The IEEE International Conference on
Industrial Informatics (INDIN 2008)
DCC, Daejeon, Korea July 13-16, 2008

Evaluation of OPC UA Secure Communication in Web Browser Applications

Annerose Braune, Stefan Hennig, Sebastian Hegler

***Abstract*—The *OPC UA XML Web Services Mapping* offers a web service interface to access process data. Web services use XML technology for data exchange. Present-day web browsers include XML functionality already as a standard feature, they are therefore very promising candidates for the implementation of monitoring and operating functions for industrial processes. However, the acceptance of web services in industrial automation depends on adequate security realizations. For this purpose, the *Web Services Security Stack* provides several specifications to meet the requirements for secure message exchange. The *OPC UA XML Web Services Mapping* refers to these specifications.**

The application of web browsers for monitoring and operating of technical processes using OPC UA web services demand the computation of cryptographic algorithms within the scripting engine of the web browser. However, available scripting languages are not designed to compute complex mathematical, i.e., cryptographic, algorithms. Therefore, at the Institute of Automation of the Technische Universität Dresden the suitability of a native web browser for monitoring and operating of industrial processes with OPC UA based secure communication was analyzed. The paper shows representative measured computing times of cryptographic algorithms in *JavaScript*. The security specification *XML Signature* – which is mandatory for *OPC UA Web Services Mapping* – requires about 700ms to create a signature. Finally, the paper discusses methods to improve the performance.

I. Introduction

WEB services are increasingly accepted also in automation. Web service communication partners may interact using XML based messages and the internet protocol suite. First implementations of web services in automation devices are already available [1], [2]. The application of web services in automation includes XML based exchange of process data. This offers new opportunities also for visualization and control of automation systems. Current browsers already include XML functionality as a standard feature. They are therefore promising candidates as runtime environments for visualization applications.

However, the benefits of web services, e.g. distributed and dynamic architectures, with the possibility of establishing communication relationships between multiple participants, create potential security gaps. Therefore, web services require security measures to guarantee security in connection with *accessibility*, *confidentiality*, *non-repudiation*, and *integrity* of mission critical information at XML level: *application layer* security is required. If web browsers are used for monitoring and operating of industrial processes, security algorithms have to be integrated into a web browser as applications at layer 7 of the *ISO/OSI reference model* [3]. In this case, a standard web browser must be able to execute sophisticated mathematical security algorithms in real time.

At the *Institute of Automation* at *Technische Universität Dresden*, the user interface language *XVCML (eXtensible Visualization Components Markup Language)* for web based monitoring and control of technical processes with XML based data exchange is developed. First web based visualization and control applications, e.g. of a process plant model, were implemented with different web services such as *OPC XML-DA*. Preparing for the communication with *OPC UA Web Services* [4], the web browser has to be able to execute *XML Signature* algorithms. For web browsers without any additional plugins, security algorithms have to be implemented with the scripting language *JavaScript*. However, because of the performance of *JavaScript* a real time capability of such solutions is not guaranteed. Therefore, at the *Institute of Automation* appropriate algorithms were implemented and the performance was tested. This paper presents first results.

This paper first is introduced with an overview of the security objectives of industrial automation. The *Web Services Security Stack* is introduced, and the technologies realizing *security objectives* are explained. Then, the implementation of *XML Signature* algorithms in *JavaScript* and their computation times are presented. These will be compared against the computation times of the implementation of *WS Secure Conversation Language*, also implemented in *JavaScript*. Concluding, the results will be evaluated. Finally, a summary rounds up this paper.

II. Web Services and Information Security

The *NAMUR Worksheet NA 115* [5] and the *VDI/VDE guideline 2182* [6] define security objectives for industrial automation. Both documents state that the specified security objectives are in principle identical to those of classical information technology *(IT)* systems. Referring to classical IT systems, *ISO/IEC 13335-1* [7] defines *information security* as "all aspects related to defining, achieving, and maintaining" the following generic security objectives.

