International Conference on Computing, Communication System and Informatics Management (ICCCSIM)

Venue : Hotel RAMADA , Bur Dubai, UAE

Date : 29 – 30 July, 2012

An Open-Source Based Application for Creating and Verifying Digital Signatures for XML Documents

Hesham Naser Elzentani

Electrical and computer engineering

Faculty of technical sciences, University of Novi Sad

Novi Sad, Serbia

hesham342002@yahoo.com

Abstract :

Introduce the smart card technology and capabilities of access it from Java platform, also introduce the XML Signature specification and how signatures can be created/validated by using the Apache Santuario library. Specifying the system for the digital signing of XML documents using smart cards, digital signatures are consistent with the XML Signature specification and implement the system by using open source components and the Java platform.

Keywords - *Smart Card, Digital Signature, XML signature*

I. INTRODUCTION TO DIGITAL SIGNATURE

Digital signatures can be classified into two main categories: single signature and multiple signature, it relies on public key cryptography (PKC). Different PKC schemes have been used to implement digital signature and data encryption. For example, The RSA (Rivest-Shamir-Adleman) scheme, The Digital Signature Algorithm (DSA) scheme, The ElGamal scheme and The elliptic curve digital signature algorithm (ECDSA) scheme [1]. Digital signature scheme guarantees information security properties, which are Authentication, Non-repudiation and Integrity [3]. Figure 1 shows the steps followed when a single user signs a document, figure 2 illustrates the steps followed by the receiver of the signed message [2]. The Crypto-hash function is a one-way algorithm that converts a sequence of characters into a shorter fixed-length value.

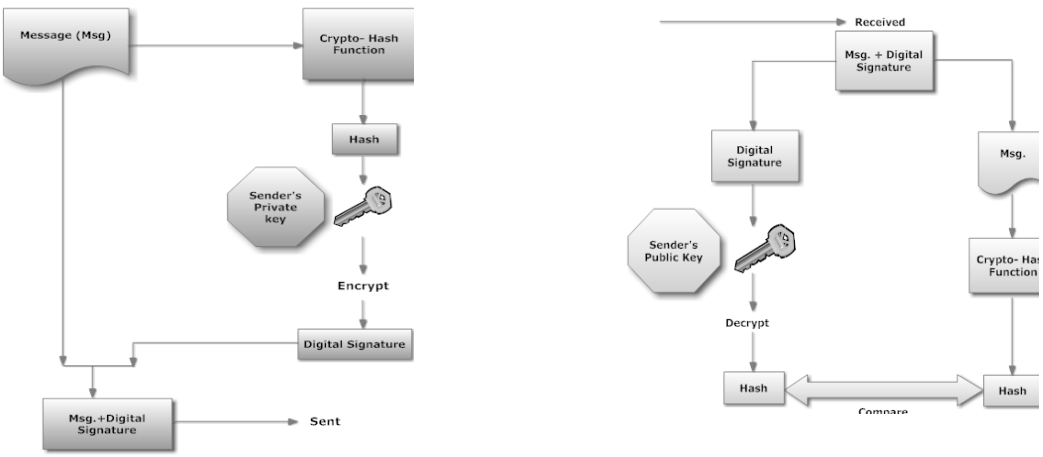

Figure 1. A single user signing a message

II. INTRODUCTION TO SMART CARDS

A smart card is a portable and tamper-resistant computer, carrys both processing power and information [5]. Private keys are stored in the card itself and are never sent outside the card. They also have the ability to store a large number of personal keys. Thus it is possible to have separate keys for different applications. They can also perform the encryption process within the card itself [12].

A. Types of Cards

The types of cards are: Embossed, Magnetic Stripe, Smart cards, Memory Cards, Microprocessor Cards, Cryptographic Coprocessor Cards, Contactless Smart cards and Optical Memory Cards [6], [7].

III. INTRODUCTION TO XML DIGITAL SIGNATURE

The very features that make XML so powerful for business transactions are the application of encryption and digital signature operations to XML-encoded data . Two new security initiatives designed are XML Signature and XML Encryption. XML Signature is a joint effort between the World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF), and XML Encryption is solely W3C effort. XML signatures are digital signatures designed for use in XML transactions [4].

