

## Secured RESTful Sensor Web Enablement Services for Wireless Sensor Networks

Bouhouchi R\*, Yengui S and Ezzedine T

Department of Computer Science, National Engineering School of Tunis, Tunisia

### Abstract

The security and interoperability of an adopted and advanced architecture within heterogeneous components, based on the Open Geospatial Consortium (OGC) Sensor Web Enablement Architecture (SWE) and RESTful web service, requires integrity and confidentiality in the different communication protocol.

RESTful services are considered a versatile lightweight solution relied upon by a number of advanced web services. At the same time, RESTful services suffer from a lack of meta-data description concerning security requirements. We introduce the REST security protocol to provide secure data transfer service, together with its quality and its performance analysis when compared to equivalent WS-security configuration.

The work in this paper aims to introduce a security protocol of communication between the sensors based on SWE services and the adopted RESTful interface. In this way, we provide a REST security protocol which will implement a secure lightweight sensor message and analyze its performance comparing to equivalent web services.

**Keywords:** Representational state transfer; Performance; Protocol; Sensor web; Wireless sensor networks; Sensor web enablement; SOS

### Introduction

There is important on-going progress in the field of sensor networks deployment, especially with regards to controlling and monitoring the environment through the measurement of environmental physical values. Pollution, climate, global warming and natural disasters are of global significance, directly impacting human well-being. The criticality of the consequences requires that the communication tools be secure, providing strong confidence in the data transfer protocol. REST is suitable lightweight for such application; thus securing these web services whilst respecting the SWE standards is the proposal we present in this paper.

Confidentiality, availability and integrity are the security features that we will apply on web services in order to secure. Finally, section 5 we present our conclusions and future work.

### Related Work

Diverse research and studies related to RESTful security has been conducted to provide security methods for data exchange between sensors and applications.

The security solution of Amazon S3 [1] REST security model supports authentication and the client encryption data over HTTP requests. Requests are based on a token method to protect the data from unauthorized access, deletion or modification. Transmitting the proof of identity and ensuring the request authenticity it is the role of the token, which brings the signature value calculated in every request. The security of the data transfer depends upon the integrity of the end-points. The paper is organized as follows: Section 2 introduces the most important works that deal with the REST security and SWE standards. In section 3, we start by introducing the architecture system based on SWE standards and REST technology presenting our security approach for an adopted SWE services to a REST architecture style. In section 4, we analyze and evaluate the performance of the proposed security approach and position our security orientation regarding WS security APIs via a benchmark. Rouached [2] research showed that RESTful services are much lighter than SOS services.

Our approach is to secure our sensors communication channel

using the lightweight RESTful interfaces based on SWE services. We propose to apply a specific security policy on the sensors data exchange using the lightweight JSON format based on OGC standards.

The same solution by SAP Labs France [3] but it brings more flexibility and server benefits from a PKI environment in order to serve its clients by rendering services without the need to maintain and generate secret keys. Users can use the REST security protocol with any service providers by simply uploading of their public key.

The security solution of Nevada Solar Energy-Water-Environment [4] explains the authentications implied for RESTful web service such as: HTTP Digest Authentication, HTTP Basic Auth, Access Token and OAuth, and API Key. This security solution uses the data collected from various sensors and stores it within the database.

All these previous studies and others article present valuable security models and approach for REST security and its performance. Some of them provide excellent results with regards to securing communication channels between sensors and application servers; however they do not provide an interoperable and secured solution which respects standards. Sergio [5] research provided a variety of interfaces by sustaining interoperability of SWE and proposed the use of RESTful services based on SWE standards.

### REST Security for SWE

In this section, we will demonstrate the principle concept of secured data transfer based on REST security principle and SWE standards.

**\*Corresponding author:** Bouhouchi R, Department of Computer Science, National Engineering School of Tunis, Tunisia, Tel: (216) 71 874 700; E-mail: [riadh.bouhouchi@yahoo.com](mailto:riadh.bouhouchi@yahoo.com)

**Received** November 03, 2015; **Accepted** January 29, 2016; **Published** January 31, 2016

**Citation:** Bouhouchi R, Yengui S, Ezzedine T (2016) Secured RESTful Sensor Web Enablement Services for Wireless Sensor Networks. J Telecommun Syst Manage 5: 126. doi:10.4172/2167-0919.1000126

**Copyright:** © 2016 Bouhouchi R, et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

## SWE framework and secure RESTful web services

**SWE framework:** The SWE Framework is an idea from the Open Geospatial Consortium (OGC) for a protocol that describes Sensors and Sensor Observations, designed to unify communications between sensors using a particular set of tools or a suite of standards encodings. Those standards define the appropriate data format for sensor data and metadata, and web services interfaces.

