Published by IEE Computer Society, IT Professional, Jan/Feb 2004

Web Services

An XML Alternative for Performance and Security: ASN.1

Darren Mundy and David W. Chadwick

Performance tests of XML and ASN.1 found that signed complex XML messages can be up to 1,000-percent slower to decode than an equivalent ASN.1 message

Over the past few years, Extensible Markup Language (XML) has become the preferred syntax for transferring business information across the Internet, receiving widespread endorsement from major IT industry players such as Sun Microsystems, IBM, and Microsoft, as well as many governments. The UK government, for example, is firmly committed to making XML the basis for electronic transactions through their e-Government Interoperability Framework, e-Government Metadata Framework, and GovTalk program. //Nancy: Do you know if one of the items in the "Resources" sidebar relates to this? If not, please ask the author for have a reference here on where you this info came from. We should have some of the references in parentheses within the body of the article, in addition to the Resources sidebar.//Indeed, many UK government agencies are investigating the adoption of XML as the protocol for e-services, such as medicinal prescribing, contract tendering, and personal health records.

But although many have touted XML as a true e-business enabler, rarely does a single technology suit every IT scenario. In XML's case, the drawback is performance. XML is an uncompressed textual syntax that remains in human-readable form from creation to deletion. This characteristic degrades performance because it increases data-stream size, which in turn increases data transfer time, and it takes time to construct and deconstruct the syntax.

In the Internet world, performance is one of user satisfaction's critical components , so having some quantifiable comparison with other transmission syntax would be valuable input to any XML adoption decision. Motivated by this concern, we devised a series of tests to compare XML messages with equivalent messages written in Abstract Syntax Notation One with Basic Encoding Rules (ASN.1/BER). ASN.1 is a protocol specification language, first standardized in 1984. The encoding rules, of which BER is only one variety, are used to condense the ASN.1 textual representation into a binary data stream. Many Internet-based applications, air and road traffic control systems, mobile phones, and power grids use ASN.1. Overall, ASN.1 (regardless of its encoding rule set) tends to emphasize efficiency, while XML is targeted more at facilitating application development.

The goal of our tests was to gather data to inform the UK Department of Health, which has requested the use of XML for encoding electronic prescriptions. In short, we wanted to see if XML or ASN.1 was the more efficient mechanism. Our tests cover the creation, transmission, and retrieval performances of the two languages. The test bed was a trial electronic prescription system already in place that uses an application certificate to transmit a prescription. This certificate is similar to a public-key certificate—a fundamental component of Web security—but as its name suggests, an attribute certificate contains one or multiple user attributes, rather than the user's public key. We chose to focus solely on the performance issues of XML and ASN.1, deliberately omitting application-development issues such as the ease of building applications and debugging protocol errors. Although that seems unfair, given that XML's strength is application development, we felt that someone should examine performance, which is end users' main concern. Too many IT projects fail because developers do not give adequate weight to user needs, such as the desire for high performance.

LANGUAGE CHARACTERISTICS

To appreciate our test format and results, it helps to know something of the two languages' structure and intent, as well as their strengths and weaknesses.

ASN.1

ASN.1 describes the structure and syntax of transmitted information content, letting programmers define the abstract syntax of a data element (or data type). The abstract syntax describes the syntactical structure and typed contents of data that will be transmitted across some medium. The language is based firmly on the principles of *type* and *value*, with a type being a (nonempty) set of values. The type defines what values the program can subsequently send at runtime, and the value is what the program actually conveys at runtime. For example,

```
AllowedAccess ::= BOOLEAN
```

is the abstract syntax for a data element of type AllowedAccess, whose values at runtime can be one of the type Boolean values TRUE or FALSE and where TRUE might actually be a value conveyed within a binary-encoded data stream at runtime. Binary encoding will be in hexadecimal notation, 01 01 FF, for example, with the first octet signifying type Boolean, the second its length, and the third its value.

