSL" [19] takes the next step by comparing two different methods of distributed communication (RMI and web services) and comparing the impact of implementing comparable security measures via WS-Security and RMI-SSL, but does not consider different levels of security.

However, there are fewer examples of research that specifically examine the performance impact of different levels of security in SOAP-based web service environments. Sosnoski's work [18] approaches our topic by discussing different approaches to securing web services, but does not offer any quantitative comparison. The work by Liu, Pallickara, and Fox [20] is notable in this context as it focuses on the comparison of performance of different encryption algorithms and does a limited comparison of different security mechanisms. It differs from the work described in this paper due to the far more extensive set of security approaches and metrics that are considered in this paper.

The work that is most closely related to the work presented in this paper is by Chen, Zic, Tang, and Levy [3]; this work compares different security mechanisms in detail and addresses payload size. The contribution of the work described in this paper is the comparison of a larger range of payload types and sizes, analyses of different payload complexities, the secure conversation security mechanism, and mechanism startup costs, as well as the examination of the cost of security from the perspective of message size and resource usage.

3 Experimental Approach

Adding security to web services has a cost that depends on the type (or types) of security mechanisms used. This cost can range from performance degradation, to increased message size, to additional consumption of resources, or some combination thereof. It is commonly accepted that more complex security mechanisms incur greater costs due to increased overhead. However, this may or may not be always true because the cost incurred by security is context- and situation-dependent. In order to make informed decisions about performance/security tradeoffs, it is necessary to measure and compare these costs.

The experiments described in this paper seek to establish a baseline of results that can be used to compare different security mechanisms in a web services context. With such a baseline of measurements, further experiments can be designed and executed to compare the tradeoffs in more complex scenarios, such as contexts in which network connectivity is intermittent or where available computing resources are constrained or limited.

3.1 Experiment Goals

The goal of the experimentation was to find the answers to three high-level questions that address the cost/benefit tradeoffs of security in the context of WS-Security and SOAP-based web services.

1. What is the cost of security with respect to roundtrip (request-response pair) time?
2. What is the cost of security with respect to message size?
3. What is the cost of security with respect to resource usage?

3.2 Experiment Subjects

This section describes the different end-to-end, message-level security mechanisms that were selected to be benchmarked by the experiments. Each security mechanism under test was evaluated from the perspective of the three experiment questions. In addition, combinations of the mechanisms (e.g., integrity and confidentiality) were benchmarked to compare their performance with that of the individual mechanisms. Combinations of mechanisms will generally provide a more secure environment overall than individual mechanisms alone and are commonly used in commercial and industrial security frameworks such as multifactor/multilayer authentication [12].

Security Mechanism #1: No Message-Level Security (No Security)
The first mechanism under test used a plain SOAP message exchange with no security overhead. This established a performance baseline.

Security Mechanism #2: Message Authentication (Password Only)
The second mechanism involved username/hashed password authentication implemented at the message level. To accomplish this, authentication tokens within the SOAP message were enabled using WS-Security.

Security Mechanism #3: Message Integrity (Sign Only)

The third mechanism involved ensuring message integrity through the use of digital signatures and hashing. To accomplish this, WS-Security in the message header was enabled by adding a digital signature to the SOAP document, using XML Signature, for both the request and the response.

Security Mechanism #4: Message Confidentiality (Encrypt Only)

The fourth mechanism involved encryption to ensure message confidentiality. To accomplish this, WS-Security was used in the message header to encrypt the SOAP document, using XML Encryption, for both the request and the response.

Security Mechanism #5: Integrity and Confidentiality (Sign Then Encrypt)

The fifth mechanism involved a combination of two different approaches, digital signatures and encryption. To accomplish this, WS-Security was used in the message header first to apply a digital signature and then to encrypt the SOAP document, for both the request and the response.

Security Mechanism #6: Integrity and Confidentiality (Encrypt Then Sign)

The sixth mechanism reverses the order of operations in Security Mechanism #5. To accomplish this, WS-Security was used in the message header first to encrypt and then to apply a digital signature to the SOAP/XML document, for both the request and the response.

Security Mechanism #7: Secure Conversation

The seventh mechanism provides a conversation-level (as opposed to a message-level) approach to security. A conversation, or sequence of messages exchanged between two participants, can have security data abstracted to a predetermined token that is established at the beginning of the conversation. The token is attached to the header of each following message and specifies the mechanisms to be used for each message in the conversation. This approach is in contrast to mechanisms 2 through 6, in which all necessary security mechanism information is included in the header of each message.

The implementation of the Secure Conversation mechanism for this experiment is slightly different from the WS-SecureConversation standard. This implementation uses the sample implementation provided by Apache Rampart that leverages a SymmetricBinding construct for bootstrapping—with a timestamp, a signature, and encryption as the security mechanisms [10, 1].

3.3 Experiment Architecture

The test suite consisted of several experiments that focused on the performance aspects of interest: roundtrip time, message size, and resource usage.

The experiments were run on a pair of servers running Apache Tomcat with Apache Axis2 as the SOAP engine and Apache Rampart as the security module [14, 2, 1]. The experiments leveraged sample implementations of the different security mechanisms provided by Apache Rampart. These examples consisted of a simple echo pattern in which the client passes a “Hello World” string to the service and the service echoes the string back [1]. Most of the experiments modify the basic service in minor ways to test different aspects of the communication. These modifications are explained in detail in the following sections. To gather performance measurements, the

examples were modified using the Perf4J and Apache Log4J framework and inserting instances of the Perf4J StopWatch class [7, 8].

All of the test cases are based on request-response message exchanges using HTTP as the communications channel. Because an actual implementation is likely to use HTTPS/SSL, this aspect is considered a constant and therefore was removed from these test cases. Figure 1 provides a high-level view of the basic architecture of the testing framework.

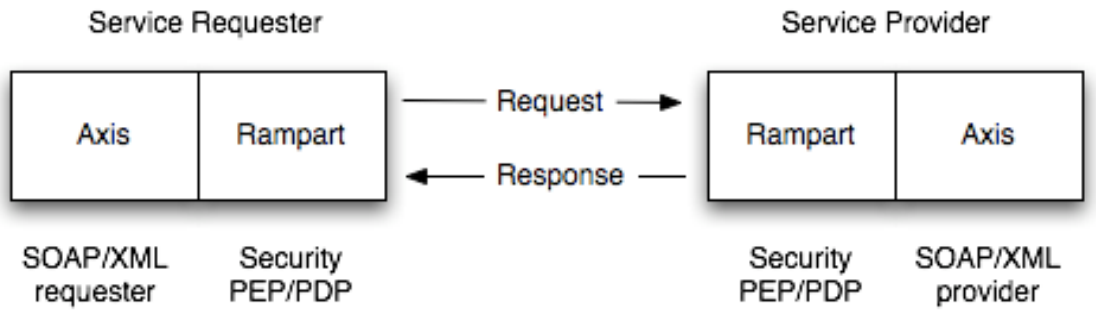

Figure 1: Basic Implementation Architecture of the Experiments

For further technical details on the experiments, please consult Appendices A-E.

3.4 Experiment Configurations

The testing efforts looked at several ways to benchmark the impact of security on roundtrip response time, message size, and resource usage. To analyze this impact, each experiment exercised one of two web services using one of two test configurations.

3.4.1 Web Services

"Echo" Web Service (Simple Payload Structure): This service takes an input string and returns the same string as the response.

"Employee Details" Web Service (Complex Payload Structure): This service takes no input and returns a one-element array of employee objects. An employee object contains an employee's details, such as name, e-mail address, phone number(s), and social security number.

3.4.2 Configurations

"Static Payload" Configuration: In this configuration, a client application executes the target web service 10,000 consecutive times with the default input. (The Employee Details service has no default input.) Both services were exercised in this configuration.