The work in this level aims at developing the interoperability and improving the security of data provided by sensor networks based on SWE standards.

SWE offers a specific language and service interface in order to guarantee a smooth and standardized transfer between sensors and data storage. This 'core' is divided in two parts:

- The "Service Model": This standard defines 4 interfaces of sensor related web service types
- The "Information Model": contains the data model primarily for the encoding of the sensor observations and metadata results.

### Part one: Information model

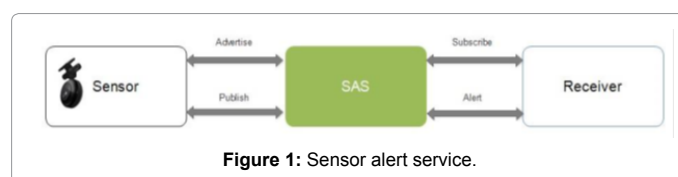
- **Sensor observations service (SOS):** The standard, which defines a web services interface; providing not only querying observations but also sensor metadata. Furthermore, this norm allows other operations such as; registering new sensors and remove existing ones, and defines new methods to insert new sensor observations.
- **Sensor planning service (SPS):** The standard that defines interfaces for queries which provide information about the abilities of a sensor and how to task it.

This Standard is designed to support queries that have the following purposes:

- To determine the viability of a sensor planning request (SPR)
- To submit and commit a request
- To ask about the status of the demand
- To update or remove a request
- To request details and information about additional OGC Web services to provide access to data collected by the requested task
- **Sensor Alert Service (SAS):** To determinate how alert or "alarm" conditions are defined and detected. The "SAS" norm is used to focus on alerts from sensors and sensor webs, so the SAS itself acts like a registry rather than an event notification system.
- **Web Notification Services (WNS):** This standard defines a set of specifications which show the web service interactions with the notifications. A web service can communicate and exchange information with other web services without needing prior knowledge of these other Web Services.

### Part two: information model

- **Observations & Measurements (O&M):** The standard which specifies an XML implementation for encoding observations from a sensor and for features and behaviors involved in sampling whilst taking those observations (Figure 1).
- **Sensor Model Language (SensorML):** The standard which not only provides an xml schema for defining the geometric, dynamic and observational characteristics of a sensor, but also describes sensors systems and the processes associated with the sensors observations.



**Figure 1:** Sensor alert service.

- **Transducer Model Language (TransducerML or TML):** Refers to the conceptual model and XML Schema for exchanging live streaming or archived data from any sensor systems.

OGC standards facilitate the adaptation of external tools, forms and model to Restful interface which has been well introduced in previous research; however, how can we guarantee the confidentiality, the integrity and the availability of the sensor data transfer once this implementation?

## Security policy for an adapted SWE framework to REST architecture

Our approach here is to adopt an advanced security policy within heterogeneous components: adopt the Sensor Web Enablement services with secure RESTful web service as shown in Figure 2. This architecture is already defined and based on two characteristics regarding the development of REST interfaces for each element of the service model (SOS, SAS, SPS, and WNS): Each service can be applied as an application server that encapsulates SWE service instances, and each operates as a proxy for the service to offer RESTful interface to the data [6-8].

Each deployed sensor node can join the network by its unique ID in order to provide, send and receive data using the different Service Model Standards components (SOS, SAS, SPS, and WNS) and the different operations of REST (POST, GET, Delete, Put, etc.) (Figure 2).

In our case, the most important level is the SOS service and the RESTful interface (RESTful SOS) which guarantees interoperability and acts as a proxy to the existing SOS. This proxy also transforms encoded observations in the Observations and Measurements format to lightweight JSON format, which is considered as an independent platform for a data exchange and requires lower overhead and less secured resources compared to the XML format shown in Figure 3.

Proposing secure communication by respecting the philosophy of RESTful services and taking full advantage of the reusing the

HTTP protocol with the minimum overhead:

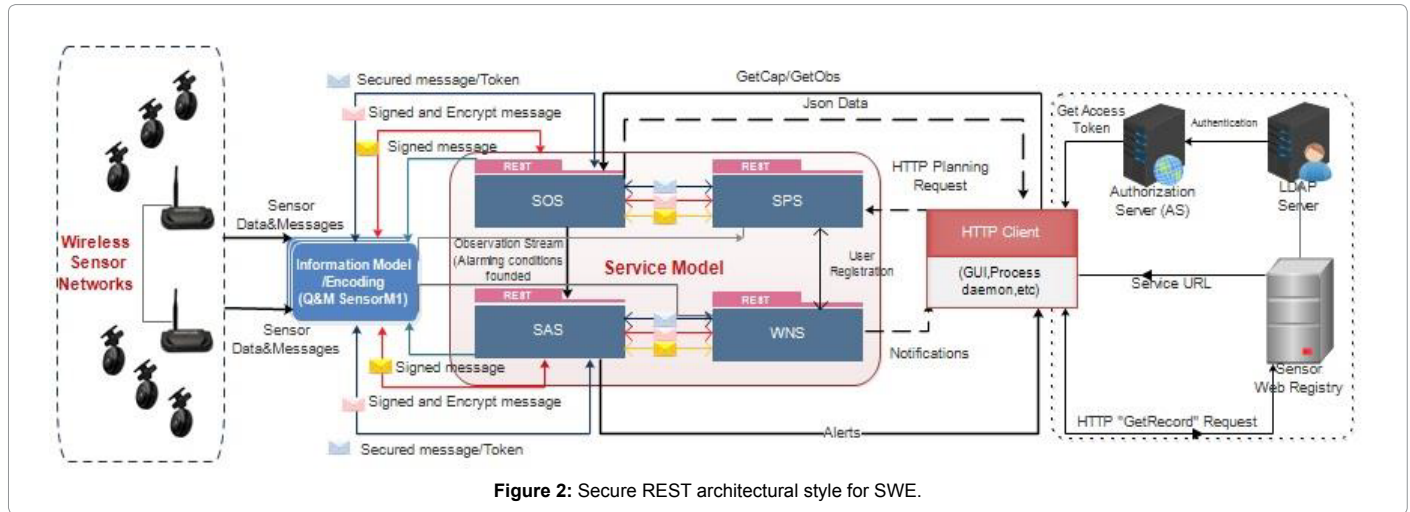
REST is ideally suited to exposing data over such networks and has low bandwidth.

This advantage will be more efficient when we guarantee a secured communication between the sensors based on SWE services and those on the adopted RESTful interface (Figure 3).

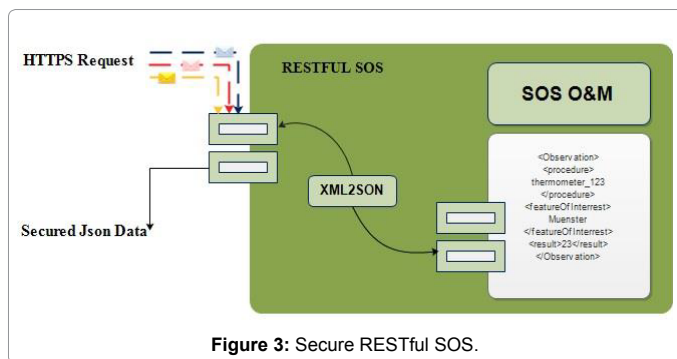
In our case, reuse HTTP protocol to its full advantage will be the pillar of our RESTful security principal protocol, which will be very aligned to the WS-security standard (confidentiality, Authenticity, and non-repudiation).

In this session, we show the steps to attach signature information to a sensor message, the encryption of a REST message and the basic authentication methods for REST.

**Message signature:** The use of digital signatures for transmitting



**Figure 2:** Secure REST architectural style for SWE.



**Figure 3:** Secure RESTful SOS.

sensors data through non-secure channels can be very valuable in combating forgery and preventing the misrepresentation of digital information [9,10].

In our case, a digital signature is a form of electronic signature, which assures that receiving server that the sensor message is the same message as intended by the sensor source.

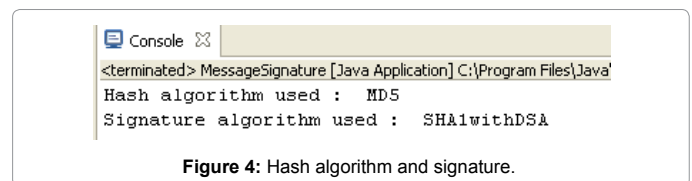
In this case, a digital signature authenticates sensor messages and guarantees the correct transmission of electronic data.

The advantages of a digital signature process are better overall performance of authentication, integrity, and non-repudiation. The principal of our implementation is to ensure secure communication at the message level. The execution of the secured signature REST program needs some requirements: Msg is a message, sig is a signature algorithm name, dig is a digest algorithm, cid is a Certificate Id, pk is the sender private key, url path the requested path and hds are headers element to protect. So, we started to declare variables that we will use in our implementation.

```

/***** Variables declaration*****/
static boolean v;
static String theURL ;
static MessageSignature request;
static byte[] dv;
static byte[] valeursSignature;
static String message ;
static Cipher cipher;
    
```

In our program, we allow client to decide which algorithm to use



**Figure 4:** Hash algorithm and signature.

```

MessageDigest dig =
MessageDigest.getInstance("MD5");
Signature sig=
Signature.getInstance("SHA1withDSA");
    
```

- MD5 (Message Digest 5) is a cryptographic hash function that computes, from a given message its hash.