A. The Components of an XML Signature

Figure 3 shows the XML Signature structure [4], there are three types of the XML Signature algorithms [8]:

1) *Enveloped XML Signature.*
2) *Enveloping XML Signature.*
3) *Detached XML Signature.*

B. How to Create an XML Signature

Here are quick steps of how to create an XML signature [4]:

1) *Determine which resources are to be signed.*
2) *Calculate the digest of each resource.*
3) *Collect the Reference elements.*
4) *Signing.*
5) *Add key information.*
6) *Enclose in a Signature element*

C. *Verifying an XML Signature*

A brief description of how to verify an XML signature [4], [8], [9]:

1) Verify the signature of the <SignedInfo> element. To do so, recalculate the digest of the <SignedInfo> element (using the digest algorithm specified in the <SignatureMethod> element) and use the public verification key to verify that the value of the <SignatureValue> element is correct for the digest of the <SignedInfo> element.

2) If this step passes, recalculate the digests of the references contained within the <SignedInfo> element and compare them to the digest values expressed in each <Reference> element's corresponding <DigestValue> element.

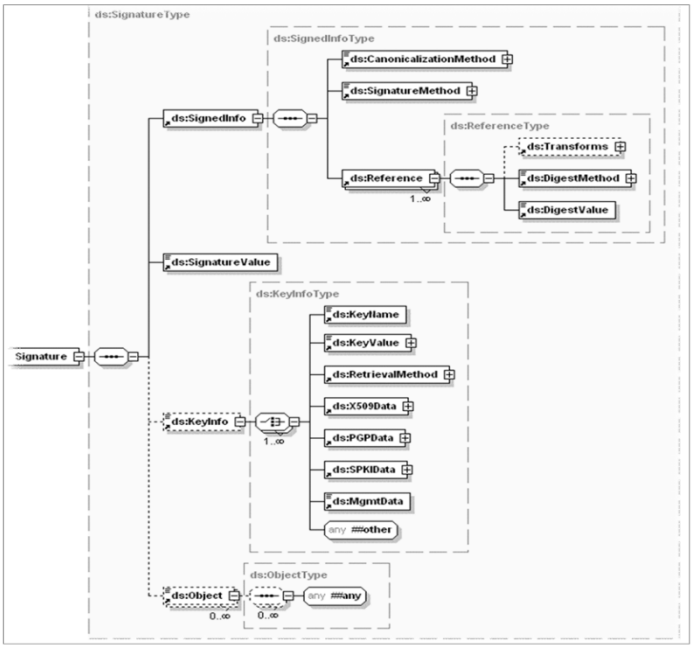

Figure 3. *XML Digital Signature structure*

IV. System Specification and Functions

A. *Card Initialization and Personalization*

Smart card must be initialized and personalized. Figure 4 shows the use-case diagram of card initialization. The initialization is the process of creating initial directory and file layout on the smart card and setting the Personal Identification Number (PIN) (including its Personal Unblocking Key, PUK). In this model, layout is created in accordance with the PKCS#15 standard. In order to create this layout, the card's transport key must be verified. Only when this key is successfully verified, the card will allow creation of folders and files on it. If card contains any other data, it must be deleted, before a new layout is created.

Figure 5 shows the use-case diagram of the card personalization, it is a process of generating the keys on the smart card and creating the user's certificate. The certificate creation includes load of the public key form the card, entering the user's data, load of the certificate authority (CA) keys and signing the created certificate with CA's private key. On creation, the certificate is stored on the smart card.

