Journal of Information Engineering and Applications
ISSN 2224-5758 (print) ISSN 2224-896X (online)
Vol 1, No.2, 2011

A Practical Approach for Implementation of Public Key Infrastructure for Digital Signatures

M. Indra Sena Reddy[1*] P.J. Bhat[2] Rajeev Chetwavani[3] and K.Subba Reddy

2. R. G. M. College of Enginereing & Technology, Nandyal, A.P, India.
3. School of Economics, Peking University, 176 Zhong Guan Cun Street, Beijing 100086, China
4. ISRO, Satellite centre, Bangalore, India.
5. R. G. M. College of Enginereing & Technology, Nandyal, A.P, India.

* E-mail of the corresponding author: mir555mittapalli@gmail.com

Abstract

In an organization with its own Private Network , it is not just enough to transfer the documents from one person to another, but also it needs to ensure that the document retains its integrity, confirms the authenticity of the sender, provides privacy, if required and it is safe against repudiations. To address these, it is proposed to establish a public key infrastructure (PKI) for digital signatures with in the organizations. Public key infrastructure provides robust and rigorous security measures to protect user data and credentials. In this paper, it is proposed that documents are digitally signed on before being sent as an authentication measure. The trust between two parties and digital signatures are reinforced by components of public key infrastructure (PKI) namely Public Key Cryptography, Certificate Authority (CA), Certificates, Certificate Repository (CR), and also a simple application to demonstrate the same would also be attempted.

Keywords: Open SSL, Public Key Infrastructure, Digital signatures, Certificate Authority, Certificate Repository.

1. Introduction

Many applications require agreeing on a shared secret. It would be nice to be able to authenticate messages without needing to share a secret. Public key cryptography makes this possible. If a person signs a message with his secret signing key, then anyone can use his public key to verify that he signed the message. Essentially, the public key and private key are interchangeable. If a person encrypts a message with his private key, anyone can decrypt it. If a person didn't encrypt the message, using his public key to decrypt the message would result in garbage. This popular scheme called DSA (the Digital Signature Algorithm) for Digital Signing, which the SSL protocol and the Open SSL library both support.

An attacker might establish a proxy connection to the intended server, and then just eavesdrop on all data. Such an attack is called a "man-in-the-middle" attack. This attack is possible because public key cryptography provides no means of establishing trust when used on its own. Public Key Infrastructure (PKI) provides the means to establish trust by binding public keys and identities, thus giving reasonable assurance that we're communicating securely with who we think we are.

Using public key cryptography, we can be sure that only the encrypted data can be decrypted with the corresponding private key. If we combine this with the use of a message digest algorithm to compute a signature, we can be sure that the encrypted data has not been tampered with. What's missing is some means of ensuring that the party we're communicating with is actually who they say they are. In other words, trust has not been established. This is where PKI fits in.

2. Literature review

Public key cryptography was conceived by Diffie and Hellman (1976, 1977) and Rivest, Shamir in and Adleman designed the RSA Cryptosystems (1978), the first public key system. Each public key cryptosystem has its own technical features, however they all share the property that given an encryption key it is computationally infeasible to determine the decryption key and vice versa. Theoretically, no confidential information needs to be exchanged before secure communication is possible. Everyone has access to the recipient's public key and even though the communication is private, the message cannot be authenticated. This shows that public key cryptography on its own is not

enough. If traditional paper based commerce are to be reproduced in the electronic environment, the following are required:

- Security policies to define the rules under which cryptographic systems should operate
- Products to generate store and manage certificates and their associated keys
- Procedures to dictate how keys and certificates are generated and distributed

A trusted and authenticated key distribution infrastructure is necessary to support the use of public keys in a public network such as the Internet. Recent efforts in standardization have seen developments on a number of fronts.

2.1 Evolution of PKI standards

The X.509 Recommendation provides a useful basis for defining data formats and procedures for the distribution of public keys via certificates that are digitally signed by CAs. X.509 does not however include a profile to specify the supporting requirements for many of the certificate's sub- fields, extensions or for some data values.

The standards effort produced an outline for PKI of X.509 Version 3 certificates as well as Version 2 Certificate Revocation Lists. The Internet PKI profile went through eleven draft versions before becoming RFC 2459 Housley *et al* (1999). Other profiles have been developed for particular algorithms to make use of RFC 2459.

