

On Encrypting and Signing Binary XML Messages in the Wireless Environment

Jaakko Kangasharju

Tancred Lindholm

Sasu Tarkoma

Helsinki Institute for Information Technology

PO Box 9800, 02015 TKK, Finland

E-mail: {jkangash,tancred.lindholm,sasu.tarkoma}@hiit.fi

Abstract

In the wireless world there has recently been much interest in alternate serialization formats for XML data, mostly driven by the weak capabilities of both devices and networks. However, an alternate serialization format is not easily made compatible with XML security features such as encryption and signing. We consider here ways to integrate an alternate format with security, and present a solution that we see as a viable alternative. In addition to this, we present extensive performance measurements, including ones on a mobile phone, on the effect of an alternate format when using XML-based security. These measurements indicate that, in the wireless world, reducing message sizes is the most pressing concern, and that processing efficiency gains of an alternate format are a much lesser concern.

1. Introduction

In recent years two developments in the computing landscape appear to be having a significant impact on the future. One of these is the rising popularity of XML, which is now being used also for machine-to-machine messaging, most notably in the form of SOAP [24, 25]. The other is the increasing number of available mobile devices with sophisticated networking capabilities, potentially heralding an age of truly pervasive, or ubiquitous, computing [14, 19].

There has been concern that XML is not suitable for use on mobile devices, due to its verbosity and processing requirements. Because of this, there have been proposals to replace XML with an alternate “binary XML” format, which would be compatible with XML on some level, but purported to be more compact and more efficient to process. When communicating with existing systems on the fixed network, gateways can convert between this binary format and XML to permit piecemeal introduction of the new format.

However, this compatibility breaks down in the case of security features such as encryption and digital signatures.

If serialized content is encrypted, a gateway cannot convert it, so the ultimate recipient needs to be able to understand the used format. In the case of signatures the signature will be computed over the serialized form, so again the recipient will need to be able to regenerate that version.

In this paper we explore the effect of a binary format in the context of XML security, in particular to determine what, if any, benefits such a format could bring. We focus on communication between a small device using a wireless link and a server in the fixed network. While direct, peer-to-peer, communication between small devices is also an important topic, the issues of compatibility arise more strongly in the client-server case due to the number of existing deployed systems. In spite of this focus, many of our measurements should still be directly applicable also to the peer-to-peer case.

The rest of the paper is organized as follows. In Section 2 we review a usage scenario for XML security and give a brief overview of the relevant specifications. Section 3 presents the compatibility solutions and compares them with each other, and Section 4 provides our measurements. Section 5 is a review of related research, and Section 6 concludes the paper.

2. XML Security

There are several existing ways to secure network traffic, many of which can be deployed immediately without needing to worry about interoperability at the application layer. On the network layer it is possible to use IP Security [9] for authentication and encryption. Transport layer connections can be secured with SSL [3], which provides authentication and a secure communication channel. The problems with these are that they only secure network traffic, so stored data needs to be re-encrypted and re-signed, and that they lack the granularity to support some use cases.

We consider the case of a user wishing to place an order with an online retailer. The order will include the user’s identification, identification of the ordered goods, and payment information (such as a credit card). The user will

want to keep private, i.e., encrypt, the payment information. The retailer wishes to authenticate the user to make sure no fraudulent orders are placed, which requires a digital signature. Authenticating the retailer to the user is better handled at the messaging protocol level, and not at the message level.

An SSL-based solution to this case requires the decryption and verification to happen at the outward-facing system that accepts client connections. Using message-level security, the client's signature and encryption will be part of the message, allowing the outward-facing system to act only as a simple forwarder of this information. Trusted systems, which do not face the larger network, can then perform the actual security processing. This is helpful both in reducing security risks and in separating order processing from payment processing at the retailer.

These kinds of considerations are applicable in a wider context to workflow systems in general. In a workflow system a message is processed by several processors in sequence, but each processor is typically interested in only a small part of the message. With fine-grained security it is possible both to compartmentalize secret data better and to reduce processing during the workflow by targeting signatures and encrypted content to each processor individually.