Integrity ensures the accuracy and completeness of assets.

Confidentiality guarantees that information is not made available or disclosed to unauthorized individuals, entities, or processes.

Manuscript received March 19, 2008. This work was supported by a research project at Institute of Automation, Technische Universität Dresden.

Annerose Braune is associate professor the Institute of Automation, Technische Universität Dresden, 01062 Dresden, Germany, annerose.braune@tu-dresden.de.

Stefan Hennig is doctorate student at the Institute of Automation, Technische Universität Dresden, 01062 Dresden, Germany, stefan.hennig@tu-dresden.de.

Sebastian Hegler is student at the Department of Computer Science, Technische Universität Dresden, 01062 Dresden, Germany, sh824836@inf.tu-dresden.de.

978-1-4244-2171-8/08/$25.00 ©2008 IEEE

Availability makes sure that resources or services are accessible and usable on demand by an authorized individual, entity, or process.

Authenticity ensures that the identity of an individual, entity, or process is the claimed one.

Non-repudiation is the ability to prove that an action or an event has taken place.

Accountability ensures that the action of an individual, entity, or process can be linked to it.

Reliability requires the consistency of intended behavior and results.

The security objectives *availability* and *reliability* have to be provided by the underlying infrastructure. For example, a redundant system can achieve and maintain availability. Information transfer does not have direct influence on these two objectives, so they are not subject to security considerations concerning the messages interchanged between a web service and a web browser.

The message format of web services is defined on the XML based message exchange paradigm SOAP [8]. The SOAP message specification does not demand that messages have to be transferred merely from the initial sender to the ultimate receiver, but rather additional processing nodes can be integrated into the message path. All involved nodes can modify the SOAP messages. To comply with the security objectives *non-repudiation* and *accountability*, all modifications have to be linked to the corresponding node. In terms of the security objective of *integrity*, a node may not modify any data, unnoticed by another node. At last, *confidentiality* requires that information may not be disclosed to any unauthorized nodes. Therefore, the implementation of the security objectives at the message level – the application layer of the *ISO/OSI reference model* – is necessary to meet the requirements of these security objectives.

XML Security, with the two standards *XML Signature* [9] and *XML Encryption* [10], is the key concept to meet the required generic security objectives at the application layer. Both standards describe an XML based notation to express digital signatures or encrypted data within XML documents. The *Web Services Security* specification [11] defines the approach to integrate *XML Security* into SOAP messages. This forms the basis for higher level standards like *Web Services Secure Conversation Language* [12] (*WSSC*), which provides a mechanism to preserve a security context over multiple message exchanges. Figure 1 shows the entire XML web services stack. Also, it shows *WSSC* residing on top of the *Web Services Trust Language* [13] (WS-Trust). *WS-Trust* defines the procedure of creating and exchanging security tokens, which are required for *WSSC*. The *Web Services Addressing* specification [14] "provides transport-neutral mechanisms to address Web services and messages" [14]. Therefore, *WS-Trust* and *WS-Adressing* are required for *WSSC*. The security extension *Secure Sockets Layer* [15] (*SSL*) of the underlying transport protocol HTTP is out of the scope of security considerations concerning the messages interchanged between a web service and a web browser, because SSL is classified as belonging to the transport layer of the *ISO/OSI reference model*.

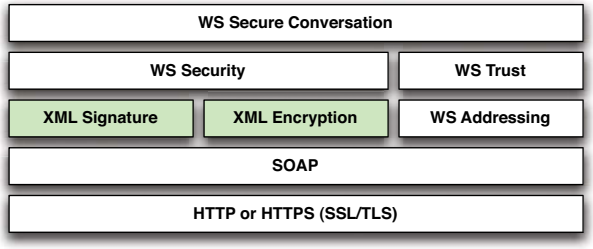

Fig. 1. The XML web services stack [4]

XML Encryption complies with the generic security objective *confidentiality*. However, *XML Encryption* only meets the requirement to conceal information from unauthorized individuals, entities, or processes. The security objective *confidentiality*, though, also implies *authorization*. This means that services or resources may not be used by unauthorized individuals, entities, or processes. *Access Control Lists*[1] (ACL) or the XML based access control language *XACML*[2] [16] meet this requirement.