The primary performance advantage of ASN.1 is that it encodes values *before* transmission using one of many encoding mechanisms, such as BER, the Distinguished Encoding Rules (DER), the Packed Encoding Rules (PER), or the recently introduced XML Encoding Rules (XER). The encoding rules specify how the values of the abstract data types are converted into byte strings ready for transfer. The recipient must usually be aware of the type definition before receipt because the sender does not transfer the type definition, but requires the recipient to infer it from the message-exchange context. BER are very efficient and create *type, length, value* (TLV) byte streams, so after reading the length field, the recipient knows how many data bytes the value comprises. PER is not based on TLV streams, which makes it even more efficient than BER, and it can provide even greater optimization. For example, PER never encodes the value's length unless it has to. If something has a fixed length, it does not encode the length field. For some data types, such as Boolean, BER specifies alternative encoding rules. DER is a subset of BER; it removes all alternative encoding and mandates one particular encoding rule for every data type. (Later, we describe our rationale for choosing to compare XML with ASN.1/BER and present the results of comparing rule sets.)

During transmission, the ASN.1 data stream is never in a human-readable form unless the programmer has used ASN.1/XER. Only after the ASN.1 compiler has transformed it into some local data display format can humans easily read it.

XML

XML, a subset of the Standard Generalized Markup Language, is a set of infinitely extensible rules that let programmers encode data values in text format. XML documents contain information for transmission, which consists of markup—a rough correspondence to tag and length in BER's TLV encoding—and character data—roughly the value part in BER's TLV encoding. Programmers can impose constraints on the XML document structure by using document type definitions (DTDs) or XML schema, both of which describe the allowed markups that a conformant XML document can contain. A DTD might contain a definition such as

```
<!ELEMENT allowedAccess (#PCDATA)>
```

which states that the element allowedAccess is of type parsed character data (PCDATA) and that an XML parser can process it on receipt. <allowedAccess>TRUE</allowedAccess> might be a value of allowedAccess sent in an XML document.

Even this small example illustrates why XML is verbose and consequently creates large data streams. The data stream for this definition is 35 bytes, compared to 3 bytes for the same example in ASN.1 BER. XML is transferred in textual format with no binary encoding or compression. Further, the recipient must examine every byte received to determine a data value's end. Application programmers prefer XML, however, because it remains in a constant human-readable format throughout the process.

Similarities and differences

In some sense, DTDs and XML schemas map to ASN.1's abstract syntax type definitions, and XML documents map to ASN.1-encoded byte streams. Outside this mapping, ASN.1 and XML/DTDs have significant differences, mostly in their design goals. From a user perspective, XML/DTDs are restrictive because they lack any concept of data type, while ASN.1 is rich with built-in data types and support for user-defined data types. From a performance perspective, XML is verbose because its design emphasizes human readability, while ASN.1 encoding rules (except XER) are more efficient because their design goal was optimal performance. From an application programmer's perspective, XML is easier to debug, because programmers can read the data stream without any special software tools. Trying to read an ASN.1 BER or PER byte stream, on the other hand, is very complex, although programmers can use free tools, such as dumpasn1, to display ASN.1 data in its original source form.

Finally, the XML 1.0 specification is much newer, simpler, and easier to understand than the ASN.1 documentation, which has gone through several iterations and thus contains many more sophisticated features.

TESTING INFRASTRUCTURE

For our performance analysis of XML and ASN.1/BER, we used client and server application programs in Java, which exchange messages using attribute certificates. The client's task is to create the attribute certificate (in either XML or ASN.1) and transmit it to the server application through standard sockets. The server first verifies the certificate and then parses it into a data structure for easy access to any of its data elements.

As Figure 1a shows, the main part of an attribute certificate is the *attribute-certificate information*, which contains details about the issuer, the recipient, the validity time, and so on, as well as embedded user attributes. Figure 1b shows the structure of the attribute-certificate information. The sender (client) stores the prescription structure as an attribute within the attribute-certificate information, which also contains the signature, signature method, and signature value. All these are amalgamated to form the attribute certificate.

```
Attribute Certificate Info ::= SEQUENCE {
version     AttCertVersion,
holder      Holder,
issuer      Issuer,
signature   AlgorithmIdentifier,
attrCertValidityPeriod AttCertValidityPeriod,
attributes  SEQUENCE of Attribute,
issuerUniqueID  UniqueIdentifier OPTIONAL,
extensionsExtensions OPTIONAL
}
```

(b)

Figure 1. Attribute certificate. (a) Certificate structure and (b) content of attribute-certificate information.