"Dynamic Payload" Configuration: In this configuration, a client application executes the target web service 10,000 consecutive times with an input payload that is increased in each execution. This configuration is only used with the Echo service, which has a default initial payload of a simple "Hello World" string.[2] The payload is increased with each execution by appending an ad-

[2] The experiments with the Echo service used a "Hello World " string, a 12-character string with a space character at the end. There was no particular reason to use this string instead of the 11-character "Hello World" string. It was an arbitrary choice.

ditional “Hello World” string to the payload used in the previous execution. This results in a range of payload sizes from 12 bytes up to 100+ kilobytes over the 10,000 executions.

3.5 Experiment Measures

Consistent with the experiment goals, each experiment analyzed three different benchmarking measures:

1. roundtrip response time
2. message size
3. resource usage

While we can measure roundtrip response time and message size directly, our approach to measuring resource usage had to be less obvious. One approach we could have taken to analyze the resource usage of the different security mechanisms would be to record CPU usage and memory usage over the course of the execution of a test. However, CPU and memory usage are extremely dependent upon the resources available on a given platform. We chose instead to simply record those measures and observe the impact upon them (if any) of different Java Virtual Machine (JVM) environments.

Our results (which will be discussed in detail later) showed that in resource-constrained environments, memory management tasks such as garbage collection can have significant performance impact upon the execution of even simple web services. Therefore, our approach was to identify the inflection point at which the memory management tasks stopped impacting the performance of the experiments. While exact by no measure, this approach can give a general idea of the resources necessary to effectively leverage the security mechanisms under test.

4 Results and Analysis

This section describes experiments we performed to test the performance implications of the security mechanisms discussed in Section 3.2 and summarizes our results. The purpose of this analysis is to understand the impact of using different security mechanisms on system performance. As discussed in Section 3, the experiments also establish a baseline of metrics for future work on tradeoffs in complex environments.

4.1 The Cost of Security: Roundtrip Response Time

Our first focus was on the performance of each security mechanism for a roundtrip response time. The initial experiments confirmed our hypothesis that adding security to web services considerably increases roundtrip response time. Roundtrip response time of SOAP-based web services with different security mechanisms can be as much as 20 times higher than that of an unsecured web service, as shown in Table 1. This table shows the average, median, and standard deviation roundtrip time in milliseconds of each mechanism in a test using the basic Echo service (simple payload structure) in the static payload configuration.

Table 1: Statistical Summary per Mechanism, Echo Service/Static Payload (ms)

	No Security	Password Only	Sign Only	Encrypt Only	Sign -> Encrypt	Encrypt -> Sign	Secure Conversation
Average	3	8	38	36	65	65	39
Median	3	7	37	36	64	65	38
Std. Dev.	0.696	1.862	2.814	2.218	3.332	3.117	3.682

Building on these results, we looked at the performance of each security mechanism by running an experiment with the Echo service using the dynamic payload configuration. Figure 2 shows the performance measures of each mechanism in this experiment.

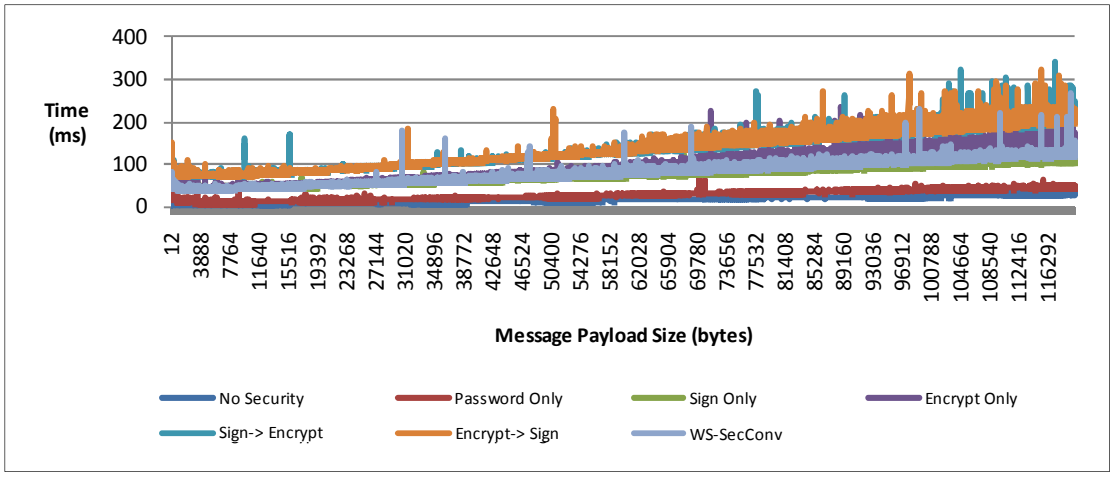

Figure 2: Roundtrip Time per Mechanism, Echo Service/Dynamic Payload

Finally, we re-ran the experiment with the Employee Details service (complex payload structure) in the static payload configuration to examine how the different security mechanisms handle the difference in payload type and size. These results are shown in Table 2.

Table 2: Statistical Summary per Mechanism, Employee Details/Static Payload (ms)

	No Security	Password Only	Sign Only	Encrypt Only	Sign -> Encrypt	Encrypt -> Sign	Secure Conversation
Average	100	106	174	157	197	190	228
Median	100	105	173	156	196	190	237
Std. Dev.	5.298	6.115	8.376	7.847	9.673	7.494	23.483

The data collected from these experiments can be analyzed in a number of ways. The only overall conclusion is that there is no clear "winner"; however, the analysis presented in the following sections should help in making design decisions.

4.1.1 Simple vs. Complex Payload Structure

From the results in Table 1, and Table 2, and Figure 2, it is clear that message payload structure (whether a simple string or a complex object converted to an XML-based structure) can significantly impact the performance of each security mechanism. Even though it would be logical to assume that the more the complex a mechanism the larger the impact on roundtrip response time, results from the three experiments do not support this assumption. We analyzed Sign Only versus Encrypt Only, the two Sign/Encrypt mechanisms, and Secure Conversation.

It is useful to compare the performance of Sign Only to Encrypt Only for the Echo service case to see where they diverge, because Figure 2 suggests that they are roughly equal for some small message payload sizes. Figure 3 shows this comparison.

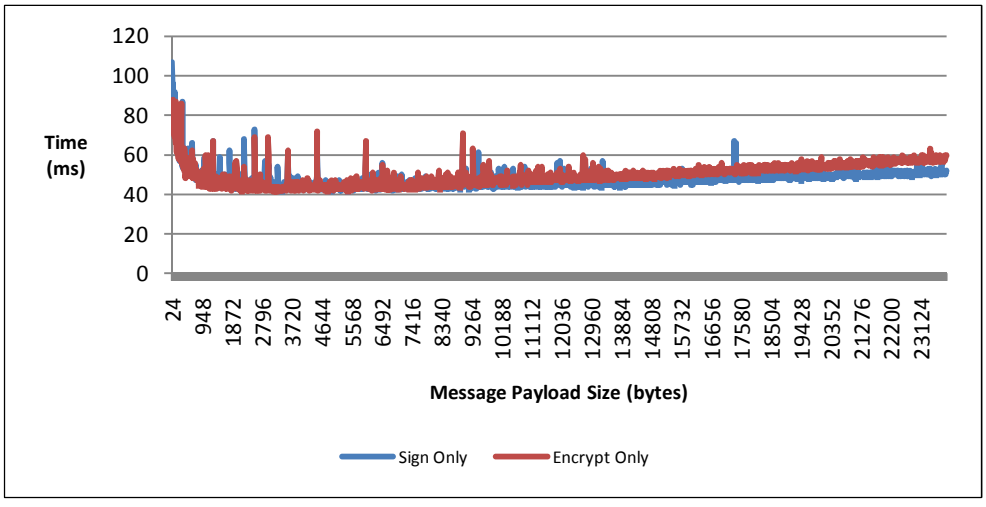

Figure 3: Sign Only versus Encrypt Only for Message Payloads up to 20kb, from Figure 2 Data

Analysis of this data suggests that for messages up to approximately 7kb in payload size with simple payload structure, Encrypt Only performs as well as Sign Only. However, past the 7kb payload size point Sign Only provides superior performance.