XML Signature meets the requirements of the objectives *integrity*, *non-repudiation*, and *authenticity*. Based on *authenticity*, *XML Signature* also implicitly meets the objectives *accountability* and *confidentiality*.

According to [5], the security objectives *availability* and *integrity* are the most important ones. Depending on a concrete use case, e.g. for a transfer of mission critical process data, also the security objective *confidentiality* can have the same importance. Therefore, the new standard *OPC UA* requires at least *XML Signature* as mandatory for the *XML Web Services Mapping* [4] of the specified communication services. Concerning the realization of the security aspects, the *XML Web Services Mapping* of *OPC UA* only refers to the web services security standards mentioned above.

Based on the aforementioned considerations, the suitability of digital signatures for web-based process visualization will be evaluated.

III. XML Signature

XML Signature [9] is a system to encode digital signatures in an XML format. It uses the same established standards and techniques of cryptography as usual digital signatures. The basis of digital signatures is *asymmetric cryptography*. In contrast to symmetric cryptography, an entity creates a *key pair*. One of these keys, the *public key*, can safely be published. With it, data can be encrypted, for only the private key to decipher. The *private key* can also be used to generate digital signatures, which can be verified using the public key.

The required steps to sign a message and to verify the signature of this message by the communication partner using digital signatures are as follows:

[1]On most operating systems, ACLs define which user can access which files or services.

[2]The *eXtensible Access Control Markup Language* provides an XML dialect to declare authorization policies and a processing model to evaluate the intended access against the policies.

1) Calculate the hash code[3] of the data to sign.
2) Encrypt the hash code using the private key. This ensures the security objective of *non-repudiation*, because only the signer knows the private key.
3) Attach the hash code and the encrypted hash code to the message before transmitting it. Transmit the message.
4) The receiver of the message (and also of the hash code as well as the encrypted hash code) first decrypts the encrypted hash code with the public key of the communication partner. If the decrypted hash code and the plain hash code of the message are identical, then the communication partner is the expected one *(authenticity)*. Otherwise, the message has to be discarded or rejected, because of a potential attack.
5) The receiver calculates the hash code of the same data and compares the two hash codes – the one received with the message and the one calculated. If the hash codes are identical, then the *integrity* of the message has been proven. Otherwise, the data was modified at the communication channel: the message has to be discarded.

Summarizing, digital signatures meet the security objectives *integrity*, *authenticity*, *non-repudiation*, and *accountability*. Based on *authenticity*, digital signatures also comply with the requirement *confidentiality*.

XML Signature implements the aforementioned procedure. It defines the XML element *Signature*, shown in listing 1.

Listing 1. The structure of XML Signature [9]

```
<Signature ID?>
   <SignedInfo>
      <CanonicalizationMethod/>
      <SignatureMethod/>
      (<Reference URI? >
         (<Transforms>)?
         <DigestMethod>
         <DigestValue>
      </Reference>)+
   </SignedInfo>
   <SignatureValue>
   (<KeyInfo>)?
   (<Object ID?>)*
</Signature>
```

The element *SignedInfo* represents the data to be signed. Before the actual signature can be calculated, a *canonicalization algorithm* has to be applied. This algorithm orders attributes of elements alphabetically, and transforms indentation and whitespace of the original XML document into a defined format. Only in this way digital signatures can be applied in the first place. The *SignatureMethod* element defines the cryptographic algorithm used to sign the data. The *Reference* element uses a *URI* to refer to the actual data to be signed. It includes the hash code of the signed data, the *DigestValue*, as well as the algorithm calculating it, the *DigestMethod*. The entire element *SignedInfo* will be encrypted with the private key. The resulting cipher text is then written into the element *SignatureValue*.

