ESYA API SmartCard

Kullanım Kılavuzu

versiyon 0.2

Kriptografik islemlerin güvenli bir ortamda yapılması amacıyla akıllı kartlara ihtiyaç duyulmaktadır. Akıllı kartlar özel anahtarın (private key) dısarıdan erisilmesine izin vermeyerek açık anahtar altyapısı için gerekli güvenligi saglarlar.

Akıllı kart içinde kullanıcının sertifikaları, özel anahtarları ve açık anahtarları bulunmaktadır. Her sertifikanın bir açık anahtarı ve bir özel anahtarı yine kart içinde yer almaktadır. Sertifikalar ve açık anahtarlar kart içinden okunabilmektedir. Özel anahtar ise dısarı kart dısına çıkartılamaz, anahtar ile kart içinde kriptografik islemler yapılabilir.

ESYA API SmartCard modülü akıllı kart ile islemlerine yardımcı olur, PKCS7 yapısında basit imza atabilir.

Tübitak BİLGEM

İçindekiler

1 Giriş

Kriptografik işlemlerin güvenli bir ortamda yapılması amacıyla akıllı kartlara ihtiyaç duyulmaktadır. Akıllı kartlar özel anahtarın (private key) dışarıdan erişilmesine izin vermeyerek açık anahtar altyapısı için gerekli güvenliği sağlarlar.

Akıllı kart içinde kullanıcının sertifikaları, özel anahtarları ve açık anahtarları bulunmaktadır. Her sertifikanın bir açık anahtarı ve bir özel anahtarı yine kart içinde yer almaktadır. Sertifikalar ve açık anahtarlar kart içinden okunabilmektedir. Özel anahtar ise dışarı kart dışına çıkartılamaz, anahtar ile kart içinde kriptografik işlemler yapılabilir.

ESYA API SmartCard modülü akıllı kart işlemlerine yardımcı olur, PKCS7 yapısında basit imza atabilir.

2 Gereksinimler

SmartCard API'si "ma3api-common-....jar" kütüphanesine ihtiyaç duymaktadır. Ayrıca kullanılacak akıllı kartın ve akıllı kart okuyucusunun sürücüsü sisteme kurulmuş olması gerekmektedir.

Java 6 ile birlikte akıllı karta direk kart sürücüsü olmadan komut gönderilmekte, kart ile ilgili bilgiler karttan okunabilmektedir. Bu farklılıktan dolayı ESYA API'de akıllı kart işlemleri için iki farklı "jar" vardır. Bu iki jar'ın fonksiyon arayüzleri aynıdır; sadece java 5 kütüphanesi bazı fonksyanaliteleri desteklememektedir. Eğer Java 6 kullanabiliyorsanız "ma3api-smartcard-j6-....jar"ını, java 5 kullanmak zorundaysanız da "ma3api-smartcard-j6-....jar"ını edininiz.

3 Akıllı Karta Erişim

SmartCard sınıfı akıllı kart ile ilgili işlemlerden sorumlu sınıftır. *SmartCard* sınfının çalıştırılabilmesi için hangi akıllı kartın kullanıldığının bilinmesi gerekmektedir. Çünkü akıllı kart işlemleri akıllı kartın sürücüsünün üzerinden yapılmaktadır. Java 5 ile Java 6 arasındaki temel fark bu noktada çıkmaktadır. Java 6 ile java kütüphaneleri kullanılarak kart bilgilerine erişilip hangi kart olduğu belirlenebilmektedir.

Java 6 için *SmartOp* sınıfının *findCardTypeAndSlot()* fonksiyonu ile kartın slot numarası ve kart tipi bulunabilmektedir. Eğer bilgisayara bir akıllı kart takılı ise fonksiyon doğrudan bu kartın bilgilerini dönecektir. Eğer birden fazla akıllı kart takılı ise *javax.swing . JOptionPane* ile kullanıcıya akıllı kartlardan biri seçtirilecektir.

Java6:

```
Pair<Long, CardType> slotAndCardType =  SmartOp.findCardTypeAndSlot();
Long slot = slotAndCardType.getObject1();
SmartCard smartCard = new SmartCard(slotAndCardType.getObject2());
long session = smartCard.openSession(slot);
```

Eğer görsel bir arayüzün API tarafından gösterilmesini istemiyorsanız; *SmartOp*

sınıfının *getCardTerminals()* fonksiyonu ile akıllı kart okuyucularının isimlerini alabilirsiniz. Bu isimler ile kartı kullanıcıya seçtirdikten sonra *getSlotAndCardType(String terminal)* fonksiyonuyla kullanıcının seçtiği kartın slot numarası ve kart tipi alınabilir.

Eğer kullanıcıya kart tipine göre akıllı kartı seçtirmek isteniyorsa, *SmartOp* sınıfının *findCardTypesAndSlots()* ile bağlı olan bütün kartların slot numaralarını ve kart tiplerini alabilirsiniz.