Values in Table 1 and Table 2 indicate that Sign Only performs better than Encrypt Only for messages with simple payload structure, but Encrypt Only outperforms Sign Only for messages with complex payload structure. Sign Only roundtrip time increases more slowly as payload size increases for the simple payload structure case (for more detail, see Section 4.1.3).

The data in those tables also suggests that both Sign/Encrypt mechanisms have virtually identical performance in the simple payload structure case, but Encrypt then Sign performs better in the complex payload structure case. This mechanism has a lower average roundtrip time and greater reliability (shown as a smaller standard deviation). While no data collected provides conclusive explanation for this difference, one possibility is that Encrypt then Sign causes less work because it does not require encryption/decryption of the digital signature prior to verification.

Finally, in the simple payload case Secure Conversation shows the best performance of all of the mechanisms that use encryption (see Table 1). However, when used with complex structure payloads, it shows the worst performance of all mechanisms (see Table 2). In addition, for complex payload structures, Secure Conversation appears to have somewhat unreliable performance, exhibited by the high standard deviation numbers noted in Table 2. The reason for the low reliability is unknown; one possibility is that the overhead for maintaining the conversation token may be responsible.

Conclusions

- In all cases, the best performing mechanism (aside from no security at all) was the Password Only mechanism. Because this mechanism provides relatively weak security, its use is usually not recommended. However, in situations where the need for message-level security is low (e.g., secured networks, SSL-based communications) Password Only may be a viable security mechanism that provides maximum performance.
- There are several tradeoffs between Sign Only and Encrypt Only approaches to message-level security. In contexts where either will provide sufficient security and performance, the choice may be made based on the complexity of the message payload structure. For simple payload structures, a Sign Only approach provides the best performance, while for complex payload structures the Encrypt Only security mechanism performs the best. Also, it should be noted that for messages with simple payload structure and payload size under 7kb or so, both mechanisms will result in approximately the same performance profile.
- In the case of messages with simple payload structure, both Sign/Encrypt mechanisms have the same performance. For messages with complex payload structures, Encrypt then Sign outperforms the Sign then Encrypt approach. Also, if a token-based session mechanism such as Secure Conversation is unnecessary or undesired, the Encrypt then Sign mechanism is probably the better approach.
- When used with messages with simple payload structure, Secure Conversation is the top performer of all the encryption-enabled mechanisms we tested. However, Secure Conversation does not perform well with messages with complex payload structure and thus seems to be appropriate only when performance is not the driving quality attribute.

4.1.2 Startup Times

Startup time is the amount of time that a roundtrip communication requires on the first execution of an experiment. This time can be relevant when lost connections need to be reestablished. Our test results confirmed the intuitive expectation that more complex security mechanisms require longer startup times (see Figure 4). One interesting phenomenon is that there are two classes of startup times, **initial** communications and all **subsequent** communications. Initial communications represent the first roundtrip communication that a service participates in after deployment or redeployment. Results show that the startup time associated with initial communications tends to be roughly twice as long on average than the startup time for all subsequent communications with that service.

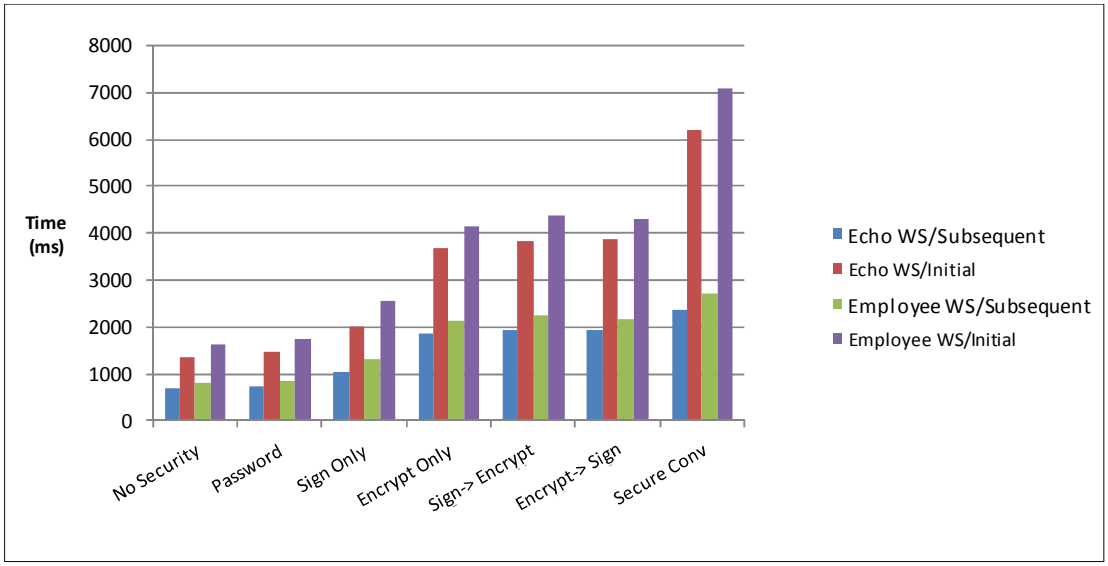

Figure 4: Startup Times per Mechanism/Service, Initial versus Subsequent Communications

The data in Figure 4 for Sign Only startup time matches to some extent data for its non-startup roundtrip time relative to performance for each mechanism in Table 1, Table 2, and Figure 2**.** The differences observed between startup and non-startup times are that

- Startup time for Sign Only shows significantly better performance than any mechanism with encryption, with Echo or Employee Details service (see Figure 4).
- Non-startup time for Sign Only, Encrypt Only, and Secure Conversation is similar with Echo service (see Table 1).
- Non-startup time for Sign Only has worse performance than Encrypt Only, though better than Secure Conversation by a significant margin, with Employee Details service (see Table 2).

Encrypt Only startup time is similar to both Sign/Encrypt options, a result that does not correlate to its non-startup performance. This result is unexpected and cannot be explained by our data or analysis. Perhaps the time required to initially access the shared keys is significant.

Startup time performance of Secure Conversation correlates best with the non-startup roundtrip time performance results presented in Table 2, where it was the worst performer. Startup time for

Secure Conversation is especially poor for initial communications: a 59% increase over the Encrypt then Sign (versus a 22% increase over Encrypt then Sign in subsequent communications).

One other notable result is the increase in roundtrip time when the Employee Details web service is used instead of the Echo service. One reason for the increase might be payload size.[3] The base payload of the Echo web service is 12 bytes, while that of Employee Details is approximately 112 bytes [26, 27]. Considering that the call to the Employee Details service has no arguments while the Echo service has the same input as output, the roundtrip payload size with the Employee Details service is approximately 450% larger than that with the Echo (112 versus 24 bytes).

To examine how greater payload size increases roundtrip time, we repeated the experiments using the Echo service with static payloads of 60 bytes (5 concatenated "Hello World" strings) and of 120 bytes (10 concatenated "Hello World" strings). In the 60-byte test, total message payload over the roundtrip sequence was 120 bytes, close to the Employee Details payload, testing the impact of sending the same total payload. The 120-byte test evaluated the impact of the same payload size being transferred on the initial communication. These tests allow us to examine if roundtrip or initial payload size was the reason for the roundtrip time increase observed when using the Employee Details web service.

As shown in Figure 5, the effect of payload size on roundtrip time appears to be negligible. Some small increases in time resulted as payload size increased with the Echo web service, but some small decreases were seen as well. For each security mechanism, in addition, there is a relatively constant increase in roundtrip time when the Employee Details service case was used, compared to the Echo service. As such, the data seems to suggest that payload complexity drives performance difference.