The *WS Security* [11] specification defines how to integrate the *Signature* element of *XML Signature* into a SOAP message header. However, *WS Security* does not limit the number of *Signature* elements within a SOAP message [18]. Therefore, it provides methods to sign different parts of a SOAP message with different cryptographic keys, even using different cryptographic algorithms on each part. Consequently, every modification of a SOAP message can be indisputably linked to the modifying individual, entity, or process. This fact makes *XML Signature* the basis for meeting the requirements of a secure SOAP message based communication.

The procedure mentioned above was adapted for native web browsers, without any plugins. For this purpose, the algorithms performing *XML Signature* have to be implemented in *JavaScript*. This was done using an Open Source *JavaScript* library [19]. In a test environment this solution interacted with a web service based on *Apache Axis 2* [20]. The web browser is the initiating party – he has to create a request, which has to be signed before transmitting it to the server (steps 1 through 3 in the enumeration listed above). When receiving the response of the server, the web browser has to verify the signature (steps 4 and 5) before processing the integrated data.

Since the *SignedInfo* element usually has the same length, in this case about 500 B, the processing time to encrypt this XML based information was expected to be constant. The time to calculate the hash sum, however, depends on the length of the message. Due to these expectations, in a first step the computation time to hash the data depending on the data length was measured. Thereto, an open-source implementation [21] of the SHA-1 algorithm was used. Table I shows the results of this measurement for some representative data lengths.

TABLE I
TIMES TO HASH DATA USING SHA-1

Data length [byte]	time [ms]
32	1
64	2
128	5
256	9
512	15
1,024	29
2,048	68
4,096	157
8,192	307
16,384	571
32,768	1222

The SHA-1 algorithm produces a hash sum of 160 bits length, SHA-256 produces 256 bits, and SHA-512 512 bits. Compared to the size of the entire *SignedInfo* element (500 B), the length of these hash codes is negligible. Table II shows the time needed to encrypt the *SignedInfo* element.

TABLE II
TIMES TO CREATE A SIGNATURE USING XML SIGNATURE

Operation	time [ms]
Encrypt *SignedInfo* element with private key	729

[3]A hash function or algorithm (e.g. *Secure Hashing Algorithm 1*, SHA-1 [17]) generates a character string of constant length – the *hash code*, or *message digest* – from an input of arbitrary length. These functions are specially designed so that two different inputs will only very rarely generate the same output. This property is called *collision resistance*.

For a typical communication relationship – create one request message and process the resulting response – secured by *XML Signature*, the web browser first has to create the hash code of the data to be signed. The required computation times depending on the data length are shown in table I. Thereafter, the web browser has to integrate the hash code into the *SignedInfo* element and encrypt this element using the private key. The required time of this processing step is shown in table II. Then the web browser transmits the message, and has to wait for the response of the server. After receiving the response the web browser first decrypts the signature – which was encrypted by the server – using the public key of the server. The required computing time is comparable with the one shown in table II, due to the application of the same methods. By now, the web browser verified the *authenticity* of the originator of the message – if the decrypted hash code is equal to the one received as plain text. To verify the *integrity* of the received data, the web browser has to hash the according data itself. The required times depending on the data length to be hashed are shown in table I. Then the two hashes – the one received by the message and the one self created – have to be compared. If they are identical, then the integrity of the signed data has been proven.