Java can produce attribute certificates of any complexity, so for our tests we used three attribute variations—very complex, semicomplex, and simple. The complex attribute was an auditCertificate structure, the semicomplex attribute was a UK Department of Health EtpPrescribe structure, and the simple attribute contained a simple Boolean attribute value. We produced the signature for the ASN.1 certificates using standard Java security classes in conjunction with our local Entrust (www.entrust.com) public-key infrastructure (PKI), from which we also obtained the private key for signing. We produced the signature for XML certificates using the XML Security Suite (XSS) available from IBM Alphaworks.

Client application

Our aim was to test the client and server applications for overall performance, including timing. The client application generated ASN.1 attribute certificates using simple, semicomplex, and complex ASN.1 attributes—both signed and unsigned. It also produced XML attribute certificates using simple, semicomplex and complex XML attributes—both signed and unsigned. Finally, it compressed all XML-generated certificates.

To generate the attribute-certificate structure in ASN.1, we created the attribute-certificate information and then used BER to encode it. From this, we generated an electronic signature and placed the unencoded attribute-certificate information, as well as the signature type and value previously generated, in the BER-encoded attribute-certificate structure.

To do the same in XML, we passed the XSS tool an attribute-certificate-information object, which it used to canonicalize the received XML and generate a signed XML document. The canonicalization transforms the XML against a set of rules, which ensures that any syntactical differences between equivalent XML documents do not result in different binary representations. If XML signature operations do not undergo this transformation, signature validation is likely to fail. The sender can include the public key as part of the signed XML, but because we did not include the public key in the ASN.1 structure, we removed public-key transmission from the XML structure as well.

Server application

The server application verified ASN.1 attribute certificates using simple, semicomplex, and complex ASN.1 attributes—both signed and unsigned. It also verified XML attribute certificates using simple, semicomplex, and complex attributes—both signed and unsigned. Finally, it decompressed all XML-generated certificates.

The recipient (server application) verifies signatures using the sender's public key, which is available from its Entrust profile. Once it verifies the signature, the recipient parses the data into a data structure that any data member can easily access. During parsing, the recipient parses the ASN.1 structure into the Java class structure, which it can then use to encode and decode ASN.1 attribute-certificate structures. To parse the XML message, the recipient maps the data structure to a document object model (DOM) structure, which is the basis for creating the entire XML structure in memory.

To verify the ASN.1 attribute certificate, the recipient must retrieve the value of the BER-encoded attribute-certificate information from the attribute-certificate structure and compare it against the supplied signature value using the signer's public key for decryption. To verify the XML certificate, the server must parse the structure to get the signature value. The XSS tool then verifies the signature against the public key and the XML content.

Test bed

All performance comparisons took place on the same machine, a P3 workstation with a speed of 650 MHz and a RAM of 256 Mbytes, running the Linux operating system. We repeated each test 100 times to allow for statistical variations, using the three attribute sizes to verify that performance does indeed degrade as the attribute certificate gets more complex. We also wanted to see if compressing the XML attribute certificates before transfer would significantly affect XML transmission speeds and XML processing's overall performance.

PERFORMANCE RESULTS

Perhaps the most limiting factor of XML performance is XML's output size. As Table 1 shows, XML creates data blocks approximately an order of magnitude greater than those created from BER-encoded ASN.1. Even when XML data is compressed, the XML data block can be triple that of the equivalent ASN.1.

Table 1. Comparison of data-block sizes.

	Data block size		Zipped file size
Attribute complexity	ASN.1, signed (bytes)	DOM XML,* signed (bytes)	Zipped DOM XML, signed (bytes)
Simple	384	3,704	913
Semicomplex	1,060	7,043	1,737
Complex	1,483	19,184	4,733

*Generated by the document object model.