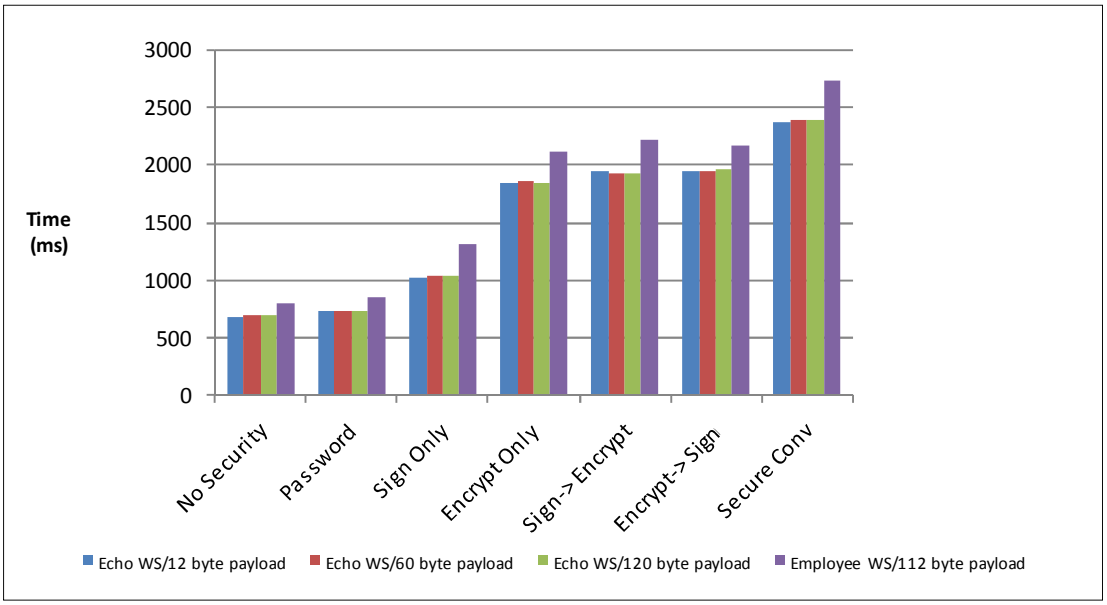

Figure 5: Comparing Startup Times for Different Payload Sizes for Each Mechanism

[3] Estimates of the payload sizes are based on methods for measuring the size of Java objects that are described in the references cited.

The final observations that can be made based on the startup data have to do with total cost of communication. While Table 2 shows that Encrypt Only outperforms Sign Only in non-startup communications with complex payloads (157 versus 174ms), Figure 4 shows that startup times for Encrypt Only exceed those of Sign Only. Figure 6 plots the total time of the two mechanisms, showing that for roughly the first 46 messages Sign Only performs better than Encrypt Only.

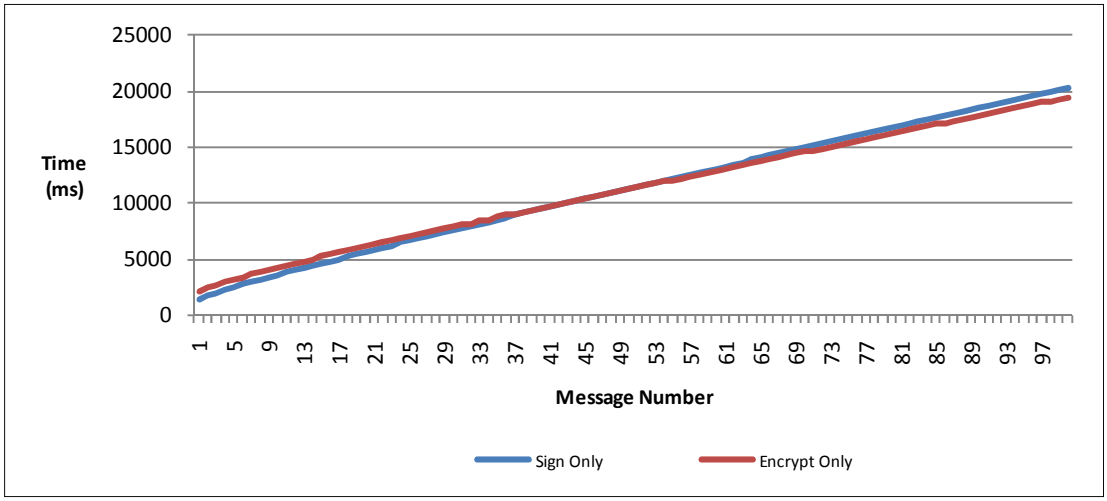

Figure 6: Total Cost of Sign Only versus Encrypt Only, First 100 Messages

Also, for simple payloads, the total cost of both Sign/Encrypt mechanisms can be lower than that of the Secure Conversation mechanism despite the lower non-startup costs of Secure Conversation for some finite set of messages. Figure 7 shows this inflection point to be roughly after the 12th message.

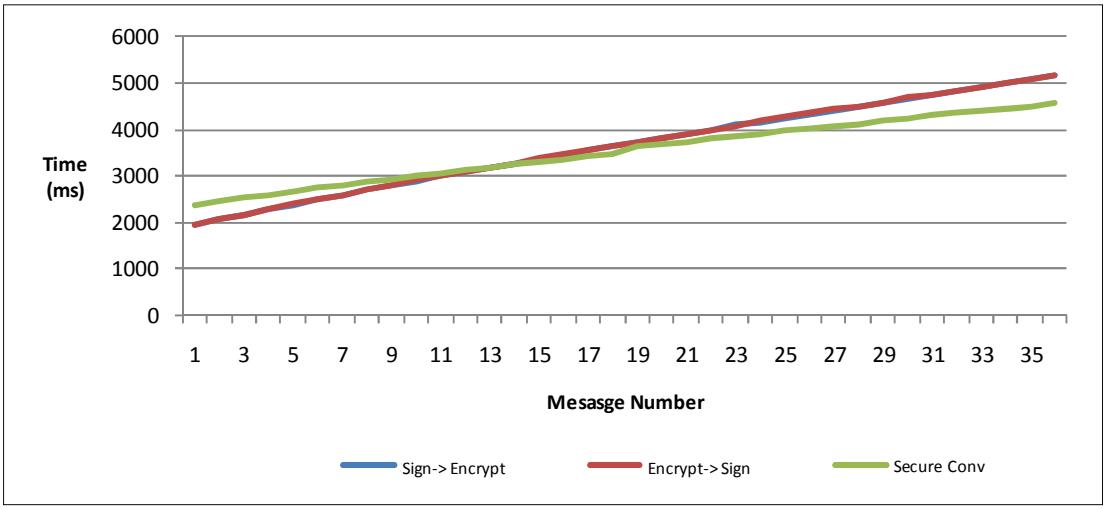

Figure 7: Total Cost of Both Sign/Encrypt Mechanisms versus Secure Conversation for Simple Structure Payloads

Conclusions

- Initial communications (the first that a service has after deployment or redeployment) will have roughly twice (or more) the subsequent communications cost.
- Any security mechanism that uses encryption will experience significantly longer startup times than any non-encryption mechanism.
- Each security mechanism has a significant impact on roundtrip response time of the initial communications a client application has with a given service. The more complex the security mechanism, the higher the cost incurred. (Recommendation: If possible, newly deployed web services should be immediately executed by some application, such as a test or dummy application, to eliminate the post-deployment initial communications startup cost.)
- Communications with services that handle complex structure payloads (such as XML data types) will have a higher startup cost than communications that deal with simple structure payloads.
- For short message sequences (< 50 message exchanges), the Sign Only mechanism performs better than the Encrypt Only mechanism, due to lower startup time. For extremely short message sequences (< 10 message exchanges) either Sign/Encrypt mechanism performs better than the Secure Conversation mechanism, due to lower startup time.