The required roundtrip times of a typical communication relationship secured by *XML Signature* will be clarified through the following equations. The acceptance of the resulting reaction time depends on the rigidity of the time constraints.

$$time_{roundtrip} = time_{sign} + time_{server} + time_{verify} \quad (1)$$

$$time_{sign} = \overbrace{1\ ms \ldots 1\ s}^{hash\ data} + \overbrace{729\ ms}^{encrypt\ hash} \quad (2)$$

$$time_{verify} = \overbrace{729\ ms}^{decrypt\ hash} + \overbrace{1\ ms \ldots 1\ s}^{hash\ data} \quad (3)$$

$$time_{roundtrip} = 1,460\ ms \ldots 3,458\ ms + time_{server} \quad (4)$$

Summarizing, the realization of *XML Signature* with *JavaScript* has been proven to work, although the processing times of cryptographic algorithms are very time consuming. This is due to the fact that most *JavaScript* runtime environments in web browsers are implemented as scripting engines, and they are therefore not as fast as binaries of compiled languages[4]. So, the achievement of real time capabilities depends on the data length and on the requirements of the monitored process.

Consequently, approaches to reduce the time needed to create or to verify a digital signature in SOAP messages have to be determined. On top of the web services security stack (see Figure 1) the *Web Services Secure Conversation Language* standard [12] promises to span a security context over multiple message exchanges. This approach will be analyzed in the following section.

IV. Web Services Secure Conversation

The *Web Services Secure Conversation Language* [12] (WSSC) offers a comparable level of security and performance to web service based applications on application layer as SSL/TLS [15], [22] offers to HTTP [23] on transport layer [18].

For this purpose, *WSSC*defines methods to establish and preserve a security context – a common state space – between communication participants. All messages exchanged within this security context belong to a particular *conversation*. *WSSC* also allows to create and to use more efficient keys [12]. The application of more efficient keys does not necessarily decrease the level of security. *WSSC* recommends to restrict the validity of a key only to one message, whereby signature and encryption employ a different key. The required steps to create a security context and exchange messages within this context using *XML Signature* are listed below, starting with the participant initiating the conversation.

1) Create a *security token*. Send this token securely to all intended participants using common *asymmetric cryptography*. The procedure of creating and exchanging security tokens is defined by the *Web Services Trust Language* [13] (WS-Trust). This is a one-time operation for each conversation.
2) Before sending a message, derive a key from this token. If encryption is demanded, two keys are required, one key to sign the message, and one key to encrypt it using *XML Encryption*. For this purpose, use the *Keyed-Hash Message Authentication Code*[5] [24] (HMAC) algorithm, which is based on a hash algorithm instantiated by a secret key, e.g. a random number:

$$key_{derived} = hmac(token, number_{random}) \quad (5)$$

3) Create the hash code of the data to sign and put it into the *SignedInfo* element specified by *XML Signature* (see section III).
4) Encrypt the *SignedInfo* element using the derived key. For this purpose, *WSSC* also uses the HMAC function – now with the derived key as the secret:

$$signature = hmac(signedInfo, key_{derived}) \quad (6)$$

5) Attach the *Signature* element specified by *XML Signature*, and the random number used to derive the key, to the SOAP message before transmitting it. Transmit the SOAP message.
6) The receiver of the message first derives the keys using the previously exchanged security token and the received random number.
7) With this key, the receiver can verify the signature.
8) Repeat from step 2, for each message to be exchanged within this security context.

The processing time to sign a SOAP message using the methods defined by *WSSC* was expected to decrease significantly, due to the usage of the HMAC algorithm to encrypt the *SignedInfo* element. *WSSC* resides on top of the web services security stack (see Figure 1) – above *XML Signature*. Consequently, the *SignedInfo* element of *XML Signature*, which has to be encrypted using the HMAC algorithm, is the same

[4] *JavaScript* was originally intended to check whether web forms are completely filled in, or to show marquees on a web page.

[5] This function is also part of the TLS standard.

as the one used in the test case of section III. This element is about 500 B long. The HMAC algorithm is based on a hash algorithm. Therefore, the computing time to encrypt the *SignedInfo* element was expected to be similar to the one of the corresponding data length shown in table I – in this special case approximately 15 ms.