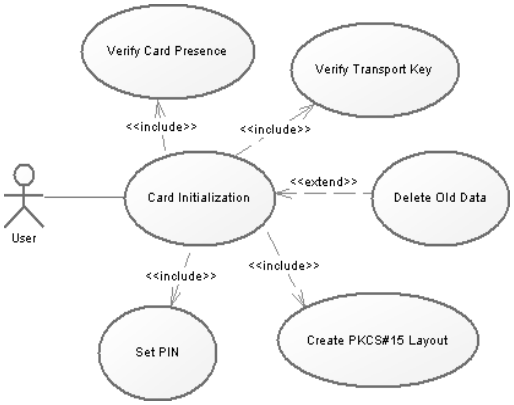

Figure 4. The use-case diagram of the smart card initialization

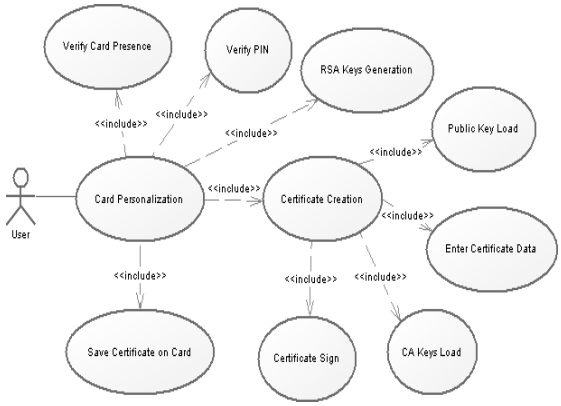

Figure 5. The use-case diagram of the smart card personalization

B. Creation of XML Signature

The creation of digital signature on XML document is presented in Figure 6. After the required XML file is loaded for signing, the card's PIN is verified. The process of signing XML document consists of creating the ds:Signature element which contains ds:KeyInfo within the user's certificate which is loaded from the card. The created ds:Signature is sent to the smart card for signing.

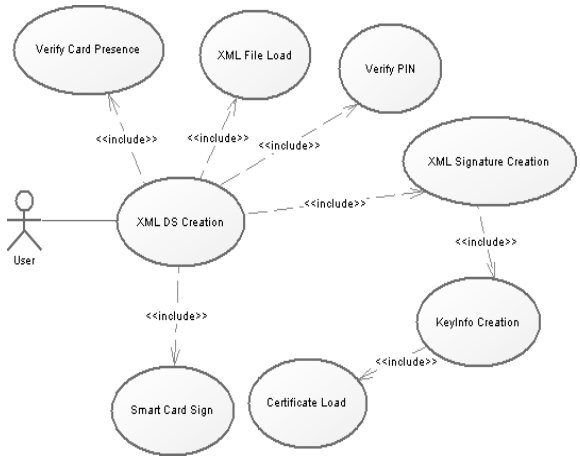

Figure 6. *The use-case diagram of the XML digital signature creation*

C. *Verification of XML Signature*

The verification of a digital signature on XML document started with loading the required document, select the signature which will be verified (if the document contains more than one signature), and verifying the signature using the retrieved certificate which stored inside the ds:KeyInfo element. In this activity, accessing to the smart card is not required, because the certificate is stored inside the ds:KeyInfo element (see Figure 7).

D. *Card Personalization Model*

The class diagram presented in Figure 8, shows the classes used for the smart card personalization. The presented model is relied on the Java Security API. The main class is *CertificateGenerator* which loads a user's private key, creates a certificate and stores it on a card. This class uses *SmartCardManager* for accessing to a smart card. Since a card is viewed as a key store file, the *SmartCardManager* class uses appropriate *KeyStore* class to accessing the card. The class *SubjectData* models a card's owner data, while *IssuerDate* models data of a card issuer. Beside standard Java security providers, two additional providers must be registered: *SunPKCS11* for a smart card access and *BouncyCastleProvider* for a certificate signing. *CertificateGenerator* uses *X509V3CertificateGenerator* from Bouncy Castle project to generate certificate. The generated certificate is signed using a CA's private key stored in the key store file.