Transmission time

Transmission time is a crucial factor in many real-time applications, and for an electronic prescription system, the time it takes a pharmacy to receive a prescription is crucial for normal operations. Even so, we decided not to test actual transmission times between two machines because link speed is rarely constant, and our aim was to remove any variable environmental factors from our testing. Instead, we used a theoretical constant link speed of 64 kilobits per second (kpbs) and 256 kbps, speeds typical of what the latest technology or broadband connection might achieve in transmitting data to a pharmacy, with no lost packets. Table 2 shows the theoretical results with these constant link speeds.

Table 2. Theoretical transmission times with a constant link speed of 64 Kbps and 256 Kbps.

Attribute complexity	ASN.1, signed with 64 Kbps (ms)	ASN.1 signed with 256 Kbps (ms)	DOM XML, signed with 64 Kbps (ms)	DOM XML signed with 256 Kbps (ms)
Simple	47	12	452	113
Semicomplex	129	32	860	215
Complex	181	45	2,342	585

The performance over an actual modem link could be worse than the times in Table 2, depending on the transmission protocol and any limitations it imposes on packet size. For example, the maximum segment size in TCP/IP (transmission control protocol/Internet protocol) on an Ethernet network is generally set at 1,460 bytes. Two of the ASN.1 data-blocks for a simple attribute (see Table 1) could fit into a single TCP/IP segment. All the XML data sizes are greater than the maximum segment size, on the other hand, which means that the transmission times would be longer than those in Table 2 because the data would need to be split across several packets. The time for the complex attribute would require 14 packets.

In fact, a theoretical comparison of ASN.1 against readable, unzipped XML reveals that XML is 9.6 times slower than ASN.1 for the simple attribute with a 64-kbps link. This is strong evidence that data size and the corresponding transmission times are potentially major XML performance limitations, relative to ASN.1/BER.

Compressing the XML data block before transmission reduces that performance deficit. Table 3 gives the transmission times when we maximally compress the XML data-block before its transmission using Java classes. We measured the time to zip and unzip the XML block using Java calls to get the elapsed process time. As the table shows, zipping the XML block before transfer and unzipping it after significantly increases transmission

performance. The total XML transaction time is then only 2.0 to 6.5 times slower than the ASN.1/BER transmissions.

Table 3. Theoretical transmission times for XML data blocks with zipping and unzipping (again over constant link speeds of 64 kbps/256 kbps).

Attribute complexity	Zipping time for DOM XML, signed (ms)	Unzipping time for DOM XML, signed (ms)	Total time with 64 kbps (ms)*	Total time with 256 kbps (ms)*
Simple	40	10	161.45	77.86
Semicomplex	45	15	262.03	103.01
Complex	45	15	627.76	194.43

*Total time includes zipping, unzipping, and theoretical transmission times.

Encoding and decoding of signed data

Tables 4 and 5 show the time it takes to encode and decode signed data and that ASN.1 signing and verification far outperforms the same tasks in XML. Sender ASN.1 encoding outperforms XML encoding for simple attributes by 20 percent rising to 80 percent for complex attributes, as Table 4 shows. For recipient decoding and signature verification, shown in Table 5, the difference is even more pronounced, with ASN.1 outperforming XML by 450 percent for a simple attribute and close to a 1000 percent for the complex attribute.

Table 4. Time sender takes to construct and encode signed data (ms)

Construction and Encoding Time (ms)

Attribute complexity	ASN.1	DOM XML	DOM XML relative to ASN.1 (Percentage)*
Simple	94.82	113.36	+20
Semicomplex	100.28	125.85	+26
Complex	102.79	184.12	+80

*Percentage of time longer (+) or shorter (?)

Table 5. Time recipient takes to decode and deconstruct signed data.

Decoding and deconstruction time (ms)

Attribute complexity	ASN.1	DOM XML	Relative to ASN.1 (percent)*
Simple	5.92	26.62	+350
Semicomplex	6.01	38.96	+550
Complex	6.16	67.22	+1,000

*Percentage of time longer (+) or shorter (?)

Because the XML signature process is not standard yet, the signing software may well be immature and thus not optimized. Consequently, the speed of XML signing and verification could improve. However, the increase in ASN.1 timing is minimal as the attribute complexity increases; whereas the increase in the XML timing is pronounced, as Table 6 shows.