4.1.3 Rate of Cost Increase as Payload Size Increases

The data represented by Table 3 suggests that increasing payload size causes some security mechanisms to experience higher roundtrip time. If true, this finding provides insight on which mechanisms are better suited for larger or smaller payload sizes. Table 3 presents an analysis of the rates for each mechanism at minimum payload size and 117kb payload size.

Table 3: Approximate Rates of Increase of Each Security Mechanism, from Figure 2 Data

	No Security	Password Only	Sign Only	Encrypt Only	Sign -> Encrypt	Encrypt -> Sign	Secure Conversation
Minimum (ms)	3.00	10.00	42.00	42.00	71.00	71.00	41.00
Average @ 117kb (ms)	29.55	47.45	108.18	163.64	216.18	212.45	130.00
Increase (%)	884.85	374.55	157.58	289.61	204.48	199.23	217.07

Due to the relatively high standard deviations documented in Table 1 and Table 2, it is difficult to calculate a rate of increase for each mechanism. We took the minimum value found during the dynamic payload Echo service experiment (results documented in Figure 2) and compared it to the average of the last 10 results from that data set (where payload is roughly 120,000 bytes or ~117k). We used the average time to mitigate spikes due to memory management issues. This method is inexact for determining the rate of increase, but it provides a general approximation that confirms what was observed in Figure 2. Of the mechanisms with encryption, Sign Only performs best, Encrypt Only worst, with the Sign/Encrypt mechanisms and Secure Conversation rating between the two.

This analysis confirms the hypothesis that the most complex algorithms have the highest rate of increase in roundtrip time cost as payload increases. The most interesting result is Secure Conver-

sation, which performs well compared to the encryption-based mechanisms and confirms the early analyses from examining Figure 2.

Conclusions

- For message payloads with simple structure, the Sign Only mechanism is the best non-trivial security mechanism for passing data of increasing sizes.
- For message payloads with simple structure, Secure Conversation can be a good choice for enabling security while ensuring good performance as payload size increases, compared to other mechanisms.

4.1.4 Convergence of Security Mechanisms

One property that may be important to examine is the speed at which a given security mechanism converges to its *normal* values. As shown in Figure 8, each mechanism needs a certain number of communication exchanges before it stabilizes to some predictable level of performance.

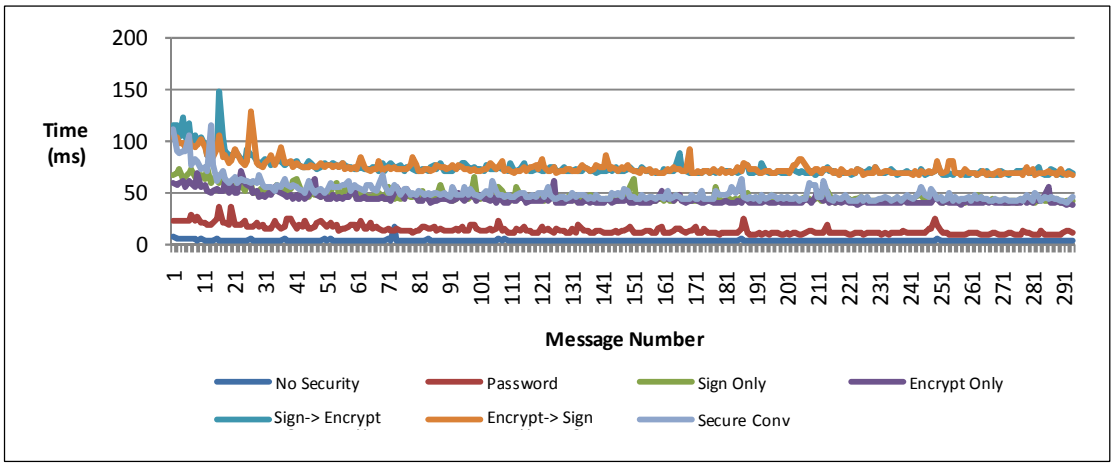

Figure 8: Startup Behavior of each Security Mechanism with Echo Service in Static Payload Configuration

This *stabilization* behavior can be defined by calculating the mathematical convergence of the sequence. A sequence of real numbers is said to converge if there is some number X such that past some point in the sequence, all following numbers in the sequence are within some range of X [25]. For a given element Z of the sequence, if it is the Y^{th} or later number in the sequence, $|X - Z| <$ some value ε.

Unfortunately, inherent noise in the data prevented the calculation of a useful value of Y for any value of ε smaller than 30ms, a significantly large range for data sets that show standard deviations no larger than 4ms (as shown in Table 1). We removed noise from the equation by looking not at ranges, but at how far into the data set we have to go for each mechanism to produce a value that is less than or equal to the resulting average. While not nearly as precise a measure of convergence, this approach eliminates the effect of noise and provides a general sense of the speed of convergence. Table 4 shows the results of this analysis on the experiment that used the Echo service in static payload configuration.

Table 4: Convergence Points for Each Mechanism for Echo Service in Static Payload Configuration

	No Security	Password Only	Sign Only	Encrypt Only	Sign -> Encrypt	Encrypt -> Sign	Secure Conversation
Convergence Point	22	505	883	792	570	588	769

To contrast this, the same numbers were calculated for the Employee Details service. The results are presented in Table 5.

Table 5: Convergence Points for Each Mechanism for Employee Details Service in Static Payload Configuration

	No Security	Password Only	Sign Only	Encrypt Only	Sign -> Encrypt	Encrypt -> Sign	Secure Conversation
Convergence Point	21	138	143	98	126	148	17

On the one hand, the numbers between Table 4 and Table 5 are similar, in that No Security takes a short time to reach its ultimate average while other mechanisms take much longer. This comparison confirms the hypothesis that adding security mechanisms is costly to roundtrip time from a convergence standpoint. On the other hand, the stabilization occurs much faster in the complex payload structure case overall (with the Employee Details service), which is not easily explained and is outside of the scope of this work.

The outlier data point to note is the performance of Secure Conversation in the complex payload structure case. It outperforms even the No Security option. While extremely unexpected, this result can be explained by looking at the variance in this mechanism's performance. Figure 9 is a graph of its performance compared to the other mechanisms for the complex payload structure configuration. The significant variance in the performance of Secure Conversation affects the average roundtrip response, allowing the convergence test to be met sooner than expected.

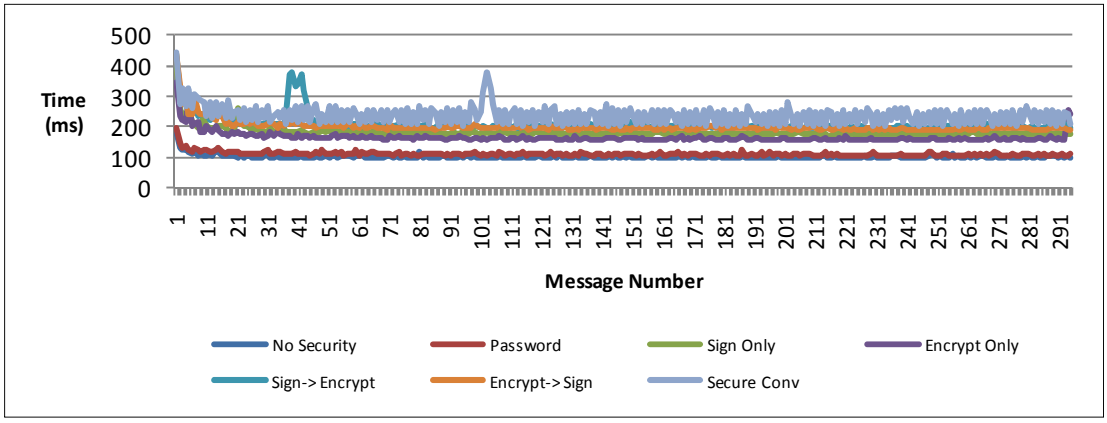

Figure 9: Performance of Each Mechanism, Employee Details Service in Static Payload Configuration

Conclusions

- The only appreciable difference in convergence rate between the security mechanisms is with the No Security option. There may be some variance between the different mechanisms.