The development of the PKI management protocols has gone though a number of iterations. RFC 2510 Adams and Farrell (1999) was developed to specify a message protocol to be used between entities in a PKI. The need for an enrolment protocol and the preference to use PKCS#10 message format as the certificate request syntax lead to two parallel developments.

The Certificate Request Syntax was developed in the S/MIME WG which used PKCS#10 Yongge Wang (1993) as the certification request message format. RFC 2510 Adams and Farrell (1999) and Myers *et al* (1999) were developed to define an extended set of management messages that flow between the components of the Internet PKI. It also included PKCS#10 as the certificate request syntax.

Development of the operational protocols has been more straightforward. Two documents for LDAP have been developed — one for defining LDAPv3 as an access protocol to repositories Wahl *et al* (1997) and one for storing PKI information in an LDAP directory Boeyen *et al* (1999). Using FTP and HTTP to retrieve certificates and CRLs from PKI repositories is specified in RFC 2585 Housley and Hoffman (1998). The theoretical approach for working of PKI and digital signatures has been explained elsewhere (Indrasena Reddy *et al,* 2011). The practical approach for working of open SSL based PKI and digital signatures has been explained in the following sections.

3. Implementation of PKI

3.1 Setting up of the PKI

This involves two main steps

- The configuration file, openssl.cnf which is referred for three openssl commands. Namely, ca, request and x509.
- Make the necessary changes to openssl.cnf present on linux system at location /etc/pki/tls. The configure file openssl.cnf is shown below.

```
#########################################################################################
[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = IN
countryName_min                 = 2
countryName_max                 = 2
stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = A.P
```


localityName = Locality Name (eg, city)

localityName_default = NANDYAL

0.organizationName = Organization Name (eg, company)

0.organizationName_default = RGMCET

organizationalUnitName = Organizational Unit Name (eg, section)

organizationalUnitName_default = CSE

###
###

the next step is creation of two new folders. This is as follows.

- Create two new folders named CA and PKI

  # mkdir /CA

  # mkdir /PKI

- Run the openssl command to generate the self signed root certificate

  openssl genrsa -out /CA/cakey.pem 1024.

  openssl req -new -x509 -extensions v3_ca -keyout /CA/cakey.pem -out cacert.pem -days 365 -config /etc/pki/tls/openssl.cnf

  - When this command is run it asks the user to enter a password for the root certificate. In this case we have used ***ca123*** as the password.
  - The CA's private key will be in the cakey.pem file.
  - Using this private key and the configuration file for openssl.cnf, we create x509 v3 CA root certificate named cacert.pem. This is a self signed root certificate.

3.2 Request a certificate

- Next run the shell script with a user name and password as command line arguments. This password will be used to protect the p12 file.

The working of script as follows

- A check is made to ensure both the arguments have been entered.
- If the directory '/PKI' does not contain the certificate of the specified user then, a new one is created. This folder shall be used for storing the p12 and the certificate of that particular user.
- In the file /home/trainee/tr00750/indra/PKI//req.conf the necessary req and req_distinguished information of the user is stored.
- The user's pfx password is converted to octal format and along with the user's name, is stored in a file priv.password at the location /home/trainee/tr00750/indra/PKI/[user].
- Using the openssl command *genrsa,* a 1024 bit private key is generated and subsequently stored in a file priv.pem at the location /PKI/[user].
- A request for a certificate contains the public key corresponding to user's private key. Using the openssl command *req*, the public key is extracted and later is stored in request.pem at the location /PKI/[user].
- Next, we use openssl *ca* command. The CA uses it's password to perform a digital signature on the user's request. The outcome of this operation is a Certificate of the user duly signed by the CA.
- Also, in our project we used PKCS#12 format as an enhanced security measure. The p12 file is password protected file which contains the sensitive data such as private key of the user. So we use the openssl *pkcs12* command to generate the [*user_name].p12* file.
- The previously, readable pem format certificate is finally encoded into the unreadable DER X509 standard.
- For convenience, we store the p12 files and certificates separately in folders /PKI/P12-files and /PKI/Certificates respectively.
- The above procedure shown in the Figure 1.

3.3 Signing a Document

In this section, we describe the procedure for performing digital signature on a document.

1. This method opens the file to be signed, reads it and then return the data contained.
2. After taking the p12 file-name and pfx password from the user, a keystore of the type PKCS12_KEYSTORE_TYPE is created and the p12 file-name and it's pfx password is stored in the keystore.