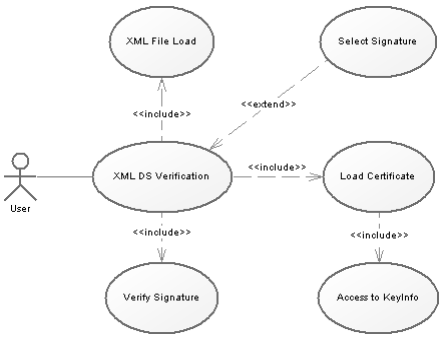

Figure 7. The use-case diagram of the XML digital signature verification

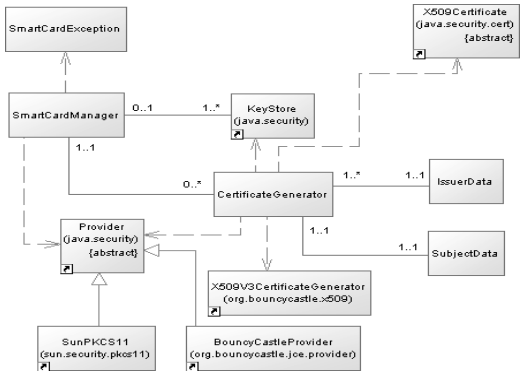

Figure 8. The class diagram of the smart card personalization

E. *Signature Model*

The class diagram shown in Figure 9 are the classes used for signature generation and verification. The class *SignatureGenerator* is used for creating digital signature on XML documents, while *SignatureValidator* is used for signature verification. *SignatureGenerator* uses appropriate specialization of the *Signature* class to create a signature. The *SmartCardSignature* class is used to create a signature using a smart card. If a private key is stored on some other medium or different type of a smart card appropriate specialization of *Signature* should be implemented. *SmartCardSignature* uses *SmartCardManager* for card accessing. The class *DOMUtil* provides method for accessing to an XML document and creating the DOM model, while the class *XMLSignatureUtil*

provides methods for creating a digital signature on XML documents. XML signatures are created using the *XMLSignature* class from Apache Santuario project. The errors that can occur during a signature generation and verification are modeled with the *SmartCardException* and *SignatureException* classes.

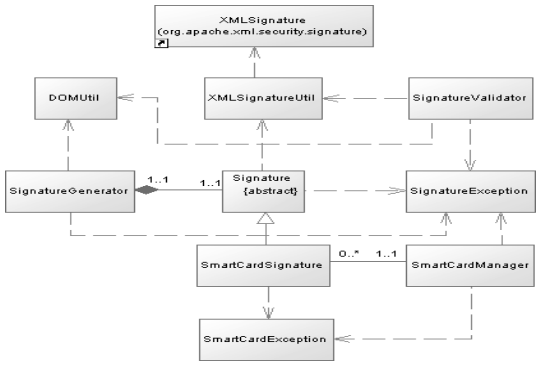

Figure 9. *The XML digital signature class diagram*

F. Signature Application Model

The major classes of the GUI application used for the signature generation and verification are presented in Figure 10. The class *SignatureApp* is the main class of the application. This class uses *RsyntaxTextArea* to show an opened XML document. The activities supported by this application are modeled as appropriate specialization of the *SignatureAbstractAction* class. The class *SignatureSignAction* uses *SignatureGenerator* to create a digital signature on the XML document and *SmartCardSignDialog* to accept a user's PIN. *SignatureVerifyAction* represents application activity of signature verification. It relies on the *SignatureValidator* class. If the opened document contains more than one signature the class *MultipleSignaturesDialog* enables a user to select which signature wants to verify.