- SHA1 with DSA creates and verifies the digital signature of a message (Figure 4).

The java code presents steps to attach signature information to the message after “digest then encrypt” processing.

**Hash and Sign technical: Message => Hash => Sign => Verification**

**SignatureAlgorithm = (Hash Function ID, Cipher ID)**

**Java code 1 : Signature of Rest messages**

```

dv
=calculerValeurDeHachage(getMessage(),di
g);
url="";
if( getMessage().equals(request))
{
theURL= url.getPath();
}
Byte[] bytes =
concat(dv,url,sig,dig,cid,hds);
digValue =
calculerValeurDeHachage(bytes,dig);
sigValue = encrypt(digValue,sig,pk);
    
```

We should start by generating method for public and private keys: Generate keys from a security of parameter produces a pair (private key :pk, public key : ppk).

#### Java function : Generate private and public keys

```
KeyPairkeyPair = generateKeyPair(key);
PrivateKeypk = keyPair.getPrivate();
PublicKeyppk =keyPair.getPublic();
publicstaticKeyPairgenerateKeyPair(longl
) throws Exception {
KeyPairGeneratorkeyGenerator =
KeyPairGenerator.getInstance("DSA");
SecureRandomrng =
SecureRandom.getInstance("SHA1PRNG",
"SUN"); rng.setSeed(l);
keyGenerator.initialize(1024, rng);
return (keyGenerator.generateKeyPair());
}
```

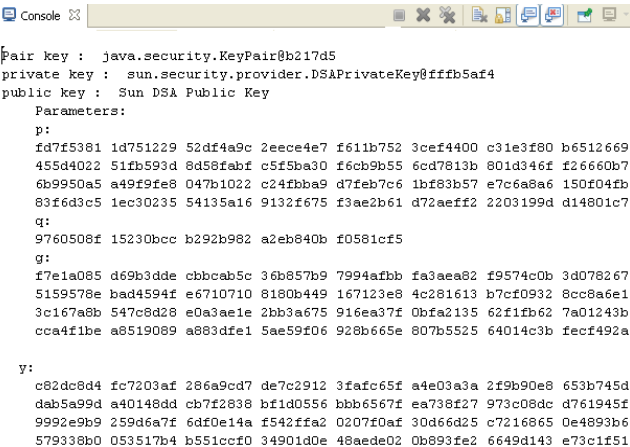
Public key was used to encrypt the message. The private key is held by the receiver only, which is used to decrypt the message encrypted with the public key (Figure 5).

We retrieve digValue from a function, which return bytes from a message send via sensor and a message digest (Figure 6).

#### Java function : Calculate hash values


```
publicstaticbyte[]
calculerValeurDeHachage(MessageDigestdig
,Stringmsg) {
dig.update(msg.getBytes());
returndig.digest();
}
```

Then we calculate the values of the signature using the adequate method (Figure 7).



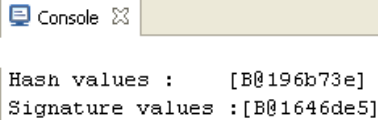
```
Pair key : java.security.KeyPair@b217d5
private key : sun.security.provider.DSAPrivateKey@fffb5af4
public key : Sun DSA Public Key
Parameters:
p:
fd7f5381 1d751229 52df4a9c 2eece4e7 f611b752 3cef4400 c31e3f80 b6512669
455d4022 51fb593d 8d58fabf c5f5ba30 f6cb9b55 6cd7813b 801d346f f26660b7
6b9950a5 a49f9fe8 047b1022 c24fbba9 d7feb7c6 1bf83b57 e7c6a8a6 150f04fb
83fd3c5 1ec30235 54135a16 9132f675 f3ae2b61 d72aef2 2203199d d14801c7
q:
9760508f 15230bcc b292b982 a2eb840b f0581cf5
g:
f7e1a085 d69b3dde cbbcab5c 36b857b9 7994afbb fa3aea82 f9574c0b 3d078267
5159578e bad4594f e6710710 8180b449 167123e8 4c281613 b7cf0932 8cc8a6e1
3c167a8b 547c8d28 e0a3ae1e 2bb3a675 916ea37f 0bfa2135 62f1fb62 7a01243b
cca4f1be a8519089 a883dfe1 5ae59f06 928b665e 807b5525 64014c3b fecf492a
Y:
c82dc8d4 fc7203af 286a9cd7 de7c2912 3fafc65f a4e03a3a 2f9b90e8 653b745d
dab5a99d a40148dd cb7f2838 bf1d0556 bbb6567f ea738f27 973c08dc d761945f
9992e9b9 259d6a7f 6df0e14a f542ffa2 0207f0af 30d66d25 c7216865 0e4893b6
579338b0 053517b4 b551ccf0 34901d0e 48aede02 0b893fe2 6649d143 e73c1f51
```

Figure 5: Public and private method for keys generator.



```
Hash values : [B@192d7d0]
```

Figure 6: Hash values.



```
Hash values : [B@196b73e]
Signature values : [B@1646de5]
```

Figure 7: Define the hash and signature values.