We also implemented *WSSC*in *JavaScript*. The test environment also consists of *Apache Axis 2* at the server side. The measured times to create a signature of an XML document using the methods of *WSSC* are shown in table III.

TABLE III
TIMES TO CREATE A SIGNATURE USING *WSSC*

Operation	time [ms]
Derive one key	16
Encrypt *SignedInfo* element with the derived key	13
Overall signature creation time	**29**

Table III only shows the time to derive a key for the encryption of the *SignedInfo* element, and the time encrypting it. The time to calculate the hash code of the data to be signed has to be added for a real application. This time is the same as shown in table I. The following equations summarize the times required for each processing step using *WSSC* to sign a message.

$$time_{roundtrip} = time_{sign} + time_{server} + time_{verify} \quad (7)$$

$$time_{sign} = \overbrace{1\ ms \ldots 1\ s}^{hash\ data} + \overbrace{16\ ms}^{derive\ key} + \overbrace{13\ ms}^{encrypt\ hash} \quad (8)$$

$$time_{verify} = \overbrace{16\ ms}^{derive\ key} + \overbrace{13\ ms}^{decrypt\ hash} + \overbrace{1\ ms \ldots 1\ s}^{hash\ data} \quad (9)$$

$$time_{roundtrip} = 60\ ms \ldots 2,058\ ms + time_{server} \quad (10)$$

Compared with the time shown in table II – and also with the times listed in the equations (1) through (4) – *WSSC* significantly decreases the time needed to encrypt data. According to the equations (4) and (10) it reduces the required time to perform a typically communication relationship approximately at about $1,400\ ms$ in this concrete test environment. Consequently, for web based process visualization, even when requiring very short response times for critical scenarios, *WSSC*can be an alternative to pure *XML Signature*.

However, using *WSSC*, all communication participants share the same secret. Due to this fact, *WSSC* cannot fulfill the security objective of *non-repudiation*. Based on *non-repudiation*, *WSSC* also cannot meet the requirement of the security objective *accountability*.

V. CONCLUSION

The computation of cryptographic algorithms within the scripting engine of web browsers is possible, in general. This has been proven to work. However, the implementation of these algorithms in *JavaScript* is not very fast, because *JavaScript* is an interpreted programming language.

WSSC provides methods to preserve a security context for multiple messages to exchange. In contrast to pure *XML Signature*, the computionally intensive algorithms of cryptography do not have to be applied to each message – they are only needed to exchange the security token: exactly once, for each conversation. However, the processing times of signatures depend on the data length to be signed. The dependent part only consists of the procedure of hashing the data to sign. These computation times are equal for pure *XML Signature* and for signatures created by means of *WSSC*. In the concrete test environment, the times to compute the hash code of the data to be signed were about $1\ ms \ldots 1\ s$ depending on the data length. Therefore, depending on a concrete use case, it has to be evaluated which data has to be signed. For this purpose, *OPC UA* provides a method to reduce the actual data to transfer. It is possible to observe process variables in terms of changes in value by means of *subscriptions*: only if the value of the observed process variable has changed, a message will be transmitted. *OPC UA* also defines additional methods to reduce the data to transfer. Furthermore, *XML Signature* enables the possibility to sign arbitrary parts of a message to be exchanged. Based on this fact, the information which really needs to be signed can be minimized. Nevertheless, in the mentioned test environment, *WSSC* reduces the computing time for a typically communication relationship using *XML Signature* approximately at about $1,400\ ms$.

Encrypting the information before transmitting it can also be required, depending on a concrete use case (see section II). The computation time to encrypt a message also depends on the data length. However, this dependency is more significant than for *XML Signature*. *XML Encryption* applies cryptography algorithms to the entire data to be encrypted, whereas *XML Signature* only encrypts the hash code. To gain performance, *XML Encryption* uses *symmetric cryptography* to encrypt the data. Then, the *symmetric key* – a shared secret – will be encrypted by an *asymmetric cryptography* algorithm. Both the encrypted data and the encrypted symmetric key have to be transferred to the recipient with each message. *WS Secure Conversation* also promises to speed up *XML Encryption*, because the shared secret only has to be exchanged once – by use of pure *XML Encryption* with each message. The actual order of magnitude of how *WSSC* can reduce computation time for encryption has to be analyzed in further work.