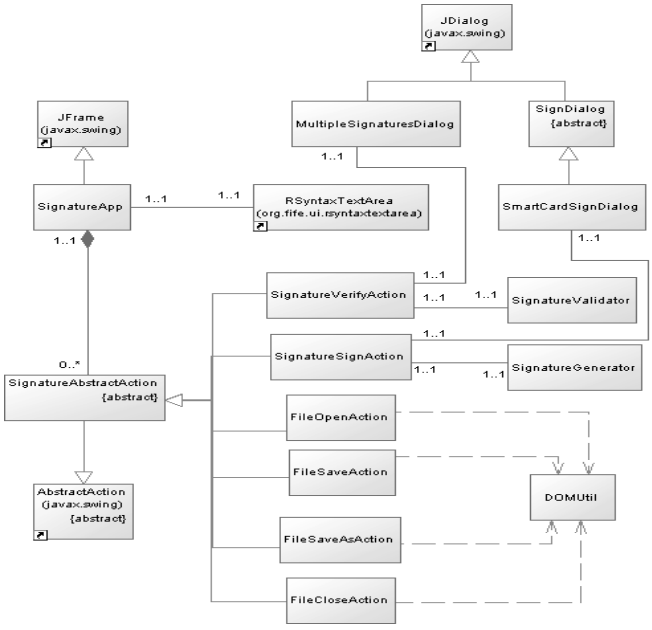

Figure 10. *Signature application class diagram*

V. SYSTEM IMPLEMENTATION

A. Smart Cards Communication

The communication with smart cards is realized through the OpenSC library [10]. It implements the PKCS#11 API. On the card OpenSC implements the PKCS#15 standard. This library, provides the tool for the card initialization, and PKCS#15 libraries for Linux and Windows based platforms.

B. Smart Card Initialization

OpenSC pkcs15-init tool creates PKCS#15 layout on the card, and sets PIN with following commands:

```
pkcs15-init –C
pkcs15-init --store-pin --id 01 --label "Hesham"
```

If the card is not empty it can be deleted with the following command:

```
pkcs15-init --erase --use-default-transport-key
```

C. Access to Smart Card from Java

The Java platform defines a set of programming interfaces known as the Java Cryptography Architecture (JCA) and the Java Cryptography Extension (JCE), which are performing cryptographic operations [11]. To facilitate the integration of native PKCS#11 tokens into the Java platform, a new cryptographic provider, the Sun PKCS#11 provider, it has been introduced into the J2SE 5.0 release. This new provider enables existing applications written to the JCA and JCE APIs to access native PKCS#11 tokens. The Sun PKCS#11 provider, acts as a bridge between the Java JCA and JCE APIs and the native PKCS#11 cryptographic API [11].

1) Sun PKCS#11 Configuration

Listing 1 shows example of creating The Sun PKCS#11 provider programmatically, and Listing 2 contains content of a configuration file.

```
String configName = “pkcs11.cfg“;
Provider p = new SunPKCS11(configName);
Security.addProvider(p);
```

Listing 1. Creating and configuring PKCS#11 provider

```
name = XMLSmartCardOpenPKCS11
library = opensc-pkcs11.dll
attributes(*,CKO_PRIVATE_KEY,*) = {
```

CKA_TOKEN = true

CKA_PRIVATE = true

CKA_DECRYPT = true

CKA_SIGN = true}

Listing 2. Content of pkcs11.cfg

D. Certificate Creation

JCA provides mechanism to access a smart card via the *KeyStore* class. In this case, a smart card is viewed as a *KeyStore* object. The *PKCS11* type of *KeyStore* is created, and also the PIN is provided. The following code illustrates this:

```
KeyStore scks = KeyStore.getInstance("PKCS11");
scks.load(null, pin);
```

Generation of RSA keys on a smart card can be realized using the *KeyPairGenerator* class. It is important to create an instance of this class using previously created *SunPKCS11* provider. Then keys will be generated on the card. The following code shows how keys can be generated on a smart card:

```
KeyPairGenerator gen = KeyPairGenerator.getInstance("RSA", provider);
generator.initialize(1024);
KeyPair keyPair = gen.generateKeyPair();
```

Creation of a certificate is presented in Listing 3. On creating X509V3CertificateGenerator instance, the required data is set, and finally the certificate is signed by issuer private key.

```
X509V3CertificateGenerator gen = new X509V3CertificateGenerator();
gen.setPublicKey(subjectData.getPublicKey());
gen.setSubjectDN(subjectData.getX590Name());
gen.setSerialNumber(subjectData.getSerialNumber());
gen.setNotBefore(subjectData.getStartDate());
gen.setNotAfter(subjectData.getEndDate());
gen.setIssuerDN(issuerData.getX509name());
gen.setSignatureAlgorithm("SHA1withRSA");
gen.generate(issuerData.getPrivateKey());
```

