Securitate - Autentificare, Autorizare, Certificate si SSL

November 24, 2005

Contents

1 Introducere

Informatia schimbata prin retea este expusa la o serie de riscuri putand fi interceptata, alterata sau impersonificata. Criptarea informatiei utilizand chei simetrice sau asimetrice, calcularea digest-urilor mesajelor si semnarea lor, ofera solutii pentru schimbul privat al informatiei (codificarea mesajelor astfel incat este computational imposibil o descifrare a acestora) si asigurarea integritatii informatiei (mesajul nu poate fi corupt/modificat fara ca acest lucru sa fie detectat). In acest laborator ne vom concentra pe o problema inerenta oricaror sisteme distribuite/comunicarii prin retea si anume autentificarea.

2 Autentificare

Autentificarea are drept scop stabilirea identitatii actorilor care doresc sa comunice sigur in retea. Este in special necesara in tranzactiile financiare cand identitatile actorilor care interactioneaza trebuie stabilite sigur, dar poate servi si in procesul de autorizare [1] unde in functie de identitatea autentificata un anumit set de drepturi este acordat. Autentificare are drept scop inlaturarea celui de-al treilea risc enuntat - impersonificarea.

Sa consideram urmatorul exemplu, pentru a stabili contextul in care intervine autentificarea: Alice si Bob vor sa incheie un contract de sponsorizare prin care Alice se obliga sa sustina activitatea stiintifica a lui Bob. Alice este manager la o firma de calculatoare si are o agenda incarcata, iar Bob este un cercetator de succes, fiind ocupat cu scrierea de articole stiintifice, participarea la conferinte si seminarii, si coordonarea mai multor proiecte de cercetare. Problema care se ridica este cum ar putea Alice si Bob sa incheie acest contract fara a fi necesara o intalnire.

Utilizand mecanisme de criptare cei doi pot stabili o conexiune sigura, insa au nevoie de o metoda care sa garanteze ca Alice si Bob sunt persoanele care se pretind a fi. Alice si Bob pot genera cate un set de chei asimetrice, pot schimba intre ei cheile publice, urmand ca mai apoi fiecare sa demonstreze detinerea cheii private asociate celei publice. In acest scenariu cei doi ar trebui sa gaseasca o metoda pentru a schimba sigur cheiele publice. Doar pe baza informatie generate local de Alice si Bob, nu se poate gasi un scenariu sigur pentru schimbul cheilor publice care sa nu implice intalnirea celor doi. Problema care trebuie solutionata este realizarea unei asocieri cheie publica - identitate care sa ateste detinatorul de drept al chei publice, asociere care sa fie recunoscuta de cel caruia detinatorul i se autentifica.

3 Certificate

Functia principala a unui certificat este aceea de a asocia unei identitati o chei publica. Certificatele sunt publice si contin informatie referitoare la subiectul certificatului [2], cheia publica a certificatului, entitatea care a emis acest certificat si semnatura acesteia. Asocierea cheie-identitate (certificatul) este recunoscuta si garantata de un tert, entitatea care semneaza certificatul si care este numita **Autoritate de Certificare**.

[1] Autorizarea se refera la acordarea si restrictionarea drepturilor unei entitati

[2] Entiatea care detinea certificatul - numele, cu ce organizatie e asociat acest nume, cu ce unitate a organizatie, locatia si tara

Un certificat poate fi obtinut printr-o cerere facuta unei astfel de Autoritati de Certificare. Cel care doreste sa detina un astfel de certificat va genera local o pereche de chei, publica si privata, va arata cheia publica Autoritatii de Certificare care ii va creea si semna un certificat continand aceea cheie publica [3]. Autoritatea de Certificare asigura integritatea datelor din certificat prin calcularea unui hash peste intregul certificat si semnearea hash-ului cu cheia privata a Autoritatii de Certificare (care detine si ea un certificat si o cheie privata asociata).

Dupa cum se observa Autoritatea de Certificare trebuie sa fie considerata de incredare pentru ca asocierea cheie publica - identitate prezenta in certificat sa fie viabila. Este usor de imaginat ca la nivel global nu se poate stabili o singura Autoritate de Certificare, atat din motive de incredere cat si de scalabilitate. Astfel exista un numar de Autoritati de Certificare considerate radacina. Aceste autoritatii sunt cunoscute si considerate de incredere iar certificatele lor (deci si cheia publica) sunt deobicei incorporate in aplicatiile care fac uz de autentificare prin certificate. Un certificat semnat de o astfel de Autentificare de Certificare este considerat (hardcodat) de incredere, insa de cele mai multe ori Autoritatile de Certificare radacina nu emit certificate direct utilizatorilor ci altor Autoritati de Certificare care pot fi responsabile cu anumite regiuni, si care, la randul lor, ar putea emite certificate pentru alte Autoritati de Certificare, construindu-se cateva nivele pana la Autoritatile care emit certificate utilizatorilor (vezi fig 1).

Un certificat al unui client va fi verificat pe lantul de semnaturi pana ce se ajunge la o Autoritate de Certificare de incredere sau se va stabili ca un astfel de lant nu exista caz in care, autentificarea va esua.

Intregul sistem cu entitati care detin/cer certificate si Autoritati de Certificare poarta numele de Infrastructura cu Chei Publice (Public Key Infrastructure - PKI).

Autentificarea poate fi ceruta de ambele entitati care doresc sa comunice sau poate fi pretinsa numai de una dintre acestea. Principiul este urmatorul: entitatea care trebuie sa se autentifice (A) prezinta certificatul detinut [4], cealalta parte (B) verifica daca certificatul este de incredere si daca acesta este cazul, genereaza aleator un set de date si cere ca acestea sa fie semnate cu cheia privata pereche a chei publice din certificatul aratat. Dupa ce

[3] In principiu se creaza local o cerere de certificat, de fapt un certificat nesemnat, care contine identitatea entitatii si cheia publica, certificat care este trimis Autoritatii de Certificare spre semnare. Autoritatea de Certificare este responsabila sa verifice daca identitatea din certificatul primit, apartine celui care a trimis spre semnare certificatul. Deobicei acest proces se face offline, find implicate si chestiuni legislative

[4] De cele mai multe ori va prezenta un lant de certificate pana la o Autoritate de Certificare posibil cunoscuta de cealalta parte

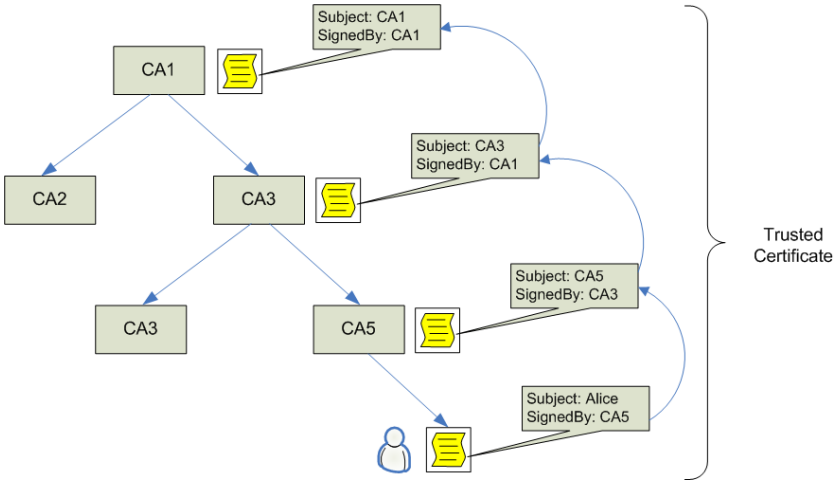

Figure 1: Autoritati de Certificare

primeste datele, B le decripteaza cu cheia publica (dezvaluita in certificatul aratat anterior) si daca obtine aceeasi valoare entitatea A (care a prezentat certificatul) este considerata detinatoarea de drept a acestuia, deci avand identitatea din certificat.

Certificatele in formatul X.509 sunt un standrd ITU-T pentru Infrastructura cu Chei Publice. Un certificat X.509 contine cel putin urmatoarele date:

- **Versiunea**. Exista trei versiuni de certificate X.509 v1, v2 si v3.
- **Numar Serial**. Identifica certificatul si trebuie sa fie unic intre certificatele semnate de Autoritatea de Certificare care emite si certificatul curent [5].
- **Algoritm de Semnare**.
- **Emitent**. Numele emitentului in format X500 (in general o Autoritate de Certificare)
- **Perioada de Valabilitate**. Data cand certificatul devine valabil si data cand acesta expira.
- **Subiectul**. Numele celui care detine certificatul (poate fi in format X.500 sau intr-un alt format - ex. URL).
- **Cheia Publica**.

[5]Numarul serial si numele emitentului identifica unic un certificat

- **Semnatura Emitentului**.

Numele in format X500 contin cateva campuri care ofera informatii suplimentare despre entitatea respectiva. In format X500 pot aparea urmatoarele atribute:

- Common Name CN - Numele detinatorului
- Organization O - Organizatia cu care este asociat numele.
- Organization Unit OU - Unitatea din cadrul organizatiei
- Country C - Tara

Certificatele X.509 v1 au aparut in 1988 si au utilizat formatul X500 pentru reprezentarea emitentului si subiectului unui certificat. X.509 v2 a introdus conceptul de identificatori unici pentru subiect si emitent pentru a se putea reutiliza aceste nume, insa aceasta propunere a fost controversata, si cele mai multe recomandari referitoare la certificate sunt impotriva refolosirii numelor. Certificatele X509 v2 au continuat sa utilizez formatul X500 de reprezentare a numelor. Certificatele X509 v2 nu au cunoscut o raspandire prea mare.

Certificatele X.509 v3 sunt cele mai recente, facandu-si aparitia in anul 1996. Noutatea adusa este prezenta extensiilor, care pot fi definite si incluse in certificate (semnatura certificatului este calculata si peste aceste extensii). Extensiile permit ca un certificat v3 sa cuprinda si alte informatii pe langa identitatea detinatorului (ex. atribute ale detinatorului). O serie de extensii sunt deja definite si acceptate ca atare (Utilizare Cheii - KeyUsage - specifica scopul in care poate fi utilizata cheia din certificat, Nume Alternative - AlternativeNames - pentru a asocia si alte nume cu cheia publica din certificat etc.) altele pot fi definite de utilizatori. Aceste extensii pot fi marcate critice sau necritice. O extensie critica ar trebui verificata si utilizata/interpretata de aplicatia care primeste certificatul respectiv, insa aceasta depinde de conventiile si modul de functionare a aplicatiei. In plus certificatele in format v3 nu obliga reprezentarea in format X500 a subiectului sau a emitentului putandu-se utiliza nume generice. Pentru un certifiat X509 v3 vezi Appendix A.

4 SSL

Sa revenim la exemplu initial, in care Alice si Bob doresc sa schimbe date prin retea. Utilizand certificate X509 si presupunand ca emitentii acestora

sunt considerati de incredere Alice si Bob se pot autentifica reciproc. Cei doi sunt interesati insa si de realizarea unei comunicati sigure, confidentiale, astfel incat mesajele schimbate sa nu poata fi descifrate.

SSL este un protocol de securitate care reuneste toate aceste mecanisme. Prin intermediul sau se poate asigura confidentialitate, integritatea mesajelor si autentificarea partilor. SSL actioneaza peste un flux TCP si ofera servicii nivelelor superioare. Protocolul HTTP poate fi utilizat peste SSL si in acest caz este numit HTTPS (Secure Hyper Text Transfer Protocol). HTTPS functioneaza pe acelasi principiu cu HTTP, diferenta constand in criptarea mesajelor schimbate intre un server web si un browseri si in posibilitatea autentificarii atat a serverului cat si a clientului. SSL este insa un protocol general care poate oferi servicii oricarui protocol de nivel aplicatie.

SSL a fost dezvoltat de Netscape Communications scopul urmarit fiind realizarea de tranzactii sigure, cu carti de credit pe Internet utilizand un browser si un server web. Versiunile 1 si 2 ale protocolului s-au dovedit a avea slabiciuni de securitate, insa in 1995 odata cu lansarea SSl v3, protocolul s-a impus ca un standard de facto.

In 1996 IETF a incercat sa standardizez protocolul SSL, rezultand protocolul Transport Layer Security (TLS) ce a adus modificari minore SSL v3. In 2001 s-a publicat RFC 2246 continand protocolul TLS insa modificarile aduse l-au facut incompatibil cu SSL v3. La acest moment situatia este neclara cu privire la protocolul acceptat pentru comunicatie sigura peste retea, majoritatea browserelor web fiind livrate cu suport atat pentru SSL v3 cat si pentru TLS.

Protocolul SSL este format din doua etape: *handshake* si *transfer*. In timpul handshake-ului clientul si serverul stabilesc un set de algoritmi pentru realizarea criptarii, stabilesc cheile care vor fi utilizate si se autentifica [6]. Dupa incheierea handshake-ului, in cea de-a doua faza, transferul, datele sunt sparte in blocuri de dimensiuni mai mici (optional pot fi si compresate) protejate pentru garantarea integritatii si criptate dupa care are loc transmisia propriuzisa la retea.

Handshake-ul este format din urmatorii pasi

1 Clientul trimite versiunea de SSL utilizata, lista algoritmilor de criptare suportati si un numar generat aleator

2 Serverul alege algoritmii care vor fi utilizati si ii confirma clientului alegerea. Serverul ii trimite certificatul sau clientului, si la fel un numar generat aleator.

[6]Doar autentificarea serverului este obligatorie, cea a clientului este optionala

3 Clientul verifica certificatul serverului, si daca este de incredere extrage din acesta cheia publica. Clientul calculeaza un secret premaster, il cripteaza cu cheia publica a serverului (extrasa din certificat) si trimite datele astfel codificate serverului.

4 Clientul si Serverul deriva din acest secret premaster si din numerele generate aleator schimbate in mesajele anterioare, un set de chei care vor fi utilizate in criptarea mesajelor si asigurarea integritatii informatiei schimbate.

5 Clientul calculeaza un MAC (Message Authentication Code) [7] peste toate mesajele de handshake schimbate si il trimite serverului.

6 Serverul calculeaza si el un MAC peste (posibil) aceleasi mesaje de handshake si il trimite clientului. Cele doua MAC-uri sunt comparate pentru a se verifica daca in timpul mecanismului de handshake a intervenit un atac.

5 JCE si BC

Java Cryptography Extension (JCE), este inclus in JDK incepind de la versiunea 1.4. Supportul JCE pentru criptare include:

- Symmetric bulk encryption (DES, RC2, and IDEA)
- Symmetric stream encryption (RC4)
- Asymmetric encryption (RSA)
- Password-based encryption (PBE)
- Key Agreement
- Message Authentication Codes (MAC)

Bouncy Castle[8] (BC) este un toolkit care ofera o implementare lightweight pentru JCE 1.2.1 si compatibil cu Java Micro Edition (J2ME) astfel incit este solutie cea mai buna pentru un API cryptografic pentru dispozitive mobile

[7] MAC calculeaza un digest al unui mesaj deci depinde de continutul acestui mesaj, dar in acest calcul intervine si o cheie. Aceasta cheie este cunoscuta de ambele parti fiind derivata in pasul anterior

[8] http://www.bouncycastle.org/

care ruleaza Java.

Security Providers

Se pot instala mai multi provideri de securitate si developer-ul poate adauga provideri in afara de cei instalat default de catre JCE. Exemplul urmator instaleaza provider-ul oferit de Bouncy Castle care se numeste **BC**.

```
import java.security.Provider;
import java.security.Security;
import java.util.Set;
import java.util.Iterator;

import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class ProviderInformation
{

    public static void main(String[] args)
    {
       Security.addProvider(new BouncyCastleProvider());
       Provider[] providers = Security.getProviders();
       for (int i = 0; i < providers.length; i++)
       {
            Provider provider = providers[i];
            System.out.println("Provider name: " + provider.getName());
            System.out.println("Provider information: " + provider.getInfo());
            System.out.println("Provider version: " + provider.getVersion());
            Set entries = provider.entrySet();
            Iterator iterator = entries.iterator();
            /*while (iterator.hasNext())
            {
                 System.out.println("Property entry: " + iterator.next());
            }*/
       }
    }
}
```

Default, pentru toti algoritmii criptografici se foloseste primul provider care apare in outputul programului de mai sus. Daca se doreste un anumit provider, se poate specifica prin numele lui, de exemplu "BC" pentru Bouncy Castle, in felul urmator:

```
Signature sign = Siganture.getInstance("SHA1WithDSA", "BC");
```

Tehnici pentru programare sigura

Parolele trebuie tinute in memorie nu ca *String*, ci ca array de caractere, si trebuie suprascrise cu zero imediat dupa folosire pentru a preveni *memmory sau disk snooping*. De exemplu, informatia ar putea sa fie citita usor atunci cind masina este obligata sa faca swapping. De asemena, atunci se serializeaza obiectele, se foloseste cuvintul cheie *transient* pentru ca informatia de pe aceste canale sa nu fie trimisa in streamul de date.

Password-Based Encryption

Password-Based Encryption (PBE) creeaza o cheie de criptare dintr-o parola. Pentru a minimaliza sansele ca un atacator sa ghiceasca parola prin forta bruta, implementarile de PBE implementations folosesc in plus un numar aleator (salt) pentru a creea cheia.

Exemplu: Pe orice sistem unix care are pachetul *openssl* instalat, puteti realiza scripturi care sa realizeze PBE.

```
Pentru criptare folosind idea-cbc:
>openssl  enc -idea-cbc -e -in plaintext_filename -out ciphertext_filename

Pentru decriptare:
>openssl  enc -idea-cbc -d -in ciphertext_filename
```

In anexa B este un exemplu de program Java care foloseste PBE si citeste parola fara a folosi clasa *String*.

Exercitiu: Realizati un alt program care citeste parola de la tastatura si realizeaza decriptarea unui text criptat cu exemplul de criptare din Anexa.

6 JAAS

Java Atuthentication and Authorization Service (JAAS) este un API pentru realizaarea autentificarii si autorizarii. Dupa cum spuneam, autentificarea inseamna determinarea in mod sigur a entitatii/grupului care executa codul Java (de exemplu un applet, o aplicatie, un servlet, etc). Autorizarea inseamna garantarea unor actiuni (de exemplu citire sau scriere a unor fisiere dintr-un director) in functie de drepturile entitatii/grupului. JAAS asigura un mod de gestionare complex ale autorizarii si autentificarii, gerstionind autentificarea unor entitati care pot avea mai multe roluri si asigurind decizii logice complexe si dinamice, la runtime, legate de autorizare. JAAS extinde astfel metodele de *access control* bazate pe semnarea codului prin introducerea de metode orientate pe utilizator *access control*.

Una din cele mai importante facilitati JAAS este implementarea frameworkului Pluggable Authentication Modules (PAM). Astfel, developer-ul poate scrie un modul standard de autentificare iar decizia legata de tehnologia de autentificare folosita este lasata la nivelul administratorului de sistem. Acest lucru se realizeaza prin implementarea tehnologiei de autentificare ca un LoginModule. Un LoginModule poate sa fie spcificat printr-un fisier de configurare dupa faza de deploy a aplicatiei. Astfel, se pot adauga tehnologii noi si se pot modifica politici de autentificare fara rescrierea unor parti din aplicatie.

Un utilizator (Subject) poate avea mai multe roluri (Principals). De exemplu, un Subject poate avea un Principal dat de cartea de identitate

si un alt Princial dat legitimatia de student, fiecare putind garanta diverse actiuni. Obiectele Principal nu sunt persistente. Unui obiect de tip Subject i se asociaza mai multe obiecte de tip Principal in urma unei autentificari reusite. Aceste obiecte nu mai sunt valabile daca utilizatorul se delogheaza.

In urma operatiei de login, se apeleaza un UsernameCallbackHandler si un PasswordCallbackHandler care daca satisfac conditiile impuse de LoginModule. Exemple de LoginModule includ un blind LoginModule care reuseste de fiecare data si nu necesita introducerea unei parole, un PasswordLoginModule care are nevoied e parola pentru autentificare sau un Kerberos pentru Single Signon. In functie de LoginModules, Subject-ul poate capata diferite Principals daca autentificarea s-a facut cu success.

Urmatorul pas este autorizarea. Daca codul Java rulat de utilizatorul care s-a autentificat, incearca accesarea unor resurse *sensitive* prin metoda *doAs*, fisierul care specifica politicile JAAS, poate specifica in functie de Principal, daca aceasta operatie poate sau nu continua.Pentru detalii despre cum se folosesc si se creeaza module de logjn, si exemple de cod este recomandata citirea tutorialului [9] despre JAAS realizat de SUN.

[9] http://java.sun.com/developer/technicalArticles/Security/jaasv2/index.html

A Certificat X.509 v3

```
Certificate:
 Data:
  Version: 3 (0x2)
  Serial Number: 2 (0x2)
  Signature Algorithm: md5WithRSAEncryption
  Issuer: O=Grid, OU=GlobusTest, OU=simpleCA, CN=Globus Simple CA
  Validity
   Not Before: May 17 22:09:43 2005 GMT
   Not After : May 17 22:09:43 2006 GMT
  Subject: O=Grid, OU=GlobusTest, OU=simpleCA, CN=Mihai Popescu
   Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
     RSA Public Key: (1024 bit)
      Modulus (1024 bit):
       00:d0:3a:f3:88:5c:98:5a:21:86:0c:55:9f:26:10:
       17:59:b5:6e:70:c5:c4:6b:bc:84:35:c7:36:07:0f:
       99:b8:d9:13:96:e7:67:db:63:1b:b3:bb:83:d7:02:
       cc:62:91:cc:cf:af:38:30:7f:cf:9b:ac:3c:7f:c1:
       c3:06:54:6d:92:b1:e7:86:4e:00:f6:f2:4f:0c:07:
       d8:aa:b0:75:73:8b:f0:20:3c:26:be:88:08:71:6a:
       ee:b2:de:06:96:af:84:fb:6b:d3:8b:56:02:7d:d0:
       cf:de:20:47:37:63:29:22:7a:5c:d8:73:44:47:73:
       59:9a:f2:7d:10:bc:60:b2:8f
       Exponent: 65537 (0x10001)
    X509v3 extensions:
     Netscape Cert Type:
     SSL Client, SSL Server, S/MIME, Object Signing
    Signature Algorithm: md5WithRSAEncryption
     44:39:e7:a5:af:b5:48:d1:4a:d5:a7:3a:9b:82:e4:35:cc:8b:
     d1:8b:aa:f1:85:56:a9:5c:b3:e7:da:0a:11:1d:2f:c2:0a:89:
     75:1f:d0:1d:7c:e0:d9:c7:40:04:15:11:a1:30:c7:0c:3f:96:
     7b:2c:0e:83:f7:a2:3e:fe:aa:b0:a0:e5:d3:78:df:f3:7c:ae:
     65:bd:42:a3:48:b5:4e:eb:d4:0e:b5:b7:7d:7e:b1:50:7d:ac:
     c1:ea:7d:cb:3f:ce:72:b5:9e:3d:00:47:5f:21:c3:a1:4b:de:
     a1:1e:e2:b1:d1:3c:b8:ad:7c:d9:9e:9f:4f:7e:8c:68:61:b0:
     e2:07
-----BEGIN CERTIFICATE-----
MIICQzCCAaygAwIBAgIBAjANBgkqhkiG9w0BAQQFADBbMQ0wCwYDVQQKEwRHcmlk
MRMwEQYDVQQLEwpHbG9idXNUZXN0MRowGAYDVQQLExFzaW1wbGVDQS1pb251Y29t
cDEZMBcGA1UEAxMQR2xvYnVzIFNpbXBsZSBDQTAeFw0wNTA1MTcyMjA5NDNaFw0w
NjA1MTcyMjA5NDNaMF0xDTALBgNVBAoTBEdyaWQxEzARBgNVBAsTCkdsb2J1c1Rl
c3QxGjAYBgNVBAsTEXNpbXBsZUNBLWlvbnVjb21wMRswGQYDVQQDExJJb251dCBD
b25zdGFuZGFjaGUwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANA684hcmFoh
hgxVnyYQF1m1bnDFxGu8hDXHNgcPmbjZE5bnZ9tjG7O7g9cCzGKRzM+vODB/z5us
PH/BwwZUbZKx54ZOAPbyTwwH2KqwdXOL8CA8Jr6ICHFq7rLeBpavhPtr04tWAn3Q
z94gRzdjKSJ6XNhzREdzWZryfRC8YLKPAgMBAAGjFTATMBEGCWCGSAGG+EIBAQQE
AwIE8DANBgkqhkiG9w0BAQQFAAOBgQBEOeelr7VIOUrVpzqbguQ1zIvRi6rxhVap
XLPn2goRHS/CCol1H9AdfODZx0AEFRGhMMcMP5Z7LA6D96I+/qqwoOXTeN/zfK5l
vUKjSLVO69QOtbd9frFQfazB6n3LP85ytZ49AEdfIcOhS96hHuKx0Ty4rXzZnp9P
foxoYbDiBw==
-----END CERTIFICATE-----
```

B Exemplu Password Based Encryption

```
import java.io.IOException;
import java.io.InputStream;
import java.io.PushbackInputStream;
import java.util.Arrays;
```

```
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.PBEParameterSpec;

public class PBenc
{
  /**
   * Reads user password from given input stream.
   */
  public static char[] readPasswd(InputStream in) throws IOException
  {
    char[] lineBuffer;
    char[] buf;
    int i;

    buf = lineBuffer = new char[128];

    int room = buf.length;
    int offset = 0;
    int c;

    loop: while (true)
    {
      switch (c = in.read())
      {
      case -1:
      case '\n':
        break loop;

      case '\r':
        int c2 = in.read();
        if ((c2 != '\n') && (c2 != -1))
        {
          if (!(in instanceof PushbackInputStream))
          {
            in = new PushbackInputStream(in);
          }
          ((PushbackInputStream) in).unread(c2);
        }
        else
          break loop;

      default:
        if (--room < 0)
        {
          buf = new char[offset + 128];
          room = buf.length - offset - 1;
          System.arraycopy(lineBuffer, 0, buf, 0, offset);
          Arrays.fill(lineBuffer, ' ');
          lineBuffer = buf;
        }
        buf[offset++] = (char) c;
        break;
      }
    }

    if (offset == 0)
    {
      return null;
    }
```

```
    char[] ret = new char[offset];
    System.arraycopy(buf, 0, ret, 0, offset);
    Arrays.fill(buf, ' ');

    return ret;
  }

  public static void main(String[] args)
  {
    PBEKeySpec pbeKeySpec;
    PBEParameterSpec pbeParamSpec;
    SecretKeyFactory keyFac;

    // Salt
    byte[] salt =
    { (byte) 0xc7, (byte) 0x73, (byte) 0x21, (byte) 0x8c, (byte) 0x7e, (byte) 0xc8,
        (byte) 0xee, (byte) 0x99 };

    // Iteration count
    int count = 20;

    // Create PBE parameter set
    pbeParamSpec = new PBEParameterSpec(salt, count);

    // Prompt user for encryption password.
    // Collect user password as char array (using the
    // "readPasswd" method from above), and convert
    // it into a SecretKey object, using a PBE key
    // factory.
    System.out.print("Enter encryption password:  ");
    System.out.flush();

    try
    {
      pbeKeySpec = new PBEKeySpec(readPasswd(System.in));
      keyFac = SecretKeyFactory.getInstance("PBEWithMD5AndDES");
      SecretKey pbeKey = keyFac.generateSecret(pbeKeySpec);

      // Create PBE Cipher
      Cipher pbeCipher = Cipher.getInstance("PBEWithMD5AndDES");

      // Initialize PBE Cipher with key and parameters
      pbeCipher.init(Cipher.ENCRYPT_MODE, pbeKey, pbeParamSpec);

      // Our cleartext
      byte[] cleartext = "plaintext".getBytes();

      // Encrypt the cleartext
      byte[] ciphertext = pbeCipher.doFinal(cleartext);
    }
    catch (Exception e)
    {
      System.out.println(e.getMessage());
    }
  }
}
```