However, *WSSC* cannot comply with the requirements of the security objective *non-repudiation*. Due to this fact, it has to be evaluated whether to minimize computation times, or to meet the requirements of all security objectives. This has to be decided for each use case. Summarizing, the realization of the security objectives at application layer can be suitable for web browser based process visualization, depending on a concrete use case.

VI. SUMMARY

Based on an analysis of the required security objectives of industrial automation, this paper considers how to achieve these security objectives by means of *XML Security* at the *OSI* application layer. With this consideration, the *Web Services Security* and also standards above it (see Figure 1) have been discussed, because the *OPC UA XML Web Services Mapping* only refers to these specifications to meet the requirements of the security objectives. According to that, these technologies are very interesting for web browser based process

visualization, and thus they have to be implemented with web browser specific software technologies. Therefore, the suitability of these technologies for the intended scope has been analyzed. For this purpose, in a first step, *XML Signature* – defined as mandatory in the *OPC UA specification* – has been implemented in *JavaScript*. In a second step, the methods of *WSSC* – which is on top of the web services security stack, above *XML Security* – has also been implemented. The computation times to create digital signatures using *XML Signature* and the methods of *WSSC* have been measured and compared.

The processing times to create digital signatures using *XML Signature* are about 700 ms. This time only comprises the computation time to encrypt the previously calculated hash code with the private key. *WSSC* provides methods to reduce the computation times of *XML Signature* and also *XML Encryption*. Using *WSSC*, the processing time realizing *XML Signature* has been reduced to about 30 ms.

Summarizing, *WSSC* can significantly reduce the required computation times of *XML Signature*, and also of *XML Encryption*. However, only *XML Signature* has been analyzed in this paper. Due to the usage of a shared secret, *WSSC* cannot comply with the requirement of the security objective *non-repudiation*. For that reason, based on a threat analysis, it has to be individually decided for each use case whether to minimize processing times or to meet all requirements of the security objectives.

To further improve performance of cryptographical functions, using the operating systems' crypto APIs seems possible, too. However, these interfaces have to be made available in the browser's *JavaScript* environment. Further research should be dedicated to this approach.

ACKNOWLEDGMENT

The authors would like to thank *Stephan Stiebitz* for the implementation of the test environment.

REFERENCES

[1] Softing AG, "OPC Toolbox C++ XML DA. Produktinformation Softing AG. 09/2003, 3rd ed." *http://www.softing.com/home/de/pdf/ia/product-info/opc/D_IA_10D_0309_OPC_Toolbox_Z.pdf*, Last checked: March, 7th 2008.

[2] Schneider Electric, "Quantum Unity Ethernet Network Modules Documentation, 3rd ed," *Schneider Electric*, 09/2005.

[3] ISO/IEC, "Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model (ISO/IEC 7498-1:1994 (E), second edition 1996)," *ISO/IEC International Standard*, 1996.

[4] OPC Foundation, "OPC Unified Architecture - Part 6: Mappings, Release Candidate 0.93," *OPC Industry Standard Specification*, 2006.

[5] NAMUR-Arbeitskreis AK 2.8 "Internet/Intranet", "NA 115 - IT-Security for Industrial Automation Systems: Constraints for measures applied in process industries," *NAMUR-Worksheet*, 2006.

[6] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, Fachausschuss Security, "IT-security for industrial automation - General model," *VDI/VDE-Richtlinien*, 2007.

[7] Normenausschuss Informationstechnik (NI) im DIN, "Information technology - Security techniques - Management of information and communications technology security - Part 1: Concepts and models for information and communications technology security management (ISO/IEC 13335-1:2004)," *Deutsches Institut für Normung e.V.*, 2006.