These problems could be solved with S/MIME [7] by splitting the message into a multipart one with each component to encrypt or sign being its own part, as is done with email. However, since much communication is moving towards XML, there would need to be a way to represent XML documents as multipart MIME messages, and there is little to no experience in such. The only available specification for splitting an XML document as a MIME message is XOP [26], but that only considers the case of splitting Base64-encoded content out of an XML document to be transmitted in binary.

To solve the issue of fine-grained XML document security, the W3C has produced specifications for XML Signatures [23] and XML Encryption [22]. XML Signatures are complemented by Canonical XML [21], which specifies an algorithm to serialize an XML document so that "equivalent" XML documents produce the same byte sequence. This is necessary so that an XML document passed through processing can still have its signature verified. The Web Services Security [11] specification from OASIS defines how XML Encryption and Signatures are used to secure SOAP messages.

In XML Signature the content to be signed is marked with a reference. This reference also includes *transformation* methods, which are applied to the signed content to get the bytes to digest. These references are collected inside a single XML element, which is then *canonicalized*, using Canonical XML or the like, the resulting bytes are digested, and this digest is signed. If the signed data is XML, one of

the transformations applied to it will normally be a canonicalization.

Use of XML Encryption results in an element that replaces the encrypted content. Such an element contains minimally an element containing the encrypted bytes. These bytes can be either embedded in the document or given as a URI reference. The encrypted element will also contain a type, which denotes, e.g., that the encrypted content is an XML element. XML Encryption also permits transformations to be applied, but these are performed only on referenced URIs to produce the actual encrypted bytes for the decrypter, and not on decrypted data.

3. Compatibility Options

For the purposes of this paper, we have assumed that a Web service client resides on a mobile device and a server resides somewhere in the fixed network. The situation will be such that the server supports XML, but potentially not any other formats. We also assume that the mobile user would prefer to use a binary format for its presumed compactness and processing efficiency benefits.

Currently it is possible for the mobile device to use a binary format if a gateway on the network side translates between this format and XML. Various gateways have been used for wireless access in IP [12], CORBA [10], and WAP [17]. Of these, WAP is the only one that rewrites the actual messages, converting them between XML and the WAP binary format [20]. The WAP gateway also includes the WTLS protocol [18], where the gateway re-encrypts and re-signs all content passing between the mobile client and the network.

The WTLS solution can obviously be extended to handle XML security. Since the gateway already needs to handle translating between XML and the binary format, it can easily handle re-encrypting and re-signing any element content as well. Furthermore, it could even be possible to design a protocol between the client and the gateway where the client simply indicates which parts of the XML are to be secured. This would relieve the client completely of the burden of security processing, apart from the necessity of maintaining a secure channel with the gateway.

However, using this *trusted gateway* model requires the client to trust the gateway completely. While current mobile phone networks already require placing some trust in the operator, the level of trust required for the gateway still seems unacceptably high. But as long as there is no ubiquitous binary format in use, some form of gatewaying appears to be needed.

We propose a model which we call the *thin gateway*. In this model the gateway is only responsible for conversions between XML and the binary format, and will not do any security processing. This permits, among other things, a much

larger selection of potential gateways for clients, since there is no additional trust involved.

The client behavior in the thin gateway model is exactly the same as if it were communicating with a binary-aware server. Specifically, the canonicalization and transformation algorithms, as well as encrypted content types, are specified to be in the binary format. The gateway will only convert between the XML and binary formats, and will not touch these values.

On the server side modifications are required only to the XML security processing. The XML security implementation needs to recognize the algorithms for the binary format, and have these available. This limits the recognition of the binary format to a single piece of code, which may even be provided by a separate entity, instead of requiring the binary format to be integrated into XML parsing and serializing.

Since the client behaves as if it were communicating with a binary-aware server, no changes are required in the client if the server acquires full binary format support. The code required at the server needs to be included only in a place where a generic interface typically exists already, and especially the actual XML parser and serializer may remain as they are. Therefore we see the thin gateway as a useful stepping stone between XML and full binary format support.

4. Experimentation Results

We performed several experiments on XML security performance in the context of Web Services Security. Our measurements were intended to discover the effect of using a binary format instead of XML, especially with mobile phones. Furthermore, as battery life is a significant concern on mobile devices, we also measure battery consumption and break it down to its components, so that we can determine the most fruitful avenues for improvements.