#### Calculate the values of signing

```
publicstaticbyte[]
signValue(Signature sig, byte[]
data, PrivateKeyclé,Stringmsg)
throws Exception {
sig.initSign(clé);
sig.update(data);
return (sig.sign());
}
```

The second java code presents the signature verification function. To verify if the signature is valid, we reverse executed the previous digest and encrypted code.

We calculated digest values then we retrieved the digest values calculated by the sender of a mobile message.

#### Java code 2 : Verification of REST signature

```
publicstaticbooleanverifySig(byte[]
data, PublicKeykey, byte[] sig) throws
Exception {
if (data.equals(sig)== true) {
returntrue;
}
returnfalse;
}
```

**Encryption:** A goal is to protect data within messages sent from sensors via RESTful web services to data storage, based on the WSE standards.

Encryption is used to protect sensitive data within the sensor message. An algorithm and a cryptographic key are used to encrypted data, whilst later the cipher text is converted back to the original plaintext [11-15].

Sensor as the originator of a message and the application server that receives the message from the sensor.

The process of data confidentiality can be applied in two steps:

- Encrypting the data. In this step, the sender (sensor) converts plaintext to cipher text.
- Decrypting the data. In this step, cipher text rendered intelligible to the intended recipient (application server) by converting it back to plaintext.

To provide data confidentiality, asymmetric algorithm is preferable, it imposes heaviness but on large quantities of data, it guarantees encryption performance. In addition, we generate a symmetric small key, easy for encryption and will be sent with the sensor message to the receiver.

This message contains the encrypted payload and the key details regarding the encryption algorithm.

This code processes the payload of a message. To share information



between public and private keys, the message contains an encrypted symmetric key.

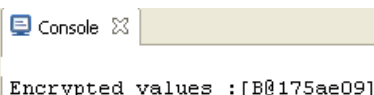
#### Encryption of a REST message

```
Public static byte[] encrypt(byte[] m,
PrivateKeyKey){
try
{
KeyGeneratorkeyGenerator =
KeyGenerator.getInstance("Blowfish");
keyGenerator.init(n);
SecretKeyblowfishKey =
keyGenerator.generateKey();
Cipher cipher;
cipher =
Cipher.getInstance("RSA/ECB/PKCS1Padding");
byte[] plainData = m;
// symmetric encryption of data and signature
cipher = Cipher.getInstance("Blowfish");
cipher.init(Cipher.ENCRYPT_MODE, blowfishKey);
cipher.doFinal(plainData);
return (cipher.doFinal(plainData));
}
catch (Exception e)
{
return null;
}
}
```

This code presents the reverse operation with respect to the above code. The message contains information about encrypted parts and codes used for key encryption and date encryption. To decrypt the data, the receiver of a message retrieves the symmetric key (Figure 8).

#### Decryption of a REST message

```
public static byte[] decrypt(byte[] data,
PrivateKey myPrivateKey) {
try {
Cipher cipher;
// decrypt secretKey with my private
key
cipher =
Cipher.getInstance("RSA/ECB/PKCS1Padding");
cipher.init(Cipher.DECRYPT_MODE,
myPrivateKey);
byte[] plainData =
cipher.doFinal(data);
return (cipher.doFinal(plainData));
}
catch (Exception e)
{
return null;
}
}
```



```
Console
Encrypted values : [B@175ae09]
```