Listing 3. Certificate creation

Saving a private key and a certificate on a smart card can be realized with the following code:

```
scks.setKeyEntry("Hesham",keyPair.getPrivate(),null, certs);
scks.store(null, pin);
```

E. Signature Creation

Signing document using the Enveloped style is presented in Listing 4. The *XMLSignature* class is used for creating the XML signature. The certificate used for the signature validation is set by calling the *addKeyInfo* method. When the *sign* method is called, the hash code is generated on the computer and then it is sent to the card which will perform encryption of it.

```
Element rootEl = doc.getDocumentElement();
XMLSignature sig = new XMLSignature(doc, null,
        XMLSignature.ALGO_ID_SIGNATURE_RSA_SHA1);
Transforms t = new Transforms(doc);
t.addTransform(Transforms.TRANSFORM_ENVELOPED_SIGNATURE)
t.addTransform(Transforms.TRANSFORM_C14N_WITH_COMMENTS);
sig.addDocument("", t, Constants.ALGO_ID_DIGEST_SHA1);
sig.addKeyInfo((X509Certificate) cert);
rootEl.appendChild(sig.getElement());
sig.sign(privateKey);
```

Listing 4. Signing XML document

F. Signature Verification

Listing 5 shows the code that validate required signature. From a XML document, are retrieved all *ds:Signature* elements, and then the required (a signature that a user wants to verify) is selected. From the selected element the *XMLSignature* instance is created, and the certificate is retrieved from the signature's *ds:KeyInfo* element. The retrieved certificate is used for the signature verification.

```
NodeList signs = doc.getElementsByTagNameNS("http:// www.w3.org/2000/09/xmldsig#", "Signature");
Element signatureEl = (Element) signs.item(sigToVerify);
XMLSignature sig = new XMLSignature(signatureEl, null);
KeyInfo keyInfo = sig.getKeyInfo();
```

Certificate cert = keyInfo.itemX509Data(0).

itemCertificate(0).getX509Certificate();

return sig.checkSignatureValue(cert);

Listing 5. Signature verification

VI. CONCLUSION

Portability of XML as a data format, and its growing use in electronic business communications were the main reasons for creating the specifications for the representation of the electron signature using the XML syntax - XML Digital Signature. The goal of this paper is to create a Java-based application that supports digital signatures of XML document. A smart card is initialized by PKCS#15 standard. The Java Cryptography Architecture (JCA) and the Java Cryptography Extension (JCE) are used to access smart cards according to PKCS#11 standard. The Apache Santuario library is used to create a digital signature for an XML document.

REFERENCES

1. Noakes-Fry, Kristen., Digital Signatures: Perspective. August 10, 2000.
2. Wheatman, Victor S. ; Noakes-Fry, Kristen. Digital and Electronic Signatures: A Quick Look. May 16, 2003.
3. Chafic Maroun, Rouhana Moussa, Digital Signature and Multiple Signature:Different Cases for Different Purposes, (July 7th, 2003).
4. Ed Simon, Paul Madsen, Carlisle Adams, An Introduction to XML Digital Signatures, (2001).
5. Zhiqun Chen, Java Card™ Technology of Smart Cards.
6. Wolfgang Rankl and Wolfgang Effing. Smart Card Handbook - 3rd Ed.
7. Steve Petri, An Introduction to Smart Cards.
8. Yuri Demchenko, Providing Integrity and Confidentiality with the XML Security: Digital Signature and XML Encryption overview and usage examples, (2005).
9. W3C, XML-Signature Syntax and Processing.
10. OpenSC Project, accessed 06.12.2010, http://www.opensc-project.org/opensc
11. Java PKCS#11 Guide, accessed 06.12.2010, http://download.oracle.com/javase/ 1.5.0 /docs/guide/security/p11guide.html
12. Sandeep Anand, Incorporating Biometrics and Smart Cards, Department of Computer Science, University of Auckland