The call to this method is as follows:

```
userKeyStore = loadKeyStoreFromPFXFile(keyStoreFileName, password);
```

3. The private key is read from the keystore and a call is made to the signDocument as follows:

```
privateKeyC =getPrivateKey(userKeyStore, password);
byte[] digitalSignature = signDocument(documentToSign, privateKey);
```

The Digital Signature Algorithm (DSA) is used for digitally signing the document, the Java methods used to accomplish this task.

4.The above procedure shown in the Figure 2.

3.4 Verification of the Signature

In this section, we describe the procedure for verifying the digital signature on a document.

The verification of the digital signature is performed using a JSP program.

Here are some of the major code snippets:

1. In this paper it is assumed that the PKI functions are performed centrally at a server. Therefore user interested in using its verification services is provided with a UI coded in JSP.

Here is an extract from the program that accepts the file to be verified, the signature file and sender's id.

```
##########################################################################################
###
<html:form type="demo.SignedFileUploadActionForm" action="/public/SignedFileUpload"
method="post" enctype="multipart/form-data">
<table border=0 align=center>
<tr>
        <td><font size=4><b> File to be Verified:</b></font></td>
        <td><html:file property="uploadFile" size="40" /></td>
</tr>
<tr>
        <td><font size=4><b> Digital signature: </b></font></td>
        <td><html:file property="sigFile" size="40"/></td>
</tr>
<tr>
        <td><font size=4><b>Senders id: </b></font></td>
        <td><html:text property="senderid" size="40"/> <input type=button
value="Search" style="color:green;font-size:15px;font-weight:bold" onclick="showSearch()">

        </td>
</tr>
<tr><td><br></td></tr>
<tr>
        <td align=center colspan=2><html:submit property="submit" value="Verify Signature"
```



```
style="color:green;font-size:20px;font-weight:bold" /></td>
</tr>
</table>
<hr>
</html:form>
```

##

2. For the successful verification of the document, digitally signed document, signature and sender's id are necessary.

The following snippets illustrate how these are stored for later access.

##

```
private SignedFileUploadActionForm mActionForm = null;
private FormFile mReceivedFile = null;
private byte[] mReceivedFileData = null
private CertPath mCertPath = null;
private X509Certificate mCertificate = null;
private byte[] mSignature = null;
private String mSignatureBase64Encoded;
private String senderid;
        {
        throw new Exception("No file received. Please upload some file.");
        }
        mReceivedFileData = mReceivedFile.getFileData();
}
```

##

3. The call to the method which reads the certificate:

This method has means to map sender id to the [*sender-id*].cert. All users' certificates are stored in server's 'Certificates' folder.

4. Signature is going to be verified here.

##

```
verifyDocumentSignature boolean signatureValid = DigitalSignatureUtils.(
mReceivedFileData, mCertificate, mSignature);
public static boolean verifyDocumentSignature(byte[] aDocument,
X509Certificate aCertificate, byte[] aSignature)throws GeneralSecurityException {
PublicKey publicKey = aCertificate.getPublicKey();
boolean valid = verifyDocumentSignature(aDocument, publicKey, aSignature);
```

##

The following method receives the public key that has been extracted from x509 certificate by the above method. Using this public key and the document that was signed with the Digital Signature Algorithm, the signature is verified.

##


```
###
public static boolean verifyDocumentSignature(byte[] aDocument,
PublicKey aPublicKey, byte[] aSignature)
throws GeneralSecurityException {
Signature signatureAlgorithm =
Signature.getInstance(DIGITAL_SIGNATURE_ALGORITHM_NAME);
signatureAlgorithm.initVerify(aPublicKey);
signatureAlgorithm.update(aDocument);
boolean valid = signatureAlgorithm.verify(aSignature);
return valid;
}
##########################################################################################
##
```

5. Verifying certificate:

```
verifyReceivedCertificate();
```

In order to check validity of the x509 certificate of the user, the validity of the issuer needs to be checked. This is accomplished by adding the CA(issuer)'s root certificate to a directory to which the program has direct access to. Finally using the verify method, as was earlier, the validity and authenticity of the user's certificate is checked against the x509 certificate of the issuer.

6. The above procedure shown in the Figure 4.

3.5 Encrypt the digitally signed document

In this section, we describe the procedure for encrypt the digitally signed document. Encrypt the digitally signed document is performed on any document of choice using a Java program.

Here are some of the major code snippets:

1. I n our project we can generate an AES key using keyGenerator initialise the
    a. key size to 128 bit.
2. Generate a secret key for the symmetric encryption.
3. Encrypt the signed document using symmetric key or secret key.
4. Encrypt the secret key with the receivers public key.
5. For the sending the document, encrypted signed document, encrypted symmetic key and sender's id are necessary.
6. The above procedure shown in the Figure 5.

3.6 Decrypt symmetric key and signed document

In this section, we describe the procedure for decrypt the symmetic key and signed document.

The decrypt symmetric key and signed document is performed using a JAVA program.

Here are some of the major code snippets:

1. Load the private key from specified file using the specified password to decrypt the symmetic key .

```
// Load the keystore from specified file using the specified password
```

2. Private key and certification extacted from given keystore using given password to access the keystore and the same password to access the private key.
3. The above procedure shown in the Figure 6.

4. Conclusions

In this paper, SSL provide an alternative to the TCP socket API that has security implemented within it.

SSL provide a private channel between communicating applications, which ensure privacy of data, authentication of the partners, and integrity. We have established means for establishing trust between two parties who are communicating data and credentials on a network. We accomplish this by setting up a private CA. A private CA exists for serving the sole purpose of acting as a trusted third party in a closed company network.

A document is digitally signed before it is sent to receiver. The digital signature detects tampering in transit and is non-repudiable. The CA reinforces the trust for the receiver. This is possible because the CA's signature on the sender's certificate tells the receiver that the signature was not performed by any random user in the network who was capable of generating a key pair. The signing and verification services are located centrally at a server. This means that the private key needs transporting to the server for signing. To ensure that the private key is not compromised in transit, we use p12 file to securely transport the private key.

REFERENCES

Indrasena reddy,Bhat and Rajiv,(2011). Establishment of Public key Infrastructure for Digital Signatures, Vol.2,No.6,pp 33-43

Pravir Chandra, Matt Messier, John Viega, (2002).Network security with Openssl, , O'reilly

Diffie, W. and Hellman, M. E.,(1976) New Directions inCryptography. IEEE Transactions on Information Theory,22 , pp. 644-654.

Rivest, R., Shamir, A. and Adleman, L.,(1978) A Method for Obtaining Digital Signatures and Public Key Cryptosystems. Communications of the ACM, 21, pp. 120-126.

Housley, R., Ford, W., Polk, W., and Solo, D., (1999) RFC 2459 "Internet X.509 Public Key Infrastructure Certificate and CRL Profile",

Adams, C., Farrell, S., (1999) RFC 2510, "Internet X.509Public Key Infrastructure Certificate Management Protocols",

Yongge Wang,(1993) PKCS#10, RSA, "The Public-Key Cryptography Standards ", RSA Data Security Inc.,

Myers, M., Adams, C., Solo, D., and Kemp, D.,(1999) RFC 2511, "Internet X.509 Certificate Request Message Format",

sWahl, M., Howes, T., Kille, S.(1997) RFC 2251"Lightweight Directory Access Protocol (v3)"

Housley, R., and Hoffman, P.,(1998) RFC 2585, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP",

Carlisle Adams, Steve Lioyd, and Second Edition. Addison Wesley, s November (2002). Understanding PKI: Concepts, standards, and Deployment Considerations,

Kaufman, Charlie, Perlman, Radia, Speciner, Mike(1995)" Network Security Private communication in a Public world", Prentice Hall,

William Stallings, (1999) Cryptography and Network Security: Principles and Practice, 2nd Edition, Prentice ,Hall.

Diffe,W.and Hellman, M.E,(1976), New Directions in Cryptography. IEEE Transactions on Information Theory, 22, pp. 644-654.

Rivets, R., A. and Adelman, L., (1978) A Method for Obtaining Digital signatures and Public Key Cryptosystems. Communications of the ACM, 21, PP. 120-126.

Wahl, M., Howe's, T., Kille, S.(1997) RFC 2251, "Lightweight Directory Access Protocol (V3).

www.certicom.com, Public- Key Infrastructure- The VeriSign Difference; VeriSign whitepaper (1999).

RSA Data Security ," Understanding PKI" (1999).

Ray Hunt,(2002) ," PKI and Digital Certification Infrastructure"

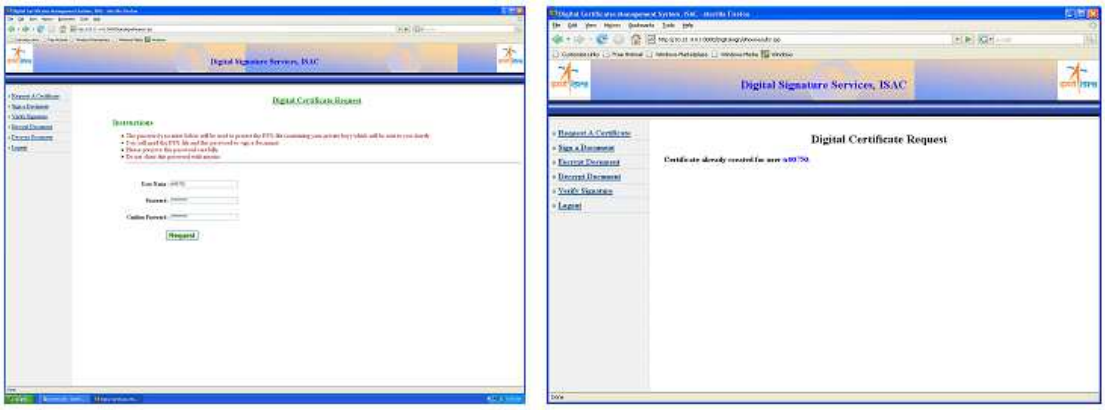

Figure 1. Request a Certificate

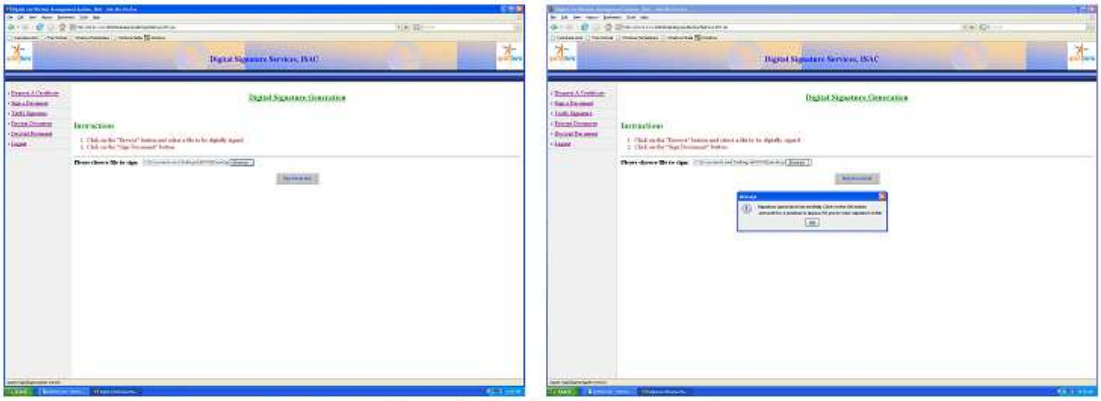

Figure 2. Signing the Document

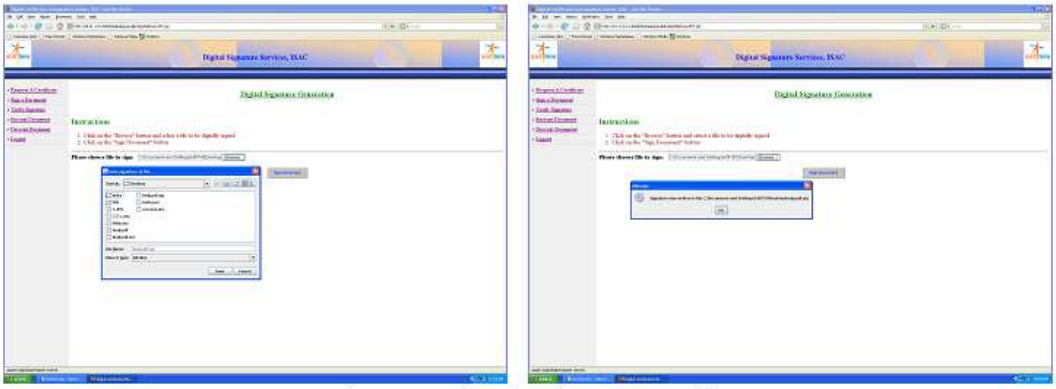

Figure 3. Saving the Signature as a File

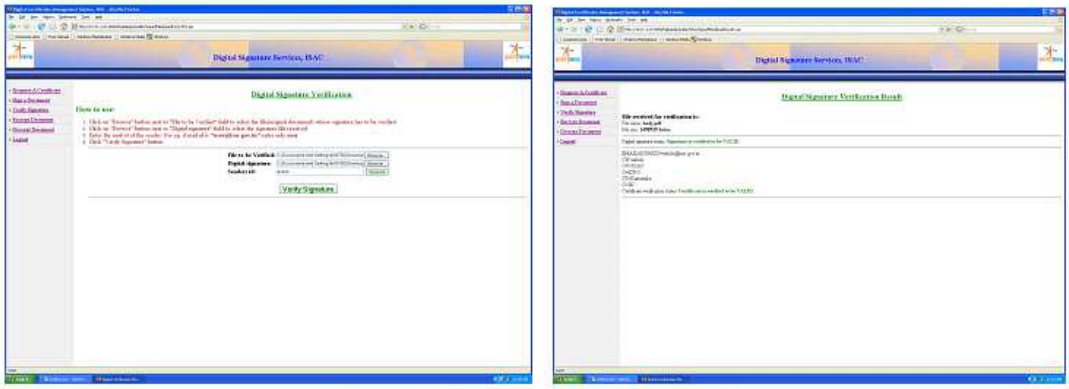

Figure 4. Verifying the Signed Document

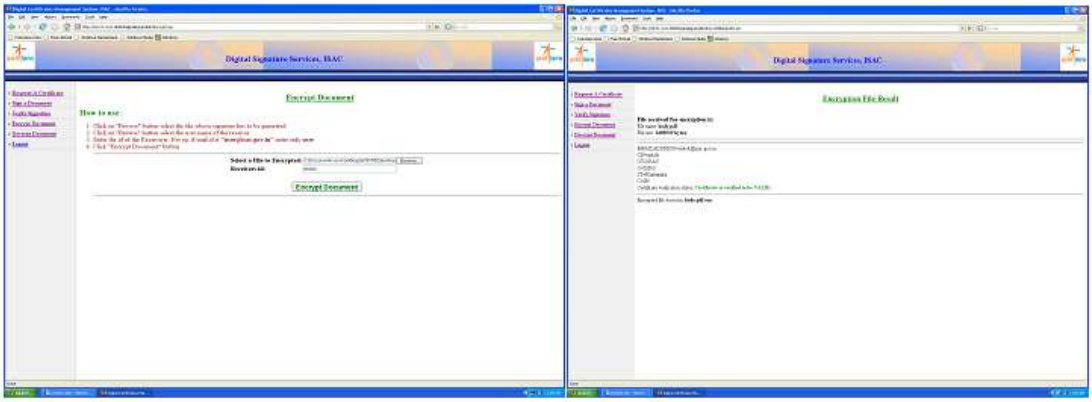

Figure 5. Encrypt Signed Document

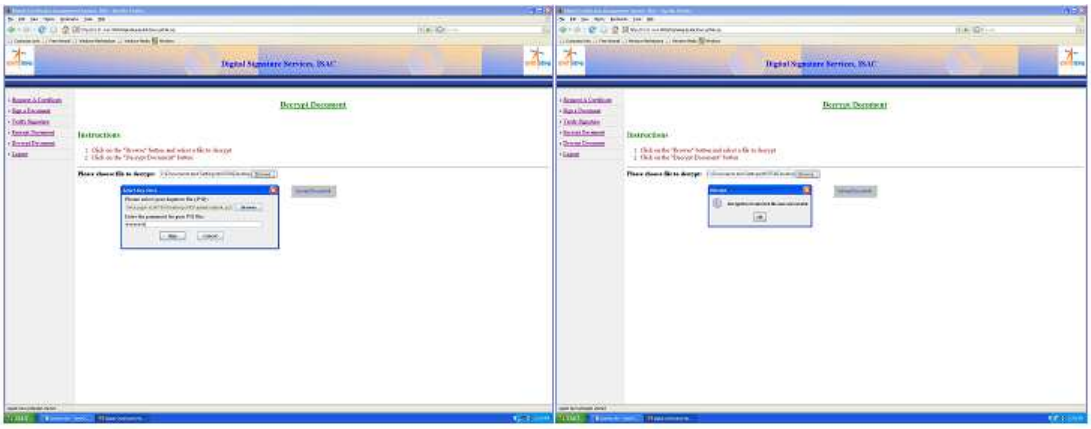

Figure 6. Decrypt Signed Encrypted Document