4.1. Experimentation Setup

Our experimentation platform consists of three components, client, gateway, and server. The server and the gateway are running on the same machine, which has a 1.5 GHz AMD Athlon XP processor, 512 MiB of main memory, and the Debian GNU/Linux 3.1 operating system. For the client we measure on two different machines. One is a desktop system with a 3 GHz Intel Pentium 4 processor, 1 GiB of main memory, and Debian GNU/Linux 3.1. The other is a regular Nokia 7610 mobile phone. The desktop systems run Java 5.0 from Sun Microsystems, the mobile phone supports Mobile Information Device Profile (MIDP) 2.0.

We used Axis, XML-Security, and WSS4J, all from the Apache project¹, for the SOAP server and its Web Services

Table 1. Formats used in the experiments

Format	Description
Xml	XML, security processing
Xebu	Xebu, security processing
Xmlunsec	XML, no security processing
Xebuunsec	Xebu, no security processing
Xmssl	XML, no security processing, over SSL
Xebussl	Xebu, no security processing, over SSL

Security implementation. We implemented the thin gateway model of Section 3 by extending XML-Security to recognize `java` scheme URIs to indicate that the correct algorithm to use is the class given by the URI. We used the thin gateway model, as that was straightforward to implement, but the results should be applicable to any situation.

The client-side system is a simple one written by us, both to make it easy to switch XML serialization formats and to run the same system on both clients. The cryptographic algorithms we used were 3DES for symmetric encryption, 1024-bit RSA for asymmetric encryption, and SHA-1 for digests. On the server side the implementations were the default ones shipped with Java, and on the client side they were provided by the Bouncy Castle library².

For the desktop client we used a fixed two-hop network route with ICMP (ping) latency approximately 0.25 ms, and for the phone client a regular GPRS connection from a major provider with a 12-hop route and ICMP latency approximately 800 ms (the latter figures were measured from a laptop computer using the same GPRS provider as in the actual experiments).

Experiments were performed with various formats, all described in Table 1. Xebu is our binary format [8], which gives similar final sizes compared to other general-purpose binary formats. Our previous measurements indicate that the results reported below for the size and processing time are typical, and not specific to this particular test data.

The actual scenario was a simple Web service invocation over HTTP containing a number of `card` elements, each containing four sub-elements (these elements are credit card descriptions, but their actual content is less relevant to the measurements than their size, which is approximately 140 bytes in XML and 70 bytes in Xebu). The measurements reported below are all plotted against the number of `card` elements contained in a message, and averaged for a single invocation. The sequence of elements was encrypted, and after this the SOAP body was signed. The server responded with a similar message, i.e., one containing the same elements with the same encryption and signing.

We summarize the experiments that we ran and their parameter variations in Table 2. In both the desktop and phone

¹<http://www.apache.org/>

²<http://www.bouncycastle.org/>

Table 2. Description of the experiments

Experiment	Description
Desktop	Messages with 20–200 elements at 20-element increments, 100 invocations, 20 replications
Phone	Messages with 2–20 elements at 2-element increments, 10 invocations, 10 replications
Battery	Message with 10 elements, enough replications to drain the phone battery, measure number of invocations

Table 3. Message size measurements

Format	Request		Response	
	Elem	Over	Elem	Over
Xml	218	4743	163	6419
Xebu	69	1580	69	2598
Xmlunsec	151	676	121	386
Xebuunsec	69	49	70	43

experiments we began with some unmeasured invocations to eliminate any incidental startup costs. The required number of these was determined experimentally by increasing their number and observing when the measurements stabilized. The battery experiment was performed to determine the amount of energy consumed by computation and communication in this context.