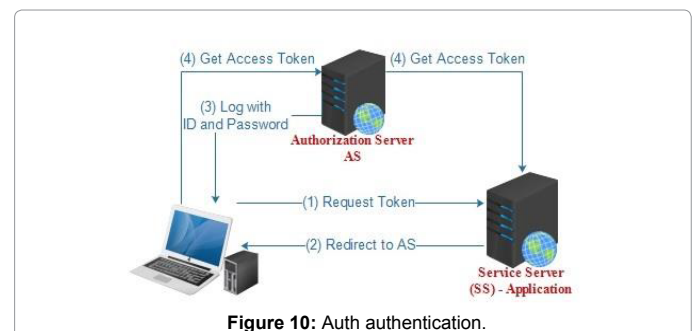
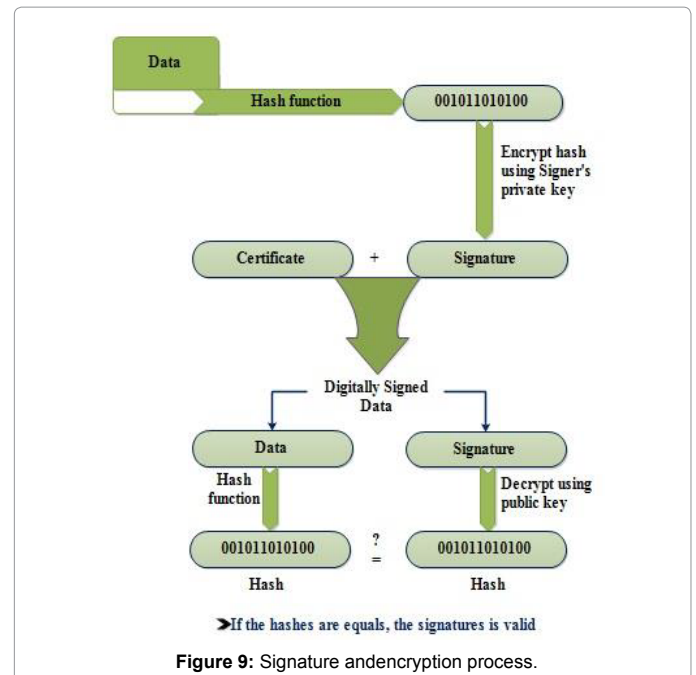
**Figure 8:** Encrypted payload during a request.

**Signature and encryption:** To enhance security, applying both a digital signature and encryption will be an important feature. Creating a signature allows for authentication, avoids repudiation [16,17]. A signature alone cannot stop attackers from accessing the content of the message. Encryption alone is considered as an effective way of protecting confidential data but do not preclude against data manipulation and then data can be changed due to the importance the integrity and security of data, a combination of encryption and signature at the message level is applied to ensure confidentiality of data and prevent intruders from any modification (Figure 9).

**Basic authentication methods for REST:** Much research has been conducted about the use of HTTP to resolve the authentication problem by developing solutions and parameterizing servers in order to authenticate the client in every request, using HTTP basic Authentication, digest Authentication, Access Token, API Key and OAuth.

All these previous methods are used for web services authentication, however HTTP basic Authentication and HTTP digest Authentication are not stateless due to the lack of session that keep the session. For this reason, the OAuth method is considered as the best method as we use a Token instead of ID and Password.

The client starts by requesting a service; the service server (SS) redirects the clients to a specific browser. OAuth process is working as following (Figure 10):



1. Client request service to SS
2. The SS redirect the client's browser to the AS
3. The client login to the AS to get his Token
4. The SS get the Token from the AS
5. Client can access to the SS with the Token

We have to note that this protocol allows a flexible way for client to authenticate. Many approaches include Client's id and password as POST parameters by the use of Authentication Http Header. It must pass a grant type ("client credentials") if they are correct, the AS return a JSON Object that contain the access Token and it Type and optionally other values needed. OAuth prefers the Authorization HTTP Header as a mechanism to request an access Token.

## Experimental Results and Evaluation

The implementation of java codes for all these security scenarios requires a specific configuration and also a middleware system for preparing Sensor Web Infrastructures (SWI) based on Sensor Web Enablement (SWE). We have used a recognized free software: 52° North Sensor Web framework, which provides implementations for all SWE services through the OX-Framework (OGC framework).

The aim of OX-Framework is to offer a flexible architecture, which provides easy access to all types of OGC Web Services tools to visualize the required data. Thin SOS Client application, Web Map Server application, and uDig Plugin application are built in the OX-Framework in order to provide access to the different sensor data, through a web graphic user interface. In our case we have chosen the Thin SOS Client to interface with SOS service. To implement this demonstration we have used a several important public web applications as the RESTful SOS deployed at URL1 of the Institute for Geo-informatics of the University of Muenster and the application of the JSON exchange format presented in URL2.

The graphic interface provides easy accessibility to any kind of SOS and the response to the O&M request will be converted into a secured JSON format.

After configuring the different security scenarios and preparing the full test environment, we conduct performance tests in order to analyze the performance of our Restful service security scenario with other security scenarios as WS-security, measuring average response time (milliseconds), throughput (transaction/seconds), and response size (KB).