However the startup costs associated with each mechanism (Section 4.1.2) overwhelm the effects of any observable differences in convergence rate.

- There is strong evidence of significant noise or some other effect that prevents Secure Conversation from converging at a predictable rate relative to the other mechanisms.
- Overall, it is difficult to take the results on rate of convergence as a strong recommendation about a security mechanism, due to an inability to measure the values precisely; therefore this generalized data should be used sparingly. If used at all, convergence rate should be considered only in simple payload structure cases. In complex payload structure contexts, this data suggests that it has even less influence.

4.2 The Cost of Security: Message Size

The second focus of the experimentation was message size. Our hypothesis was that adding security to messages will increase their size. Further, if adding security does increase message size, we wanted to determine how much of an increase to predict for each mechanism.

We extracted message sizes associated with each security mechanism from roundtrip response times. We assessed the impact of security as message payload size increases using the Echo service in a dynamic payload configuration (with payloads increasing to 100kb). As Figure 10 shows, security mechanisms that do not use encryption, such as Sign Only, add a constant amount of size (overhead) to the message, but encryption-enabled mechanisms add message overhead at a linear rate with payload size increase.

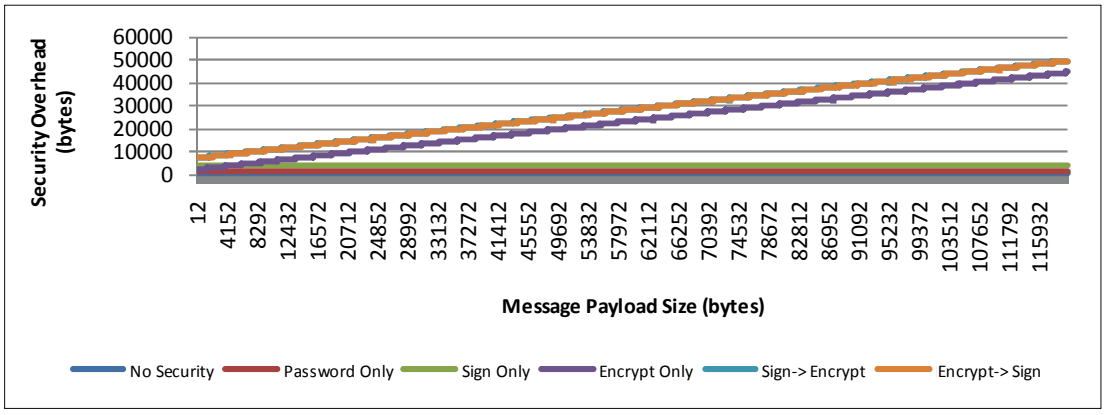

Figure 10: Message Overhead per Mechanism as Payload Size Increases

In addition to identifying a trend across increasing payload sizes, it is instructive to compare the base overhead sizes of each mechanism. This information is presented in Table 6 for the simplest case.

Table 6: Message Size for Echo Service with Static Payload Configuration (Bytes)[4]

	No Security	Password Only	Sign Only	Encrypt Only	Sign -> Encrypt	Encrypt -> Sign
Raw Payload Size	11	11	11	11	11	11
Request Size	522	1396	3956	2480	7511	7385
Response Size	479	479	4627	2677	8181	8053

In Table 6, the raw payload size data in is the message, and the request size is the message placed in a SOAP envelope. Putting the message in the SOAP envelope has a significant size cost that increases as security mechanisms are added. Further, the increased request size correlates with the response time for a mechanism, showing that cost increases with complexity. One additional result worth noting from this experiment is that Encrypt Only outperforms Sign Only, where the message with request size is 4kb or less.

Conclusions

- As with roundtrip response time, the Password Only security mechanism adds the least message overhead. Also, as with roundtrip response time, this mechanism is not recommended except in environments that have low message-level security requirements.
- Where it is important to limit message overhead while using a non-trivial security mechanism, Encrypt Only offers the best performance up to payload sizes of roughly 4k. Past this size, Sign Only should be used to avoid the steadily increasing costs of using encryption.

4.3 The Cost of Security: System Resources

The third focus of the experiments was the cost of security in terms of system resources. Our hypothesis was that security mechanisms would require substantial system resources to perform effectively. The goal was to examine the resource requirements of the security mechanisms in order to provide guidance on how to select computing resources for a given security framework.

One observation made early during the experimentation is that security carries a significant resource cost. Even using simple services, we found performance degradation due to memory management issues for all but the most basic message payloads. We addressed the problems for the most part by tuning the JVM with a more effective garbage collection algorithm and with increased memory allocation [9].

The results show that Secure Conversation is more sensitive to the amount of available resources than the other mechanisms. Figure 11 compares the impact of increasing the allocated memory to the test application for this mechanism. Realizing the maximum potential roundtrip time performance of Secure Conversation may depend upon the resources available to the infrastructure.

[4] This data was not available for the Secure Conversation mechanism. Future work will gather this data.

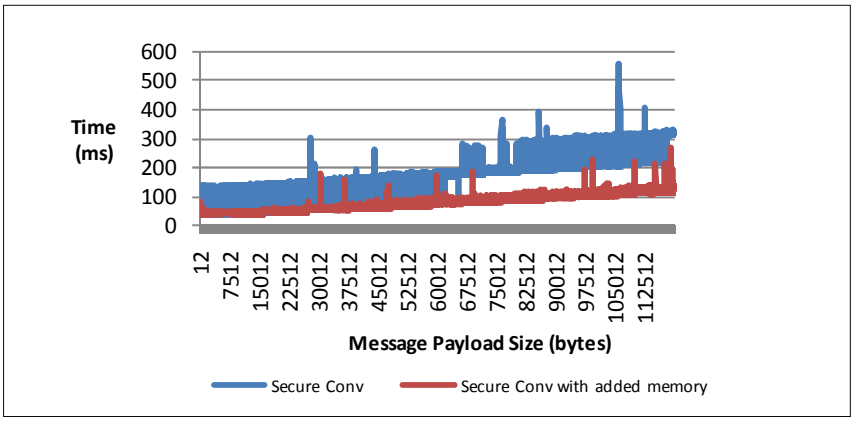

Figure 11: Secure ConversationPerformance, Default JVM Memory vs. Manually Set 25 MB–50MB Range

Conclusions

- Security mechanisms such as Sign Only and Encrypt Only (e.g., XML Signature and XML Encryption) require significant resources to perform at peak levels. Services and applications intended to implement these mechanisms as security services will likely require more than the default memory allocated to them by a standard Sun Hotspot JVM [13]. In order to meet these resource needs, developers should identify expected message payload sizes and structure complexities early in the design process and tune the services to perform optimally on the available hardware.
- Secure Conversation and similar mechanisms will likely require more resources than non-session-based mechanisms in order to perform at peak levels. Implementations of these mechanisms should be tuned aggressively to ensure that the necessary resources are available to ensure the expected performance.

5 Summary

There is no simple answer to the critical question of whether the cost of web service security is acceptable or too high. The exact cost of security on the runtime performance of a SOAP-based web service will depend on many factors. A good understanding of the quality expectations in a specific environment, as well as of the cost of security relative to the costs of service execution, network latency, and overhead can be determining factors. Variables such as JVM heap memory, garbage collection algorithm, and network bandwidth contribute to roundtrip response time. Our work shows that there is a sufficient range of alternative security mechanisms to perform an effective tradeoff analysis, once the driving use cases in a specific context have been analyzed.

Selecting the appropriate mechanisms for web service security involves weighing the security benefits of individual mechanisms against their costs. Whether measured in terms of roundtrip response time, message size, or resource usage, security mechanisms have a considerable impact on the performance of SOAP-based web services. Our experiments reveal that adding security to web services considerably increases roundtrip response time and message size and that more complicated security mechanisms require greater system resources. The experiments also establish a baseline from which to explore different security framework contexts and establish general guidelines for selecting appropriate mechanisms.