[8] N. Mitra and Y. Lafon, "SOAP version 1.2 part 0: Primer (second edition)," W3C, Tech. Rep., Apr. 2007, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

[9] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, "XML-signature syntax and processing," W3C, W3C Recommendation, Feb. 2002, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/.

[10] T. Imamura, B. Dillaway, and E. Simon, "XML encryption syntax and processing," W3C, W3C Recommendation, Dec. 2002, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/.

[11] Organization for the Advancement of Structured Information Standards, OASIS, "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)," *OASIS Standard Specification, http://oasis-open.org/committees/tc_home.php?wg_abbrev=wss*, 2006.

[12] S. Anderson, J. Bohren, T. Boubez, M. Chanliau, G. Della-Libera, B. Dixon, P. Garg, M. Gudgin, S. Hada, P. Hallam-Baker, M. Hondo, C. Kaler, H. Lockhart, R. Martherus, H. Maruyama, A. Nadalin, N. Nagaratnam, A. Nash, R. Philpott, D. Platt, H. Prafullchandra, M. Sahu, J. Shewchuk, D. Simon, D. Srinivas, E. Waingold, D. Waite, D. Walter, and R. Zolfonoon, "Web Services Secure Conversation Language (WS-SecureConversation," *OASIS Standard Specification, http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512*, 2007.

[13] S. Anderson, J. Bohren, T. Boubez, M. Chanliau, G. Della-Libera, B. Dixon, P. Garg, M. Gudgin, P. Hallam-Baker, M. Hondo, C. Kaler, H. Lockhart, R. Martherus, H. Maruyama, A. Nadalin, N. Nagaratnam, A. Nash, R. Philpott, D. Platt, H. Prafullchandra, M. Sahu, J. Shewchuk, D. Simon, D. Srinivas, E. Waingold, D. Waite, D. Walter, and R. Zolfonoon, "Web Services Trust Language (WS-Trust)," *OASIS Standard Specification, http://docs.oasis-open.org/ws-sx/ws-trust/200512*, 2007.

[14] T. Rogers, M. Hadley, and M. Gudgin, "Web services addressing 1.0 - core," W3C, W3C Recommendation, May 2006, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509.

[15] A. O. Freier, P. Karlton, and P. C. Kocher, "The SSL Protocol - Version 3.0," *INTERNET-DRAFT, http://wp.netscape.com/eng/ssl3/draft302.txt*, 1996.

[16] OASIS, "eXtensible Access Control Markup Language (XACML) Version 2.0," *OASIS Standard Specification, http://oasis-open.org/committees/tc_home.php?wg_abbrev=xacml*, 2005.

[17] D. Eastlake 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC 3174 (Informational), Sep. 2001, updated by RFC 4634. [Online]. Available: http://www.ietf.org/rfc/rfc3174.txt

[18] D. Wang, T. Beyer, T. Frotscher, and M. Teufel, *Java Web Services mit Apache Axis*. Software & Support Verlag GmbH, Frankfurt, 2004.

[19] T. Wu, "BigIntegers and RSA in JavaScript," *http://www-cs-students.stanford.edu/tjw/jsbn/*, Last checked: March, 7th 2008.

[20] The Apache Software Foundation, "Apache `<Web Services />` Project," *http://ws.apache.org/*, Last checked: March, 7th 2008.

[21] P. Johnston, G. Holt, A. Kepert, Ydnar, and Lostinet, "JavaScript MD5," *http://pajhome.org.uk/crypt/md5/index.html*, Last checked: March, 8th 2008.

[22] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," RFC 4346 (Proposed Standard), Apr. 2006, updated by RFCs 4366, 4680, 4681. [Online]. Available: http://www.ietf.org/rfc/rfc4346.txt

[23] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Jun. 1999, updated by RFC 2817. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[24] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104 (Informational), Feb. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2104.txt