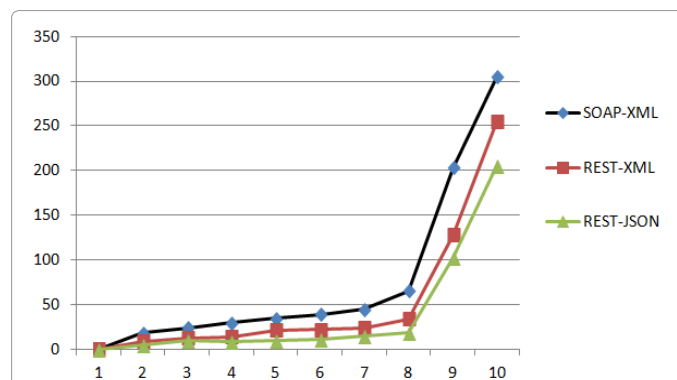
This benchmarking test scenario also requires a system for measuring and analyzing performance of the security solution that we have used; Apache JMeter is the software used in this operation.

In the following tables and figures, we present the processing of the data buffer size in terms of transmission time, using the different security mechanisms for REST and SOAP services (Figure 11).

The next Figures 12 and 13 shows the results of SOAP and REST without security.

SOAP XML \$ REST JSON-XML without security										
Response time in milliseconds	SOAP-XML	18,6	24	29,5	35	39	45	66	203	305
	REST-XML	8,6	12,4	13,8	21,3	22,1	23,9	34	128	254,7
	REST-JSON	4,5	9,8	8,9	10,6	11,5	14,5	19	103	205,5
Number of service requests		1	2	3	4	5	6	7	8	9

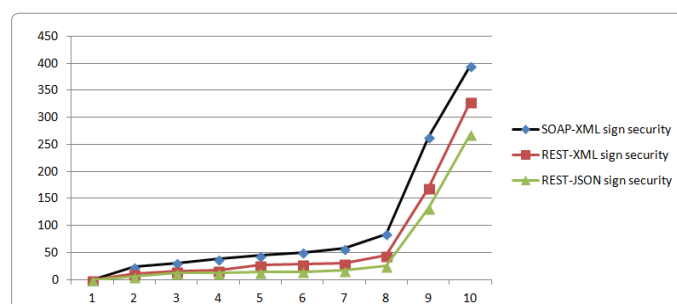
**Figure 11:** SOAP XML \$ REST JSON-XML without security.



**Figure 12:** Statics results of SOAP XML \$ REST JSON-XML without security.

SOAP XML \$ REST JSON-XML with sign security										
Response time in milliseconds	SOAP-XML sign security	24,18	31,2	38	46	51	59	85	264	396,5
	REST-XML sign security	11,1	16,1	18	28	29	31	44	170	330
	REST-JSON sign security	5,8	12,7	12	14	15	17	25	133	270
Number of service requests		1	2	3	4	5	6	7	8	9

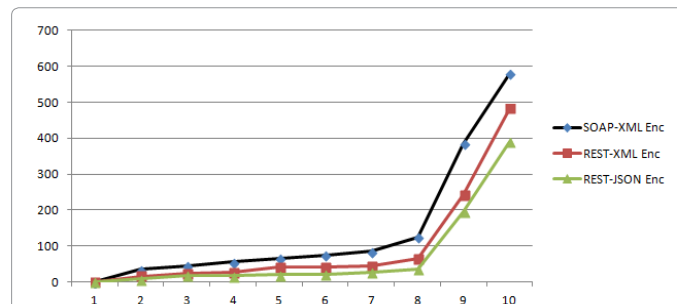
**Figure 13:** SOAP XML \$ REST JSON-XML with sign security.



**Figure 14:** Statics results of SOAP and REST with sign security.

SOAP XML \$ REST JSON-XML with encryption										
Response time in milliseconds	SOAP-XML Enc	35,3	45,6	56	66,5	74,1	85,5	124	385,7	579,5
	REST-XML Enc	16,3	23,5	26,2	40,4	42	45	65	242	483
	REST-JSON Enc	8,5	18,4	17	20	22	28	36	195	390
Number of service requests		1	2	3	4	5	6	7	8	9

**Figure 15:** SOAP XML \$ REST JSON-XML with encryption.



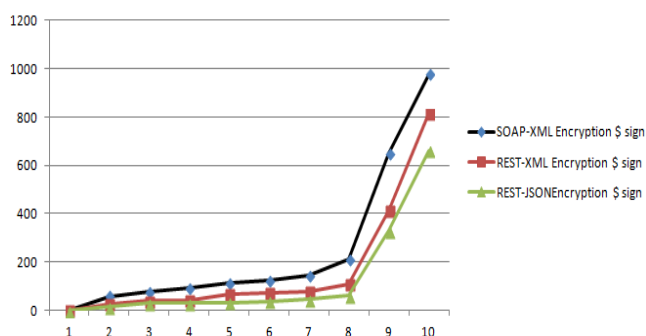
**Figure 16:** Statics results of SOAP XML \$ REST JSON-XML with encryption.

The following Figures 14 and 15 shows the results of SOAP and REST with signature security.

The following Figure 16 shows the results of SOAP and REST with encryption.

SOAP XML \$ REST JSON/XML Encryption \$ sign security										
Response time in milliseconds	SOAP/XML Encryption \$ sign	59,4	76,8	94,3	112	124,8	144	209	649,6	976
	REST/XML Encryption \$ sign	27,4	39,6	44,2	68,1	70,8	76	109	412	813
	REST-JSON Encryption \$ sign	14,3	31,1	29	34	37	45	61	328	660
Number of service requests		1	2	3	4	5	6	7	8	9

**Figure 17:** SOAP XML \$ REST JSON/XML encryption \$ sign security.



**Figure 18:** Statics results SOAP XML \$ REST JSON/XML encryption \$ sign security.

The following Figures 17 and 18 shows the average statistics results of SOAP and REST with encryption and signature

The difference between SOAP and REST and also the difference between REST-JSON and REST-XML in terms of average processing time is mentioned in the all figures; therefore we can conclude differences in performances regarding signature and encryption, depending on the processing data size.

REST-JSON security shows always better performances than REST-XML and SOAP.

URL1: <https://svn.52north.org/svn/swe/incubation/OXRestWS/trunk/OX-RestWS/>

URL2: <http://swe.unimuenster.de:8080/52nRESTfulSOS/RESTful/sos>

/AirBaseSOS/

## Conclusion and Future Work

In this research, we have presented a new approach to providing security for an adopted RESTful architecture model with OGC's SWE services. Secured exchanging of data respecting the REST philosophy and SWE standards is considered as an important extension to the SWE services.

We also examined the performance evaluation results and analyzed the impact of the secured messages on the performance of REST web services. The security approach presented demonstrated the efficiency of the secured JSON message in terms of communication time and size reduction. As future works, enhancing the encryption based on authentication tokens will be our priority.

## References

- Trilles S, Belmonte O, Diaz L, Huerta J (2014) Mobile Access to Sensor Networks by Using GIS Standards and RESTful Services. Ieee Sensors Journal pp: 287-294.
- Richardson L, Ruby S (2007) RESTful Web Services.
- Web Services Security (2004) SOAP Message Security 1.0 (WS-Security 2004) OASIS Standard 200401.
- Crockford D (2006) The application/json Media Type for JavaScript Object Notation.
- Choi T, Gouda MG (2011) HTTP/I: An HTTP with Integrity. Department of Computer Science, the University of Texas at Austin, IEEE.
- Gambarotto P (2009) Technologies pour Web Services faciles: REST, JSON. INPT DSI, ENSEEIHT Departement Informatique.
- Kumar PP (2012) Comparing Performance of Web latest/dev/RESTAPI.html. Service Interaction Styles: SOAP vs. REST. Proceedings of the Conference on Information Systems Applied Research New Orleans Louisiana, USA.
- Choi T, Gouda MG (2009) HTTP Integrity: A Lite and Secure Web against World Wide Woes. Department of Computer Science, the University of Texas at Austin. Tech Rep TR pp: 09-41.
- World Wide Web consortium (W3C) (2003) Simple Object Access Protocol (SOAP).
- Choudhary P, Rajendra A, Roberts N (2012) Http Based Web Service Security over Soap Dogan Yazar, Demo Abstract: Augmenting Reality with IP-based Sensor Networks.
- Rouached M (2012) RESTful Sensor Web Enablement Services for Wireless Sensor Networks, IEEE. Browse Conference Publications, Saudi Arabia.
- Bogdanovic-Dinic S, Veljkovic N, Stoimenov N (2010) Ginissense-applying ogc sensor web enablement. Earth Science. 13<sup>th</sup> AGILE International Conference on Geographi c Information Scienc, Guimarães, Portugal.
- Forsberg D (2014) RESTful Security: Nokia Research Center Helsinki, Finland Sungchul Lee, Ju-Yeon Jo, Yoohwan Kim: Environmental Sensor Monitoring With Secure RESTful Web Service Ijsc.
- Shneidman J, Pietzuch P, Ledlie J, Roussopoulos M, Seltzer M, et al. (2004) Hourglass: An infrastructure for connecting sensor networks and applications. In Harvard Technical Report TR-21-04.
- Yazar D, Dunkels A (2009) Efficient application integration in IP-based sensor networks. In Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, Builds, New York, USA.
- Serme G, de Oliveira AS, Massiera J, Roudiery Y (2012) Enabling Message Security for RESTful Services SAP Labs, France.
- Amazon (2012) Amazon Simple Storage Service REST Security Model.