6 Future Work

There are many opportunities for further investigation. These include high-level questions raised by the initial experimental results, as well as the effect of additional configuration parameters on the overall performance of secure web services. In this section, we briefly describe the following possible future work in those areas.

6.1 Encryption Overhead vs. Message Overhead

Overhead associated with an encryption increases with payload size. In contrast, message overhead stays constant when a message has been authenticated or digitally signed. Further investigation could reveal why this difference occurs. It would also be useful to find whether there is an optimal point at which the overhead associated with encrypting a large message outweighs the message overhead associated with splitting the message into smaller chunks and sending each chunk individually.

6.2 Potential Alternative Experiment Configuration Elements

Possible configuration elements that could be analyzed in further experiments include algorithm variation (for encryption and digital signing), simulation of constrained environments, alternative security mechanisms, and alternative security framework infrastructures.

6.2.1 Encryption and Signing Algorithms

The existing experiment setup can be leveraged as a baseline and expanded to include measurement of the performance impact of combinations of the following encryption and signing algorithms:

- Suite B algorithms
- AES-128
- SHA-1
- 1024 bit RSA
- For SSL ciphers - 3DES

6.2.2 Message Size

Security testing can be expanded to include measurement of the performance impact of larger files (1MB+) in order to understand the impact of moving large files such as media (pictures, video, etc.) or programs.

6.2.3 Resource Limitations

Security testing can be expanded to include measurement of the performance impact of the various security implementation alternatives in a resource-constrained environment.

6.2.3.1 Network Bandwidth

One possible experiment is the simulation of an unreliable network environment to measure the performance impact of various security implementation alternatives under conditions such as low-bandwidth connection, unexpected connection loss, and unexpected packet loss. Possible software that could be used to support these tests includes NistNet [26] and WanEm [27].

6.2.3.2 CPU Speed / Available Memory

Another possible experiment is the simulation of a reduced-hardware environment from the standpoint of CPU and memory.

6.2.3.3 JVM Settings

Tuning the JVM environment includes increasing heap size and modifying the garbage collection algorithm to reduce the impact of the garbage collector as message size increases. Refinements like these would be helpful to in determining minimum recommended settings for different mechanism and payload combinations. Additional experimentation can also be performed to see if any other JVM settings impact performance.

6.2.4 Additional Security Mechanism Alternatives

6.2.4.1 Partial Encryption

This alternative includes encrypting part of the message, such as the XML content, but not all of it.

6.2.4.2 Secure Session Token

This alternative implements the WS-SecureConveration standard and adds a Security Context Token (SCT) to the message.

6.2.4.3 Double Encryption

This alternative performs encryption twice on a message to provide additional security.

6.2.5 REST-Style Web Services

This alternative examines how REST may be used to improve performance in certain cases [22].

6.2.6 Token by Reference

For constrained devices, token-by-reference (as in SAML) may offer an improvement over passing tokens by value.

6.2.7 Alternative Security Framework Infrastructures

The experiments in this report use the Axis2 SOAP engine with the Rampart module for providing WS-* security implementations [2, 1]. Alternative SOAP engines such as the IBM WebSphere SOAP engine can be tested to see if there is possible impact by the engine on different security implementations [28].

Appendix A Hardware Configurations

Table 7: Hardware Configurations

Property	Service Provider Hardware Value	Service Consumer Hardware Value
Host Name	pcbls.sei.cmu.edu	pcblt.sei.cmu.edu
Operating System	Ubuntu Linux 2.6.28-15-generic SMP	
CPU	2 X Intel Pentium 3.6GHz	2 X Intel Pentium 3.8GHz
Memory (RAM)	2GB	
Hard Disk	142GB	

Appendix B Development Environment

Table 8: Development Environments

	Sun Java SDK 1.6	Apache Ant	Apache Maven
Responsibility	Java is used as the key programming language to code all the experiments.	Apache Ant is a tool for building Java programs. It is used to run the Rampart samples.	Apache Maven is a build tool for a Java project. It is required to build Rampart from sources.
Version	Sun Java JDK 1.6 Binary distribution	Apache Ant version 1.7.1	Maven version: 2.0.8
Source Modifications	None	None	None
Configuration Modification	None	None	None

Appendix C Runtime Environment

Table 9: Runtime Environment

	Sun Java SDK 1.6	Apache Tomcat 6.0.20	Apache Axis2	Apache Rampart	Log4J
Responsibility	The Java virtual machine is the runtime platform for execution of all the experiments.	Apache Tomcat is used as the application server that hosts the SOAP engine.	Apache Axis2 is the SOAP engine used by both the clients and the services.	Apache Rampart 1.4/1.5 implements the WS-security specification and can be deployed as a module to Axis2.	Apache Log4J is used as the logging framework for services as well as the clients.
Version	Binary distribution Java(TM) SE Runtime Environment (build 1.6.0_16-b01) Java HotSpot(TM) Client VM (build 14.2-b01, mixed mode, sharing)	Apache Tomcat 6.0.20 Binary distribution	Apache Axis2 1.4.1 (Binary distribution) and 1.5.1 for different tests	Rampart 1.4.1 & 1.5 (binary and source distribution)	Log4J-1.2.15 (binary distribution)
Source Modifications	None	None	None	Rampart 1.4/1.5 sources were modified in order to measure the time taken by various components of Rampart. Rampart (version 1.4) was built from sources using the Maven build tool. Only one Rampart module (rampart-core-1.4.jar) was built from sources; 1.5 was directly used from the distribution.	None

	Sun Java SDK 1.6	Apache Tomcat 6.0.20	Apache Axis2	Apache Rampart	Log4J
Configuration Modification	None	`$TOMCAT_HOME/bin/catalina.sh` was modified to enable a new garbage collection algorithm. `JAVA_OPTS="$JAVA_OPTS -XX:+UseConcMarkSweepGC"`	`$AXIS2_HOME/WEB-INF/classes/log4.properites` was modified to enable logging at various levels. The `server_axis2.log` file was generated inside `$TOMCAT_HOME/logs/server_axis2.log`. • Rampart libraries and modules were deployed. • Rampart libs were deployed to `$AXIS2_HOME/WEB-INF/lib`. • Rampart and Rahas modules were deployed to `$AXIS2_HOME/WEB-INF/modules`. Services for the samples (.aar) were deployed to `$AXIS2_HOMEWEB-INF/services`.	Rampart logging was enabled and disabled using `$AXIS2_HOME/WEB-INF/classes/log4j.properites`. `log4j.category.org.apache.rampart=FATAL`	Service side: `$AXIS2_HOME/WEB-INF/classes/log4.properites` Client side: `log4j.properties` should be the `CLASSPATH`.

To take performance measurements, the samples were modified using the Perf4J and Apache Log4J framework, inserting instances of the StopWatch class.

Appendix D Mapping of Apache Rampart Samples

The experiments exercised all seven security mechanisms discussed in Section 3.2. The implementation of each security mechanism leveraged modified versions of the sample implementations provided with the default Apache Rampart installations.

The following table provides the correlation of the Apache Rampart samples to the security implementation that was leveraged. The default implementation of the Apache Rampart samples implements a message transfer using an Echo service with a basic "Hello World" string payload. For some of the experiments, the default implementation was modified to use an Employee Details service that transfers an array of Employee objects as the payload (see Appendix E).