Table 6. Average performance decrease as complexity of signed attributes changes.

Attribute complexity	ASN.1 recipient (percent)	ASN.1 sender (percent)	XML recipient (percent)	XML sender (percent)
Simple	100	100	100	100
Semicomplex	101.52	105.75	146.35	111.01
Complex	104.05	108.40	252.52	162.42

When we combine the theoretical transmission time over a 64 kbps link with the encoding and decoding times (Table 2 plus Tables 4 and 5), there really is no contest between ASN.1 and XML. ASN.1 always outperforms XML by approximately an order of magnitude.

Encoding comparison

Our choice of BER encoding for the ASN.1 data might seem puzzling, given that PER encoding creates a more optimized data stream. Our main reason for using BER encoding is that we could find no zero-cost implementation of PER in Java. Consequently, our performance results for ASN.1 might not be the best ones possible, and might have been improved with a different set of encoding rules such as PER. To test this hypothesis, we experimented with a trial version of a commercial ASN.1 tool—OSS ASN.1 tools for Java. We then compared ASN.1 encoding mechanisms—BER, DER, PER aligned, PER unaligned, XER, and canonicalized XER—when encoding a simple prescription data structure without secure operations (unsigned). We had to use unsigned data structures because the facility to produce signed data in all the encodings was not available. Table 7 presents the results of our experiment. We repeat the previous results for BER encoding with secure operations (signed) in the last row for comparison.

Table 7. Performance comparison of ASN.1 encoding mechanisms for unsigned data.

Encoding mechanism	Time to encode (ms)	Time to decode (ms)	File size (bytes)
BER	1.87	4.93	538
DER	1.8	4.92	538
PER aligned	2.854	5.158	455
PER unaligned	2.812	5.008	405
XER	6.6	22.0	2,426
Canonicalized XER	5.498	18.935	1,953
BER, signed	*100.28*	*6.01*	*1,060*

As the table shows, XER performs significantly worse than BER, DER and PER. The act of digitally signing a message adds about 500 bytes to the encoded message size, and accounts for most of the encoding processing time, whereas signature verification incurs a comparatively small overhead. We strongly suspect that this pattern will hold for DER and PER encoding, but we cannot confirm the same for XML encoding because of its significantly worse performance characteristics.

Comparison of parsing tools

Using the XML DOM model to deconstruct attribute certificates may not be the best choice for evaluating XML performance, but the XSS tool provides no other parser. The general alternative to using DOM is to use the Simple API for XML (SAX) model—an event based API that reports parsing events, such as an element's beginning and end, to the calling application. Unlike the DOM model, SAX does not keep a tree structure of the document in memory. It therefore requires less processing time, but is limited to the retrieval of a single field. To see how performance differed between SAX and DOM, we ran a simple parsing experiment on a prescription data structure without any secure operations, using the two sample programs that come with the Xerces parser, DOMCount and SAXCount. We found the DOM parsing tool took more than twice as long to parse the structure as the SAX parsing tool (and longer than the IBM DOM tool we used in all our other performance measurements). Consequently, if we had used SAX instead of DOM, we would have seen a significant drop in deconstruction time—up to half the time DOM took to deconstruct the complex attribute certificate. If the application must retrieve more than one field from the data structure, however, the SAX parser cannot be used.

PRICE OF POOR PERFORMANCE

XML did not become popular by accident. Both senior managers and developers find it easy to manipulate and understand. We do not dispute this advantage, but we feel that performance is key to end users. Thus, a real-time system that deals in multiple transactions per second and requires strong authentication through digital signatures is not a good fit with the XML protocol. Within an electronic prescription system, for example, the performance hits could be critical. Pharmacists receiving the digitally signed prescriptions are precisely the users who require optimum performance, as they try to dispense drugs rapidly in a busy pharmacy. General practitioners (GPs) would experience delays with creating and electronically signing prescriptions. The administration time for each prescription would also be far greater, which could perhaps delay payments to pharmacists and healthcare prescribers and generally create dissatisfaction with the system. Thus, end users such as GPs and pharmacists would see no advantage to using XML instead of ASN.1.

Worse, user dissatisfaction on both ends might ultimately lead to the system's rejection. Because end users are aware of system performance, and not the underlying data-encoding mechanisms, we believe that performance should be a major concern in designing this kind of system. Others seem to agree, given that Sun Microsystems has recently expressed a preference for ASN.1 in developing fast Web services. //Nancy: I think a reference should go here. Is it in the Resources section?// Yes, it is—"Fast Web Services" under development issues and tools. Darren, would most developers know this? If not, you can kind of tell by the wording in the Resources box. I'd

rather not have just one reference in the entire article pointing to the Resource box.

Our test results are not perfect, but they should give managers and developers something to consider, especially if they are planning to implement secure applications, where performance is a key success factor. If signed complex XML messages can be up to 1,000 percent slower to decode than an equivalent ASN.1 message, the choice (or not) of ASN.1 could significantly affect the system's acceptance by its prospective users. <end article>

Resources

Language specifications and characteristics

- ITU-T Recommendation X.680, ISO/IEC 8824-1:2002, Abstract Syntax Notation One (ASN.1): *Specification of basic notation.*
- ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002, ASN.1Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER).
- ASN.1 and XML specifications and examples; http://sec.isi.salford.ac.uk/EPP/public/comp.html
- D. Eastlake, J. Reagle, and D. Solo, "(Extensible Markup Language) XML—Signature Syntax and Processing" RFC 3275, Mar. 2002, http://www.ietf.org/rfc/rfc3275.txt
- ISO/IEC 9594-8 | ITU-T Rec. X.509 (2000) The Directory: Public-key and attribute certificate frameworks
- W3C Recommendations "XML Schema Part 1: Structures" "XML Schema Part 2: Datatypes," May 2001; http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema2/

Development issues and tools

- "XML Security Suite (XSS); www.alphaworks.ibm.com/tech/xmlsecuritysuite
- dumpasn1; www.cs.auckland.ac.nz/~pgut001
- Xerces Java Parser; http://xml.apache.org/xerces-j/• OSS ASN.1 Tools for Java; www.oss.com
- SAX 2.0; http://www.megginson.com/SAX/index.html
- "Fast Web Services," P. Sandoz and colleagues, Aug 2003, http://java.sun.com/developer/technicalArticles/WebServices/fastWS/
- K. Ewushi-Mensah and Z. Przasnyski, "Factors Contributing to the Abandonment of Information Systems Development Projects," *J. Information Technology*, vol. 9, 1994, pp. 185-201.

Electronic health-care systems

- D.W.Chadwick, D. Mundy, and J. New, "Experiences of Using a PKI to Access a Hospital Information System by High Street Opticians," *Computer Comm.*, Oct. 2003, pp. 1893-1903.
- D.P. Mundy and D.W. Chadwick, "A System for Secure Electronic Prescription Handling", *Proc. 2nd Int'l Conf. Management Healthcare and Medical Technology*, Illinois Inst. of Technology, 2002, pp. XX-XX .
- "Electronic Transmission of Prescriptions (ETP)," Dept. of Health, May 2002; www.doh.gov.uk/pharmacy/etp.htm.
- J. Larmouth, "Technical Advantages of Using ASN.1. for Telemedicine/E-Health," ITU-T Workshop on Standardization in E-Health, May 2003; www.itu.int/itudoc/itu-t/workshop/e-health/s4-02.html.

Acknowledgments

We thank Entrust Inc. for making its PKI security software available to the University of Salford.

This work was funded by the Engineering and Physical Sciences Research Council (EPSRC) under grant GR/M83483.

Darren Mundy *is a lecturer at the Centre for Internet Computing on the Scarborough campus of the University of Hull. He is also an editor of the security section of IEEE Distributed Systems online. Contact him at d.mundy@hull.ac.uk.*

David W Chadwick *is the leader of the Information Systems Security Research Group (ISSRG) at the University of Salford, British Standards Institute (BSI) representative to X.509 standardization meetings, and the international editor of X.518(93). He regularly attends IETF meetings and has written four Internet drafts about the use of PKIs and LDAP. Contact him at d.w.chadwick@salford.ac.uk.*