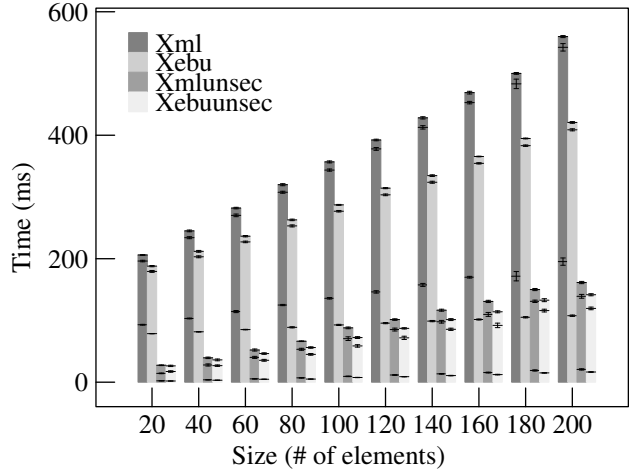
4.2. Message Sizes

We first show the sizes of the messages and their relevant components. Table 3 shows the sizes of the request and response messages in bytes, by giving the size of one element and the additional constant overhead in each message. Table 4 gives, in similar format, the number of bytes that were actually encrypted or digested.

Table 3 shows that a secured Xebu message is between 1/3 and 2/5 of the corresponding XML message in size per element. The size of a secured Xebu message quickly becomes smaller than that of an unsecured XML message. The large overhead of the secured messages is due to the

Table 4. Encryption and signing size measurements

Measurement	Encrypt	Decrypt	Sign	Verify
Xml elem	161	121	218	163
Xebu elem	69	69	69	69
Xml over	128	273	715	1502
Xebu over	15	16	111	257

**Figure 1. Times taken in the desktop experiment on client, remote, and network.**

Web Services Security SOAP header. Finally, we note that since XML requires binary data (such as encrypted content) to be Base64-encoded, the signed content in Table 4 is one third larger than encrypted content, whereas with Xebu there is no difference.

4.3. Timing Measurements

For the timing measurements we measured separately the time taken for processing the messages on the client, the time that the gateway and server spent processing, and the communication time. These times are shown in Figures 1 and 2, where the dividing lines in the bars show client processing, remote processing, and communication from bottom to top. Error lines are marked at one standard deviation.

Note that both figures have three lines marking processing times. However, in the phone experiment case the time taken for remote processing at the gateway and the server is such a negligible part of the whole that its line is indistinguishable from the line drawn for local processing.

We can also see that the time taken for communication in the phone case is much higher for the security-enabled formats. By examining network packet dumps we can see that the messages are sent in TCP segments of maximum size 1348 bytes. Since the sizes of the messages with security processing are so much larger, this generates additional round trips. With the aforementioned latency of the GPRS network this becomes clearly visible in the timings.

As with the size measurements, we also take a more detailed look into the security-enabled messages. Figures 3 and 4 show the times taken processing encrypted data and message digests. The output bar shows symmetric key generation, key encryption, data encryption, and digest compu-

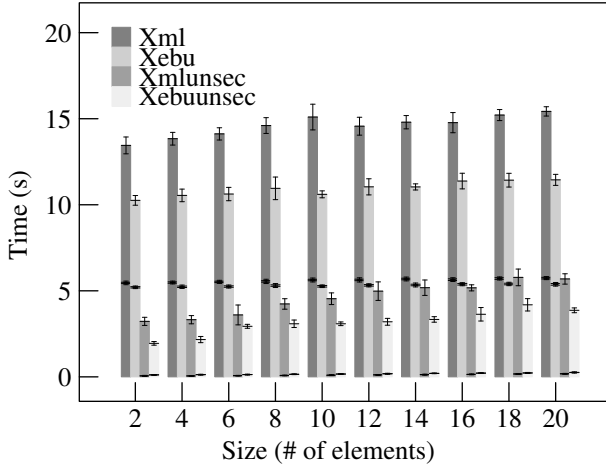


Figure 2. Times taken in the phone experiment on client, remote, and network.

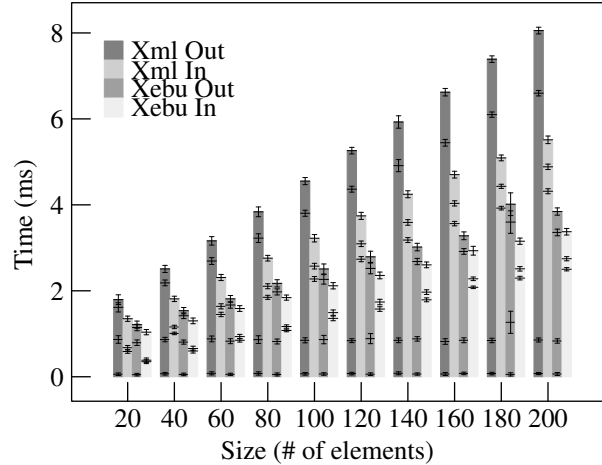


Figure 3. Time taken in the desktop experiment for fast security operations.

tation. The input bar shows data decryption, digest verification, and signature verification. Both of these sequences are from bottom to top.

For large messages, the dominant output component is data encryption, because its processing requirements grow with message size. For the smaller messages of the phone experiment the constant-time RSA public-key operation of symmetric key encryption dominates. For input processing the most expensive one is again data decryption with the RSA public-key operation of signature verification dominating for smaller sizes.

There is much variation in many of the timings on the phone, which is due to the coarse granularity of the phone’s internal clock. The actual measurement numbers show that the phone is capable of measuring time only in 15- or 16-millisecond increments. Since many of the operations only take a few such increments, we see both no variation and high variation, but very little low variation.

From the gathered data we can estimate that for the same processing time as a single key encryption we can encrypt symmetrically approximately 3.5 kilobytes of data on the desktop, and 4.2 kilobytes on the phone. Similar numbers hold for the verify/data decryption pair of operations.

Finally, we show the time taken by RSA private key operations, i.e., signature computation and key decryption, in Table 5. These times do not depend on the message size at all. As we can see when comparing to Figures 3 and 4, these two operations consume several times the time spent on all other security-related processing. Extrapolating from Figure 3 we see that in the same time as one of these operations we could potentially encrypt at least 250 kilobytes.

Furthermore, we note that for the smallest messages on the phone these operations consume up to 85 % of the pro-

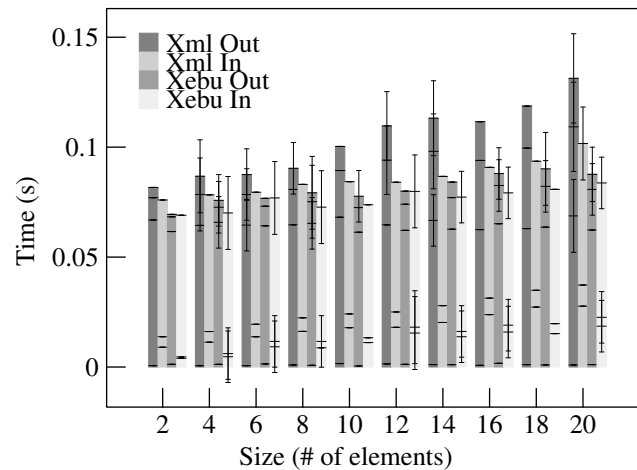


Figure 4. Time taken in the phone experiment for fast security operations.

Table 5. RSA private key operation sizes and times

Measurement	Size (B)	
	Desktop (ms)	Phone (s)
Signing size	35	
Keydec size	128	
Xml Signing	35.70±0.01	2.19±0.01
Xml Keydec	35.52±0.04	2.26±0.00
Xebu Signing	35.71±0.03	2.20±0.01
Xebu Keydec	35.34±0.03	2.23±0.00

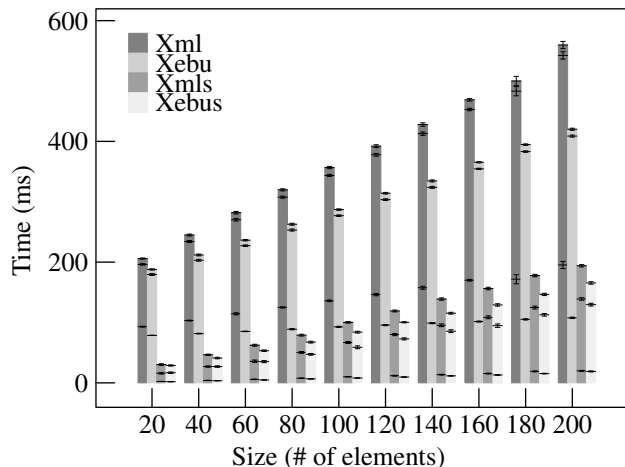


Figure 5. Times taken in the desktop experiment on client, remote, and network for XML security and SSL.

cessing time on the client and almost 50 % of the total invocation time. Therefore it seems prudent to attempt to avoid repeated RSA private key operations. However, there is no reason to avoid RSA public key operations: as we saw earlier, the requirements of these are quite comparable to other processing.

4.4. Comparison with SSL

While SSL is not appropriate for all use cases, as mentioned in Section 2, it is a widely deployed security solution, and very suitable for many other cases. For this reason we also ran our measurements using HTTP over SSL without XML-level security. This was done by replacing the `http` URLs in Java’s standard connection opening with `https` URLs. The algorithms used in SSL were forced to be the same as those with XML security, namely 3DES, RSA, and SHA-1.

We show the results of comparing SSL with our security-enabled formats in Figures 5 and 6, in the same way as the comparison with completely unsecured formats in Figures 1 and 2. To reiterate, each bar shows the time taken for local processing on the client, the time taken for processing at the gateway and the server, and the time taken for communication, from bottom to top in this order.

Comparing to the regular HTTP case on the phone, we note that SSL adds approximately 2 seconds of processing to the invocation time. A network packet dump reveals this to be the SSL handshake cost, which takes two network round trips. Otherwise the measurements remain the same. However, we also note that the first SSL handshake takes 6 seconds due to the key exchange; on later invocations the

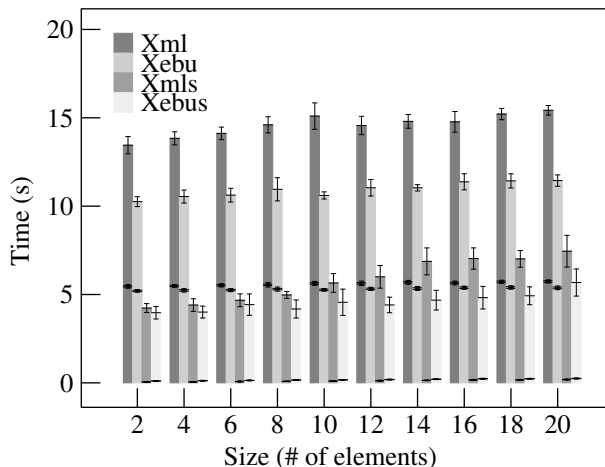


Figure 6. Times taken in the phone experiment on client, remote, and network for XML security and SSL.

session is reused, so the only overhead consists of the two network round trips of the abbreviated SSL handshake.

The SSL experiment was, however, somewhat problematic on the phone. An observation that we made concerned the unreliability of the connection: how many invocations are received by the gateway and not returned correctly to the client? For the regular HTTP case this rate remained at few tenths of a percent of the number of total invocations, but with SSL we observed a rate of 6–7 percent of such dropped connections. The most probable cause is a too stringent network timeout somewhere, but based on our past experience with phones we cannot rule out the possibility of an unreliable SSL implementation on the mobile phone.

4.5. Battery Consumption Measurements

Our final experiment was the repeated invocation of a 10-element message from the phone until its battery ran out. The measurement in this case was the number of invocations performed at the server. While we ran this measurement on all formats, we only use the `Xml` and `Xebu` measurements, since the small processing times of the other cases were not sufficient to get low variation in the results.

From the invocation counts we can derive some approximations of the relation between the energy consumption of computation versus communication. Let m be the amount of energy required to communicate one byte of data and let p the amount of energy required to run one millisecond on the processor. If E is the total amount of energy on the phone, we get the formula

$$E = i \cdot (s \cdot m + t \cdot p) \quad (1)$$

where s is the amount of data per invocation, t is the time spent processing one invocation, and i is the number of invocations.

By taking two measurements for different formats, s_1, s_2 for size, t_1, t_2 for processing time, and i_1, i_2 for number of invocations, and inserting them into Equation (1) we can solve for m and p . Our main interest is in the ratio $\frac{m}{p}$ that gives the relative cost of communication versus computation (the unit being $\frac{ms}{B}$). Solving for m and p and dividing, we get

$$r = \frac{m}{p} = \frac{i_2 \cdot t_2 - i_1 \cdot t_1}{i_1 \cdot s_1 - i_2 \cdot s_2}. \quad (2)$$

To get numbers for this equation, we repeated the Battery experiment five times for both Xml and Xebu. The number of invocations for Xml varied between 842 and 979, and the number for Xebu between 1444 and 1666. The total amount of data sent over the network per invocation, including lower-layer protocol headers, came to 17385 bytes with Xml and 8703 bytes with Xebu. The processing time for a single invocation can be computed by subtracting the communication time from the total time, and it comes to approximately 5600 ms for Xml and 5230 ms for Xebu.