Table 10: Mapping of Test Cases to Rampart Samples Used in the Experiments

Test Case	Security Mechanism Alternative	Rampart Sample Number[5]
1	No Security	Sample.01 (Basic*)
2	Password Only—Message Authentication with Username and Password	Sample.02 (Basic)
3	Sign Only—Message Integrity (Digital Signature)	Sample.04 (Basic)
4	Encrypt Only—Message Confidentiality (Encryption)	Sample.05 (Basic)
5	Sign then Encrypt—Multi-Layered Message Security (Digital Signature -> Encryption)	Sample.06 (Basic)
6	Encrypt then Sign—Multi-Layered Message Security (Encryption -> Digital Signature)	Sample.07 (Basic)
7	Secure Conversation (X509 token indicating Timestamp, Digital Signature, and Encryption)	Sample.04 (Policy)

[5] Apache Rampart comes with two types of samples: Basic and Policy. Basic samples leverage different security mechanism atomically while Policy samples use WS-Security Policy Language to configure a communication session.

Appendix E Employee Class

For reference, the following is an excerpt from the Java Employee Class used by the Employee Details service that was discussed in Appendix D. The definition of the get and set methods is not included because they are the expected typical default implementations. While a real implementation would use data types more appropriate for each data attribute, this implementation uses *string* as the type for each attribute in order to make the object as comparable as possible to the simple "Hello World" string payload used with the Echo service.

When an object of this class is created, instead of using dummy data, each attribute was simply initialized to "Hello World," again to ensure that the object is as comparable as possible to the payload used with the Echo service.

```
Import java.util.Date;

Public class Employee {

  private String name;

  private String empNumber;

  private String email;

  private String position;

  private String dateOfBirth;

  private String SSN;

  private String homeAddress;

  private String phoneNumber;

  /* Typical Get and Set methods follow*/

  …

}
```

Glossary of Technical Terms

Axis2	Apache Axis2, a SOAP engine (i.e., a software module that manages the communication of web service messages in SOAP format)
Rampart	Apache Rampart, a software module that works on a SOAP engine (specifically Apache Axis2) to automatically manage WS-Security standards on the services running on the SOAP engine.
WS-SecureConversation	An OASIS web service standard that creates a session token for a set of communications between web services and associates a user-defined set of security standards to that token. The token can be transferred in each message instead of the security standard specifics, reducing security overhead.
WS-Security	An OASIS web service standard for managing security of web service communications

References

URLs are valid as of the publication date of this document.

[1] Apache Software Foundation. (2010, February 1). *Apache Rampart—Axis 2 Security Module* [Online]. Available: http://ws.apache.org/rampart/

[2] Apache Software Foundation. (2010, November 15). *Welcome to Apache Axis2/Java* [Online]. Available: http://ws.apache.org/axis2/

[3] S. Chen, J. Zic, K. Tang, and D. Levy. "Performance Evaluation and Modeling of Web Services Security," in *Proc. 2007 IEEE International Conference on Web Services*, Salt Lake City, UT (USA), 2007, pp. 431-438.

[4] D. Sosnoski. (2009, July 7). Java Web services: The high cost of (WS-)Security. *IBM developerWorks*. [Online]. Available: http://www.ibm.com/developerworks/java/library/j-jws6/index.html

[5] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Boston, MA: Addison-Wesley Professional, 2003.

[6] OASIS Consortium. (2006, November 28). *Index of /wss/v1.1* [Online]. Available: http://docs.oasis-open.org/wss/v1.1/

[7] Perf4J team. (2010, March 23). *Overview* [Online]. Available: http://perf4j.codehaus.org/

[8] Apache Software Foundation. (2010, March 30). *Log4j: Logging Services* [Online]. Available: http://logging.apache.org/log4j/1.2/

[9] SUN Microsystems. (2010). *Java SE Documentation* [Online]. Available: http://java.sun.com/javase/6/docs/technotes/guides/vm/cms-6.html

[10] *WS-SecureConversation 1.3*, OASIS Standard, March 1, 2007 [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.pdf

[11] P. Bianco, R. Kotrmanski, and P. Merson. (2007). Evaluating a Service-Oriented Architecture. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/07tr015.cfm

[12] Sandeep Ramesh Patil. (2009, March 10). Multi-security mechanisms with multifactor authentications. *IBM developerWorks*. [Online]. Available: http://www.ibm.com/developerworks/aix/library/au-security_auth/index.html

[13] SUN/ORACLE corporation. (2010, March 29). *SUN Java Hotspot Virtual Machine* [Online]. http://java.sun.com/javase/technologies/hotspot/

[14] Apache Software Foundation. (2010, November 1). *Apache Tomcat* [Online]. Available: http://tomcat.apache.org/

[15] M. Head, M. Govindaraju, A. Slominski, P. Liu, N. Abu-Ghazaleh, R. van Engelen, K. Chiu, and M. Lewis. "A Benchmark Suite for SOAP-based Communication in Grid Web Services", in *Proc. 2005 ACMIEEE conference on Supercomputing*, p. 19.

[16] M. Head, M. Govindaraju, R. van Engelen, and W. Zhang. "Benchmarking XML Processors for Applications in Grid Web Services", in *Proc. 2006 ACM/IEEE SCJ06 Conference*, http://doi.ieeecomputersociety.org/10.1109/SC.2006.14

[17] J. Guitart, V. Beltran, D. Carrera, J. Torres, and E. Ayguadέ. "Characterizing Secure Dynamic Web Applications Scalability," in *Proc. 19th IEEE International Parallel and Distributed Processing Symposium*, Denver, CO (USA), 2005, p. 108a.

[18] D. Sosnoski. (2010, June 2). *Cleaning up SOAP Web Services* [Online]. http://www.sosnoski.com/presents/cleansoap/

[19] M. Juric, I. Rozman, B. Brumen, M. Colnaric, and M. Hericko. (2006, May). Comparison of performance of Web services, WS-Security, RMI, and RMI-SSL. *The Journal of Systems and Software [Online], 79(4)*.

[20] H. Liu, S. Pallickara, and G. Fox. "Performance of Web Services Security", in *Proc. 13th Annual Mardi Gras Conference*, Baton Rouge, LA (USA), 2005, pp. 72-78.

[21] M. Jeckle and I. Melzer. (2004, February 21). *Performance of Web Services*. [Online] Available: http://www.mathematik.uni-ulm.de/sai/ws03/webserv/PerfWS.pdf

[22] Microsoft Corporation. *Comparing Web Service Performance* [Online]. Available: http://msdn.microsoft.com/en-us/netframework/cc302396.aspx

[23] S. Vaughan-Nichols. (2002, January 7). *Fat protocols slow Web services* [Online] Available: http://www.zdnet.com/news/fat-protocols-slow-web-services/298876

[24] L. Dodds. (2002, January 16). Fat Protocols. *XML-Deviant* [Online]. http://www.xml.com/pub/a/2002/01/16/deviant.html

[25] Wikipedia. (2010, October 29). *Limit of a Sequence* [Online]. Available: http://en.wikipedia.org/wiki/Convergent_sequence

[26] National Institute of Standards and Technology. *NistNet* [Online]. Available: http://snad.ncsl.nist.gov/nistnet/

[27] TATA Consultancy Services. (2009, April 10). *Wide Area Network emulator* [Online]. Available: http://wanem.sourceforge.net/

[28] IBM. *WebSphere Application Server* [Online]. Available: http://www-01.ibm.com/software/webservers/appserv/was/

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	November 2010	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Performance Analysis of WS-Security Mechanisms in SOAP-Based Web Services	FA8721-05-C-0003

6. AUTHOR(S)

Marc Novakouski, Soumya Simanta, Gunnar Peterson, Ed Morris, Grace Lewis

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2010-TR-023

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2010-023

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Identity management (IdM) solutions in web services environments are often compared on the levels of performance and security they provide. Selecting the appropriate IdM solution for a given system or application often requires making tradeoffs between security and performance, while also considering the system's contextual and environmental requirements and constraints. This paper presents the results of a series of experiments targeted at analyzing the performance impact of adding WS-Security, a common security standard used in IdM frameworks, to SOAP-based web services. The goal of this work is to establish a baseline of performance data that can be used to explore performance/security tradeoffs in environments with complex attributes, such as resource or bandwidth limitations.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Identity management, IdM, WS-Security, SOAP-based web services	50

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL