International Journal of Advanced Computer Technology (IJACT)
ISSN:2319-7900

A PRACTICAL APPROACH TO EXPOSE THE PUBLIC KEY INFRASTRUCTURE FEATURES THROUGH WEBSERVICES

Ebot Ebot Enaw, Djoursoubo Pagou Prosper
University of Yaounde I, Cameroon
National Advanced School of Engineering
Department of Computer Sciences

Abstract

The Public Key Infrastructure (PKI) provides services that permit users to communicate in a secure manner on an unsecure network by means of digital certificates and cryptography primitives. However, in order to secure an application through cryptography and PKI, cryptographic primitives need to be implemented in the programming language used to develop the application. This raises scalability and interoperability issues as the diversity of programming languages and development environments requires all cryptography primitives to be implemented in each of the existing programming languages.

Therefore, this paper presents a practical approach to secure an application with cryptographic primitives developed in different languages through service oriented architecture technology.

Keywords: web service, PKI,cryptography,SOAP.

1 Introduction

In recent years, in order to curb cyber-attacks that are perpetrated against software, cryptography and Public Key Infrastructure (PKI) have become very popular in software development. Public Key Infrastructure through digital certificates and cryptographic algorithms provides confidentiality, integrity, authenticity and non-repudiation which are essential properties for data and transactions security. Although PKI provides so many useful features, its integration in software source code poses a major issue since for every application that needs to be secured by PKI, the cryptographic primitives used to secure the application needs to be implemented in the same programming language as that of the application in question. Given the complexity and dynamics of cryptographic algorithms, and the ever growing change in programming languages and environments, it might be difficult and cumbersome to implement and update these algorithms in the numerous programming languages available.

New concepts have been developed over the years to improve software engineering through interoperability and software component re-usage that were major concerns for developers. Among these concepts, Service Oriented Architecture (SOA) is one of the most prominent as it provides interoperability between applications and software components re-usage regardless of their programming language and environment.

The aim of this paper is to present an approach which enables the reuse of cryptographic primitives developed in Java to secure softwares developed in other programming languages using SOA and XML.

2 Related work

Some research have been done on topics related to this issue namely [1] that conducts a comparative study of Identity-based Public Cryptography (ID-PKC) and traditional Public Key Infrastructure based on several aspects including technical, architecture and policy. Their study revealed that the two systems are very similar however there are some areas in which they differ namely, the binding of the public and private keys: while in the traditional PKI, it is achieved through the use of a certificate, in an ID-PKC mechanism, the binding between the private key and the data is managed by a Trusted Authority (TA) at the point of request, while the binding between the public key and the data can be done by anyone at any point. Therefore the ID-PKC best suits environments where a centralized security policy is checked regularly and there is a strong binding between a user and the identifier of the communication end point such as mobile networks whereas PKI is more adapted to environments where the policy should be controlled locally at the level of the client.

[2] first surveyed approaches and concepts related to service oriented architecture and then in an effort to cater for the combination of the SOA paradigm with event-driven processing, proposed an extension to SOA called xSOA that incorporates a service composition layer to offer necessary roles and functionality for the consolidation of multiple services into a single composite service. In addition, xSOA provides a separate tier for service management that can be used to monitor the correctness and overall functionality of aggregated/orchestrated services, supporting complex aggregate (cross-component) management use cases, such as service-level agreement enforcement and dynamic resource provisioning.

[3] in an effort to establish a baseline of performance data that can be used to explore performance/security tradeoffs in environments with complex attributes, such as resource or

ISSN:2319-7900

bandwidth limitations, presents the results of a series of experiments aimed at analyzing the performance impact of adding WS-Security to SOAP-based web services.
[4] presents the concepts and technologies that underpin web services along with their vulnerabilities and then presents WS-Security concepts as well as algorithms for SOAP message encryption.

Though [1] presents PKI architecture and [2], [3] and [4] deal with web services and its security through PKI, none of these research papers provide a methodology for providing interoperability and reusability of cryptographic primitives.

Therefore, this paper is intended to help individuals and institutions that can't afford a cryptographic toolkits for each available programming language to reuse PKI primitives developed in Java using web services to secure applications developed in other programming language.

3 Research problem

Given the surge in cybercrimes and the ubiquity of ICT, it is indispensable for software developers to use the Public Key Infrastructure to secure their applications. However securing applications through PKI requires the integration of PKI toolkit or API which contain the cryptographic primitives into that application. This raises a particular issue since for every application to secure the developer needs to find the toolkit developed in the same programming language as the application he wants to secure. In order to solve this issue we propose an approach based on Service Oriented Architecture (SOA) which uses toolkit/API developed in one programming language to secure applications regardless of the programming language used to develop them. The SOA and PKI concepts are described in section 4 and 5 below.

4 SOA

Service Oriented Architecture (SOA) is an architecture that has been designed to deliver interoperability and flexibility in software design. Companies use applications developed in different programming technologies that need to interact in an effort to accomplish business goal. Moreover, in order to adapt to the ever changing business environment, software design should be flexible so as to ensure cost effective modifications. With SOA, application business logic is divided into services that are loosely coupled and interact properly regardless of the programming language used to implement them.

SOA as depicted in the figure below relies on some concepts such as:

- WSDL (Web Services Description Language): it is the standard derived from XML used to describe the interface of web services ;
- UDDI (Universal Description Definition and Integration): it is the standard used to register and look up services. In fact, services are published in a directory where applications can invoke them through UDDI.
- XML schema (XSD): Given that SOA services are designed to operate in heterogeneous environments, XML has been defined as the language used to specify messages exchanged between services.
- SOAP: It is the protocol used at the transport layer to exchange messages between the service consumer and the service provider.

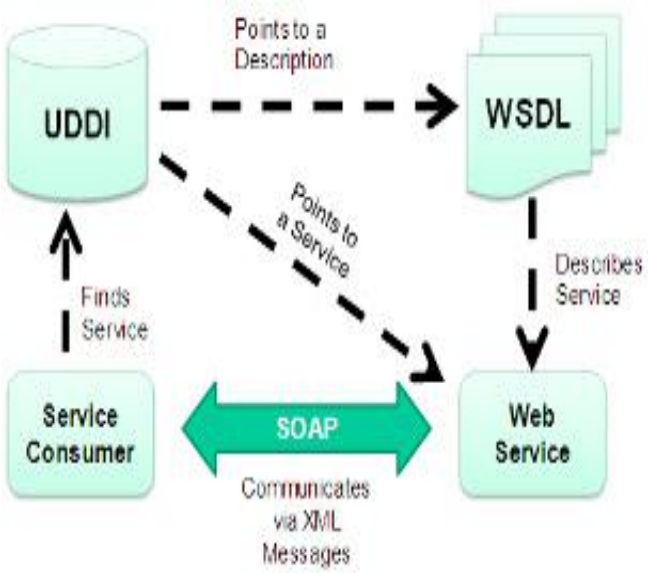

Figure 1: SOA architecture

5 PKI System

In recent years, Public key infrastructures have become the starting point for modern security mechanisms on the Internet, PKI is closely linked to the asymmetric key encryption, digital signatures and encryption services [5]. PKI is a combination of hardware, software, facilities, people, policies, and processes that are leveraged to create, manage, store, distribute, and revoke digital certificates that lie at the heart of a trusted identity system or Certification Authority (CA). PKI has been recognized as an important security component in digital infrastructures to support authentication, integrity, confidentiality, and non-repudiation [9].

5.1. PKI services

Cryptosystem which is the software or hardware implementation of the cryptographic process provides the following services [6, 7]:

- **Confidentiality** - keeping the private nature of a message is achieved through encryption; the message recipient public key is obtained from his/her certificate in order to establish an encrypted or ciphered message that can only be decrypted by the message recipient private key.

- **Integrity** – ensuring message or information has not been changed accidentally or deliberately with the help of a digital signature.

ISSN:2319-7900

- **Authenticity** - confirming the identity of an individual, a corporation, an application or device using an information system.
- **Non-repudiation** – Proving that a specific user is an author of a message or information with the help of digital signature.

5.2. PKI components

PKI integrates digital certificates, public key cryptography and Certification Authorities into an organization-wide network security architecture [8]. A typical organization PKI encompasses the issuance of digital certificates to individual users and servers; end-user enrollment software; integration with certificate directories; tools for managing, renewing and revoking certificates and related services and support. The main components of the PKI landscape are:

- **End Entity.** End-users, corporations, devices (e.g servers), or any other entity that can be identified in the subject field of the digital certificate.
- **Digital Certificate.** Electronic credential, consisting of public key, which is used to sign and encrypt data. Digital Certificates provide the foundation of a PKI.
- **Certification Authority (CA):** Trusted entity that issues and revokes digital certificates.
- **Registration Authority (RA):** An entity accredited by CA to validate requests for issuing digital certificates and identifies end entities.
- **Certificate Policy and Practice Statements:** Documents that outline how the CA and its certificate are to be used, the degree of trust, legal liabilities if the trust is broken, and other related issues.
- **Certificate Repository:** A directory of services or other locations where digital certificates are stored and published.
- **Certificate Revocation List:** List of certificates that have been revoked before reaching the scheduled expired date.

5.3. Traditional architecture

There are three basic types of architectures composed according to the level of trust of Certification Authorities (CA) on existing PKI [10, 11]:

- **Single CA (Root CA)** is a CA that issues certificates to users and systems, but not to other CAs. It is easy to build and maintain. All users trust it and have one certificate on the trust path.
- **Hierarchical PKI** is the traditional architecture, all users trust the same central root CA and all the CA have a single superior CA, except the root CA.
- **Mesh PKI** is the first alternative to a hierarchy, multiple CAs provide PKI services and the CAs are related through per-to-per relationships, each user trust a single CA. Certificates issued to CA in a mesh PKI are more complex than the ones usually found in a hierarchical PKI.

6 Our Solution

6.1. Methodology

In an effort to provide a solution for the re-usage of cryptographic primitives, we designed a new architecture based on the following methodology:

1. Select a reliable application programming interface (API) containing cryptographic primitives implementations: given the flexibility, the reliability and the robustness of Java platform especially its J2EE framework, we selected the Java Cryptography Architecture (JCA) which contains java implementations of popular cryptographic primitives as the foundation API of our architecture ;
2. Develop a middleware based on webservices that will permit applications to use cryptographic primitives contained in the API (JCA in our case) regardless of their programming language. To this end we used the JAX-WS framework that will be described in subsequent sections ;
3. Develop a module that will secure communication between the application and the middleware. To this end, we used the WS-Security standard that will be described in subsequent sections ;

6.2. Description of the system modules

With regard to the methodology presented in the previous section, our framework is made up of three (03) main components namely the JCA framework, the webservice wrapper and the WS-Security framework which are depicted in the figure below. These modules will be described in subsequent sections.

ISSN:2319-7900

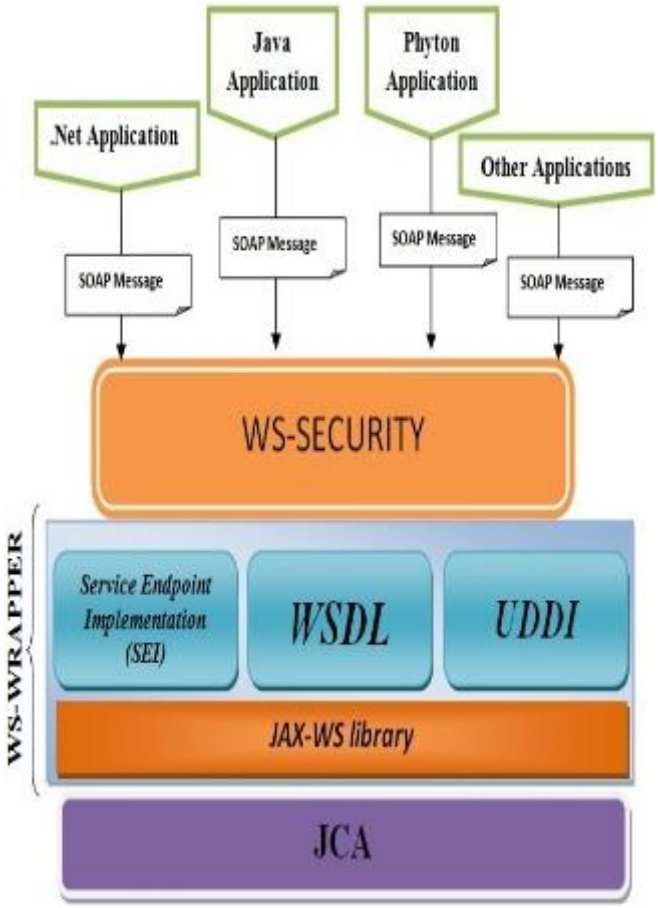

Figure 3: Architecture of our system

6.2.1 Java Cryptography Architecture

The Java Cryptography Architecture (JCA) is a set of modules and API (applications programming interface) provided by the Java framework and aimed at permitting programmers integrate security features into their Java applications easily.

It is built on two main principles [11] :

- Implementation independence and interoperability: The JCA is designed so that applications don't need to implement security algorithms rather they invoke security services they want to use from the Java platform. These security services are usually contained within components called providers. JCA enables implementation interoperability which means that different implementations can work with each other's keys, verify each other's signatures, etc.
- Algorithm independence and extensibility: Although security services are implemented in several providers in different manners, they do respect a common standard in their implementation. This permits other security algorithms to be implemented and added dynamically. However the Java platform includes a number of built-in providers that implement a basic set of security services that are widely used today among which the programmer is free to choose anyone depending on his needs.

To satisfy the two aforementioned principles, JCA relies on two main components:

- Cryptographic Service Provider (CSP) : It is a package or set of packages that implement cryptographic services, such as digital signature algorithms, message digest algorithms, and key conversion services. Thus depending on his needs, the programmer through its application simply requests a particular service (such as the DSA signature algorithm) and gets an implementation from one of the installed providers. However, the programmer can request an implementation from a specific provider. When faster or more secure versions of algorithms are released, providers can be updated transparently to the application accordingly. Additional providers may be added statically or dynamically. Clients may configure their runtime environment to specify the provider preference order which is the order in which providers are searched for requested services when no specific provider is requested. As stated earlier, although providers implement security services in different ways, they comply to a common standard called Service Provider Interface (SPI) which is an abstract class.
- Engine Class: An engine class provides the interface to a specific type of cryptographic service, independent of a particular cryptographic algorithm or provider. The type of services supported are cryptographic operations (encryption, digital signatures, message digests, etc.), generators or converters of cryptographic material (keys and algorithm parameters), or objects (keystores or certificates) that encapsulate the cryptographic data and can be used at higher layers of abstraction. Every invocation of engine class methods are routed to the provider's implementations through classes that implement the corresponding Service Provider Interface (SPI). That is, for each engine class, there is a corresponding abstract SPI class which defines the methods that each cryptographic service provider's algorithm must implement.

6.2.2 Webservice wrapper

This component is made up of software components that permit the exposure of cryptographic primitives contained in JCA framework as web service. To implement this component, we developed service endpoint implementation (SEI) based on the JAX-WS library that accesses the cryptographic primitives

ISSN:2319-7900

contained in the JCA framework. A service endpoint interface or service endpoint implementation (SEI) is a Java interface or class, respectively that declares the methods that a webservice client can invoke on the service [12]. It is worth mentioning that JAX-WS does not permit classes or methods that need to be exposed as webservices to be defined as static or final. Thus, given that several JCA engine classes or methods are static or final, to develop their corresponding SEI, we developed our own classes and methods and set them as public. The developed classes and methods were designed to invoke the genuine JCA classes and methods. The just developed SEI then gathers the input parameters, supplies them to the engine class in question, gets the result and delivers it as output. Once the Service Endpoint is developed, we use the Netbeans IDE to generate the WSDL. To sum up, this module takes as input the java code of a cryptographic primitive and delivers as output the corresponding webservice (WSDL and Service Endpoint Implementation). Although at this point an external application can fully interact with our webservice, this interaction takes place in an unsafe environment which as such jeopardizes the confidentiality, integritiy and non-repudiation of the transactions. In this light, we designed a layer called WS-Security that helps overcome this issue by providing a way to secure the communication between external applications and the webservice.

6.2.3 WS-Security

This component is aimed at securing communication between the application to be secured and the webservice. It is based upon WS-Security which is a specification that provides the following security services over web services:

- Authentication: WS-Security defines how different security tokens should be transferred within SOAP messages and how the receiver can extract and verify them to authenticate the sender.
- Integrity: WS-Security uses digital signatures, by employing XML Signature.
- Confidentiality: WS-Security provides specifications for XML document encryption to guarantee that even if the message is eavesdropped, it cannot be understood.

In order to implement WS-Security the developer needs to specify a security policy and then binds the said security policy to the WSDL.

6.3 Technical environment

To develop our solution we used:

- A server with the following characteristics: 16 GB RAM, 1To Hard disk, Intel Xeon 1.87Ghz *16
- **Netbeans 7.3**: Netbeans is a popular free IDE (Integrated Development Environment) that supports many languages like Java and PHP ;
- **Java** and **Tomcat 7.0.34.0**

7 Concrete Implementation of our approach

Within the framework of our article, we illustrate a concrete use of our methodology using the Signature engine class which is a JCA component in charge of digital signatures and we show how this raw JCA component is transformed into webservices using our methodology as follows:

7.1 First step of our methodology

Since we want to expose the digital signature features as webservices, according to our methodology we have to identify a module in the JCA framework that implements these features. In the JCA framework, the *Signature Engine Class* is the component that implements digital signature features. The class has two main methods *sign* for signing data (using the private key) and *verify* for verifying digital signatures (using the public key), nevertheless before these methods can be invoked, the *Signature Engine Class* should be initialized using the methods *initSign, initVerify* and *update*. Given that *initSign, initVerify* and *update* are used only for purposes internal to the *Signature Engine Class*, we will focus solely on *sign* and *verify* methods used to implement the data signature and signature verification processes.

7.2 Second step of our methodology

The second step consists of implementing the webservice which is made up of the *Service Endpoint Implementation* class of the *Signature Engine Class* identified previously and its corresponding WSDL.

To develop the *Service Endpoint Implementation* class we proceed as follows:

For the signature method, we first define the provider of the algorithm we want to use, then we create an instance of the *signature* class that we initialize with the private key, and finally we applied the signature algorithm with the private key to the data.

For the verification method *verif*, after recovering the provider, we initialize the *verif* method with the public key, then we applied the method to the data which outputs a Boolean “1” if the signature matches and “0” if not.

According to JAX-WS syntax all methods to be exposed as webservice should be annotated with *@WebMethod*, and classes should be annotated with *@WebService*.

The source code of the service endpoint implementation is provided below.

```
package WebService;
```

ISSN:2319-7900

```
import java.io.ByteArrayInputStream;
import java.nio.ByteBuffer;
import java.security.*;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.X509EncodedKeySpec;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
@WebService(serviceName = "SignWebService")
@Stateless()
public class SignWebService {
    Signature mysignature;
    @WebMethod(operationName = "Sign")
    public byte[] Signature( @WebParam(name = "myprovide") String provide, @WebParam(name = "myalgo") String algorithm, @WebParam(name = "myprivateKey") byte[] privatekeybytes,@WebParam(name = "mySecRan") SecureRandom random, @WebParam(name = "mydata") ByteBuffer data) throws NoSuchAlgorithmException, InvalidKeyException, SignatureException, NoSuchProviderException, InvalidKeySpecException
    {
        Provider provider = Security.getProvider(provide);
        mysignature = Signature.getInstance(provide);
        X509EncodedKeySpec privateKeySpec = new X509EncodedKeySpec(privatekeybytes);
        KeyFactory keyFactory = KeyFactory.getInstance(algorithm, provider);
        PrivateKey privatekey = keyFactory.generatePrivate(privateKeySpec);
        mysignature.initSign(privatekey,random);
        mysignature.update(data);

@SuppressWarnings("MismatchedReadAndWriteOfArray")
        byte[] signatureBytes = mysignature.sign();
        return signatureBytes;
        }
    //verification
    @WebMethod(operationName = "Verification")
    public Boolean Verif(@WebParam(name = "mypublicKey") byte[] publicKeyBytes, @WebParam(name = "mydata") byte[] signature, @WebParam(name = "datatoverif") ByteBuffer data ) throws InvalidKeyException, SignatureException, NoSuchAlgorithmException, NoSuchProviderException, InvalidKeySpecException, CertificateException
    {
        CertificateFactory certificateFactory = CertificateFactory.getInstance("X509");
        Certificate certificate = (Certificate) certificateFactory.generateCertificate(new ByteArrayInputStream(publicKeyBytes));
        PublicKey publicKey = certificate.getPublicKey();
        mysignature.initVerify(publicKey);
        mysignature.update(data);
        return mysignature.verify(signature);
    }
}
```

Regarding the WSDL of the Signature webservice, we used the Netbeans IDE to generate it automatically based on the Service Endpoint Implementation of the Signature class provided above.

7.3 Third step of our methodology

The third step of the methodology consists of adding a security layer over the webservices using the WS-Security framework. In this light, a security policy (designed to sign the header and encrypt the body of the SOAP messages) was developed and later linked to the WSDL generated in the last step.

8 Conclusion and future work

The ever growing importance of ICT/Internet in our daily life, coupled with the surge in cybercrimes, makes it mandatory to use PKI to secure applications. In fact, because the PKI provides integrity, confidentiality, authentication and non-repudiation it has become an essential part of transactions and application security. However, the integration of PKI features poses some issues namely the fact that to integrate PKI features

ISSN:2319-7900

into an application, the programmer needs to find and master an API that contains PKI primitives developed in the same programming language as that used for the application in question. This results in situations where applications developed in some programming languages where updated and stable PKI primitives are not implemented cannot be secured easily using PKI because of the unavailability of an implementation of primitives in that programming language unless the programmer develops all the primitives which can be tedious, long and risky given the complexity of cryptographic algorithms. In order to overcome this issue, we propose in this paper an approach whereby the cryptographic primitives will be exposed as webservice so as to permit their interactions with any application regardless of the programming language used to develop them. We choose JAVA to implement our webservice because of the robustness, reliability, flexibility and popularity of its environment and frameworks namely JCA. The approach used, consists of leveraging JCA cryptographic primitives to webservice using the JAX-WS framework. The interaction between applications to be secured and the webservice was later secured using the WS-Security standard. Future work can include developing webservice client for every programming language.

References

[1] Carl Ellison, " Improvements on Conventional PKI Wisdom " *1st Annual PKI Research Workshop---Proceedings*, 2002.

[2] Mike P. Papazoglou, Willem-Jan van den Heuvel, "Service oriented architectures: approaches, technologies and research issues " *The VLDB Journal (2007) 16:389–415.*

[3] Marc Novakouski, Soumya Simanta, Gunnar Peterson, Ed Morris, Grace Lewis,"Performance Analysis of WS-Security Mechanisms" in *TECHNICAL REPORT CMU/SEI-2010-TR-023 ESC-TR-2010-023, 2010.*

[4] Anoop Singhal, "Web Services Security: Challenges and Techniques " in *Computer Security Division, NIST*, 1997.

[5] N. Vatra."Public key infrastructure overview", Scientific Studies and Research, Series Mathematics and Informatics, vol.19, no. 2, pp.471 – 478, 2009.

[6] Heena Kharche, Deepak Singh Chouhan. "Building Trust in Cloud using Public Key Infrastructure": International Journal of Advanced Computer Science and Applications, vol. 03, no. 03, pp.26 – 30, 2012.

[7] Eugene Xavier.P, Naganathan.E.R."Architecting Digital Signature/ PKI based Secure Web Based Systems Through 3-D Probabilistic Software Stability Model (PSSM)". International Journal of Computer Science and Network Security. vol.10 no.9, pp 5-11, 2010.

[8] Entrust®,Inc."Securing Digital Identities & Information".http://www.entrust.com/what-is-pki/, 2014.

[8] Brands, S.A. "Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy". MIT Press, 2000.

[9] K.Schmeh, "Cryptography and Public Key Infrastructure on the Internet", 2003.

[10] Wylie Shanks, "Building and Managing a PKI Solution for Small and Medium Size Business". SANS Institute Reading Room Site, 2013.

[11] Oracle Java SE Documentation , "Java Cryptography architecture" in http://docs.oracle.com/javase/7/docs/technotes/guides/security/ crypto/CryptoSpec.html.

[12] Oracle Java EE tutorial, "Building webservice with JAX-WS" in http:// docs. oracle.com/javaee/6/ tutorial/doc/bnayl.html

Biography

Dr. EBOT EBOT ENAW obtained his B.Eng hons degree from Liverpool University in Electronic Engineering in 1989. He later obtained an M.Eng degree in Telecommunication Engineering from The University of Manchester England in 1991. He returned home where he was recruited in the University of Yaounde I, as an assistant lecturer. He pursued his university studies and obtained a PhD in Computer Sciences from the National Advanced School of Engineering of the University of Yaounde I, where he is currently a senior lecturer. His area of specialization include: computer network security, cryptography and formal specification and verification; theorem proving and model checking. He has published some research articles in international journals namely *Formal model of a group key agreement protocol. Journal of Computational Technologies, 7(4):4–20, 2002*. In 2006 he was appointed Director General of the National Agency for Information and Communication Technologies Cameroon, a position he occupies till date. Major activities of the agency include amongst others: securing the Cameroon cyberspace through three key services: Computer Incidents Response Team (CIRT), Public Key Infrastructure (PKI) and Computer Security Audits.
Dr. EBOT EBOT ENAW may be reached at

ISSN:2319-7900

ebotenaw@yahoo.com

DJOURSOUBO PAGOU Prosper obtained his Master degree in Computer science engineering from the National Advanced School of Engineering of the University of Yaounde I in 2009. He holds several certifications in networking and cybersecurity namely CCNA, CCNP, CEH, ECSA. In 2013, he was appointed subdirector of the National Computer Incidents Response Team (CIRT) of Cameroon, a position that he occupies till date.