With these numbers we can calculate r from Equation (2) using each pair i_1, i_2 where i_1 is an Xml measurement and i_2 is a Xebu measurement. From these measurements we get the values

0.47, 1.06, 1.44, 2.86, 28.76

as the minimum, 25th percentile, median, 75th percentile, and maximum, respectively, for r .

The calculated numbers for r give the number of milliseconds that the processor can run for the cost of communicating one byte. The Nokia 7610 has a 123 MHz ARM processor, so taking the 25th and 75th percentiles we can estimate that with the energy cost spent on communicating one byte, it is possible to perform 130,000 to 350,000 processing cycles. The minimum and maximum values would be 58,000 and 3.5 million cycles.

5. Related Work

There already exist several binary formats for XML data [13, 15] and sufficient interest exists that the W3C is working on standardizing one³. As the measurements above indicate, in this environment message size reduction is the most important characteristic of such a format. The general-purpose formats, applicable to any XML data, achieve approximately 50 % reduction in size for small messages [8]. Schema-based formats can achieve up to a 95 % reduction [2], but they are tied to a specific schema.

³<http://www.w3.org/XML/EXI/>

The other security solutions mentioned in Section 2, IP Security, SSL, and S/MIME, have been the targets of prior performance measurements on handheld devices as well [1]. The processing requirement results appear to be in line with ours, but due to the use of Wireless LAN and no mention of network latencies, total communication times are not directly comparable. This work also includes energy consumption measurements performed similarly to ours, but does not consider how consumed energy is split between computing and communication.

We note that existing detailed measurements of XML Signatures [16] indicate that most of the time in signature processing is actually spent on canonicalization. As our experimental system was designed so that it wrote and read everything directly in canonical form, this effect is not visible in the figures of Section 4.3, but we could observe some of it on the server while running the experiments.

The measurements that we performed were intended to reflect a single message exchange. For a longer-term exchange of messages, it is beneficial to establish a security context, such as IP Security's security association or SSL's secure tunnel (we saw partial effects of this with the SSL session reuse). At the Web service level such an establishment method is defined by WS-SecureConversation [6].

An architecture for secure Web-services-based communication for the pervasive environment is given in [5]. This system uses Web Services Security for its security needs, and the authors' conclusion is that security interoperability is possible even with low-cost devices. Unlike the off-the-shelf components used in our measurements, this system's Web service implementation is a special-purpose one, written especially for the embedded devices they are targeting. Energy consumption is briefly considered and requirements for processing calculated, but there is no breakdown of costs.

6. Conclusions

Based on the measurements in Section 4 we can see that the overwhelmingly most time-taking operation is asymmetric encryption. With RSA this is limited to the private key operations, but with other systems both private and public key operations can be costly. As in many cases only RSA public key operations are needed (e.g., for certificate verification), RSA seems the most viable candidate for this environment.

From the energy consumption measurements we can also infer that compressing XML messages is a very worthwhile thing to do, at least in this kind of environment. In addition, as we saw, the amount of energy that we can spend computing for each byte saved is quite large, so processor-heavy compression schemes are not to be disregarded. Therefore any purported gains in processing efficiency from switching

to a binary format do not appear very significant. The same applies to improvements in XML processing efficiency.

As an example, our previous measurements on the effectiveness of binary XML [8] indicate that gzip on top of XML gives, for 3-kilobyte messages, a 75 % reduction in size for the cost of 0.40 milliseconds of additional processing, which is a clear benefit in light of Section 4.5. Note, however, that when combined with security processing, compression needs to be applied before encryption and this would need to be indicated (as we noted already, XML Encryption does not offer any method for doing this).

In our measurements the overhead of SSL is not very significant compared to other processing; this overhead would further decrease if the server did other processing. We therefore agree with the analysis of [4] that SSL is fully usable in the wireless world. Furthermore, we also agree with [16] in that SSL should be used in preference to XML-based solutions if the required security semantics allows this.

Based on our experience in the area, we believe that the adoption of an alternate serialization format for XML in the wireless world is very likely in the near future. As security is vitally important in the modern networked world, it must not be compromised or lessened. In our view the thin gateway model is a valid method for the initial inclusion of XML-based security in the case where an alternate format is adopted. Independently of that, however, we see the processing requirements of cryptography and energy requirements of large messages to be the major issues in this field.

References

- [1] P. G. Argyroudis, R. Verma, H. Tewari, and D. O'Mahony. Performance analysis of cryptographic protocols on handheld devices. In *Third IEEE International Symposium on Network Computing and Applications*, pages 169–174, Aug. 2004.
- [2] M. Cokus and D. Winkowski. XML sizing and compression study for military wireless data. In *XML Conference and Exposition*, Baltimore, USA, Dec. 2002.
- [3] A. O. Freier, P. Karlton, and P. C. Kocher. *The SSL Protocol Version 3.0*. Netscape Communications, Nov. 1996.
- [4] V. Gupta and S. Gupta. Securing the wireless Internet. *IEEE Communications Magazine*, 39(12):68–74, Dec. 2001.
- [5] J. Helander and Y. Xiong. Secure Web services for low-cost devices. In *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 130–139, May 2005.
- [6] IBM, Microsoft et al. *Web Services Secure Conversation Language (WS-SecureConversation)*, Feb. 2005.
- [7] Internet Engineering Task Force. *RFC 3851: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification*, July 2004.
- [8] J. Kangasharju, S. Tarkoma, and T. Lindholm. Xebu: A binary format with schema-based optimizations for XML data. In A. H. H. Ngu, M. Kitsuregawa, E. Neuhold, J.-Y. Chung, and Q. Z. Sheng, editors, *6th International Conference on Web Information Systems Engineering*, volume 3806 of *Lecture Notes in Computer Science*, pages 528–535, New York, USA, Nov. 2005. Springer-Verlag. Short paper.
- [9] S. Kent and R. Atkinson. *RFC 2401: Security Architecture for the Internet Protocol*. Internet Engineering Task Force, Nov. 1998.
- [10] Object Management Group. *Wireless Access and Terminal Mobility in CORBA, Version 1.2*, May 2005.
- [11] Organization for the Advancement of Structured Information Standards. *Web Services Security: SOAP Message Security 1.0*, Mar. 2004.
- [12] C. Perkins. *RFC 2002: IP Mobility Support*. Internet Engineering Task Force, Oct. 1996.
- [13] P. Sandoz, A. Triglia, and S. Pericas-Geertsen. Fast Infoset. On Sun Developer Network, June 2004.
- [14] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8(4):10–17, Aug. 2001.
- [15] J. Schneider. Theory, benefits and requirements for efficient encoding of XML documents. In *W3C Workshop on Binary Interchange of XML Information Item Sets*. World Wide Web Consortium, Sept. 2003.
- [16] S. Shirasuna, A. Slominski, L. Fang, and D. Gannon. Performance comparison of security mechanisms for Grid services. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 360–364, Nov. 2004.
- [17] WAP Forum. *Wireless Application Protocol: Architecture Specification*, 2001.
- [18] WAP Forum. *Wireless Transport Layer Security Specification*, 2001.
- [19] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, July 1993.
- [20] World Wide Web Consortium. *WAP Binary XML Content Format*, June 1999. W3C Note.
- [21] World Wide Web Consortium. *Canonical XML Version 1.0*, Mar. 2001. W3C Recommendation.
- [22] World Wide Web Consortium. *XML Encryption Syntax and Processing*, Dec. 2002. W3C Recommendation.
- [23] World Wide Web Consortium. *XML Signature Syntax and Processing*, Feb. 2002. W3C Recommendation.
- [24] World Wide Web Consortium. *SOAP Version 1.2 Part 1: Messaging Framework*, June 2003. W3C Recommendation.
- [25] World Wide Web Consortium. *SOAP Version 1.2 Part 2: Adjuncts*, June 2003. W3C Recommendation.
- [26] World Wide Web Consortium. *XML-binary Optimized Packaging*, Jan. 2005. W3C Recommendation.