Java 5'te ise akıllı kartın türü bilinmelidir. Aşağıdaki örnek kodda herhangi bir seçim yapılmadan birinci karta oturum açılmaktadır. *SmartCard* sınıfının *getSlotInfo(Long slot)* foksiyonu ile slot hakkındaki bilgiler edinelebilir, burdaki slot tanımlamasıyla kullanıcıdan kart seçmesi istenebilir.

Akıllı kart ile işlem yapmaya başlamak için *openSession()* fonksiyonu ile oturum açılmalıdır. Karttan sertifika okumak için *login* olmaya gerek yoktur. Yalnız imza çözme veya şifreleme işlemi yapılacaksa karta *login* olunmalıdır.

```
SmartCard sc = new SmartCard(CardType.AKIS);
long [] slots = sc.getSlotList();
//sc.getSlotInfo(slots[0]).slotDescription;
long session = sc.openSession(slots[0]);
sc.login(session, "12345");
```

3.1 Akis Kartlara Erişim

Akis kartlara Java 6 kullanildiginda Akis'in java kütüphanesi kullanilarak komut (APDU) gönderilebilmektedir. Sistemde akis sürücüsü yüklü olmasa bile AkisCIF.x.x.x.jar oldugunda karta erisilmektedir. AkisCIF üzerinden akilli karta erismek akilli kart islemlerinin süresini dolayisiyla imza süresini kisaltmaktadir. Yalniz AkisCIF üzerinden karta erisildiginde diger programlar tarafindan karta erisilememektedir. Ayni zamanda kullandiginiz AkisCIF, Akis'in bir yan ürünü oldugundan gelecekte her karti desteklemeyebilir. Yeni bir AkisCIF sürümü çiktiginda projenize entegre etmeniz gerekecektir.

Akilli karta APDU komutlari ile AkisCIF.x.x.x.jar üzerinden erisilmesinden APDUSmartCard sinifi sorumludur. Örnek bir kullanim asagidaki gibidir.

```
APDUSmartCard sc = new APDUSmartCard();
long [] slots = sc.getSlotList();
sc.openSession(slots[0]);
List<byte []> certs = sc.getSignatureCertificates();
CertificateFactory cf = CertificateFactory.getInstance("X.509");
X509Certificate cert = (X509Certificate)cf.generateCertificate(new
ByteArrayInputStream(certs.get(0)));

BaseSigner signer = sc.getSigner(cert, Algorithms.SIGNATURE_RSA_SHA1);
```

Yukarida da belirtildigi gibi AkisCIF arayüzü bütün kartlari desteklemeyebilir. AkisCIF desteklendiginde AkisCIF ile desteklenmediginde dll ile islemlerinizi yapmak için asagidaki sekilde kullanabilirsiniz.

```
BaseSmartCard bsc = null;
int index = 0;
String [] terminals = SmartOp.getCardTerminals();
String selectedTerminal = terminals[index];
long slot;
if(APDUSmartCard.isSupported(selectedTerminal))
{
      bsc = new APDUSmartCard();
      slot = index + 1;
}
else
{
      Pair<Long, CardType> slotAndCardType = SmartOp.getSlotAndCardType
(selectedTerminal);
      slot = slotAndCardType.getObject1();
      bsc = new P11SmartCard(slotAndCardType.getObject2());
}
bsc.openSession(slot);
```

4 Akıllı Kartan Sertifikanın Okunması

Akıllı karttan sertifika *SmartCard* sınıfının *getSignatureCertificates()* veya *getEncryptionCertificates()* fonksiyonları ile okunabilir. Eğer imzalama işlemi yapılacaksa *getSignatureCertificates()* fonksiyonu, şifreleme işlemi yapılacaksa *getEncryptionCertificates ()* fonksiyonu kullanılmalıdır.

Bu fonksiyonlar sertifikaların kodlanmış hallerini *byte []* olarak dönerler. Eğer ESYA API asn modülünü (ma3api-asn-....jar) kullanabiliyorsanız, kartan aldığınız byte değerlerini anlamlı hale getirmek için *ECertificate* sınıfını kullanabilirsiniz.

Atılacak imzanın kanuni hükümlülüklerinin olması için imzalamada kullanılan sertifikanın nitelikli olması gerekmektedir. Bu kontrol *ECertificate* sınıfının *isQualifiedCertificate ()* fonksiyonu ile yapılabilir.

ECertificate sınıfının *getSubject().stringValue()* fonksiyonu ile sertifikalar birbirinden ayırt edilebilir. Kullanıcı bu bilgi ile seçim yapabilir.

Ayrıca *ECertificate* sınıfının *getSubject().getCommonNameAttribute()* fonksiyonu sertifika sahibinin ismini dönmektedir. Karttaki sertikaların isim bilgisi hepsi için aynı olacağından, karttaki sertifikaları ayırt etmek amacıyla kullanılamaz. Kimin imzayı attığını göstermek için kullanılabilir.

Aşağıdaki kod bloğu akıllı kart içinden imzalama sertifikalarını alıp nitelikli olanların *Subject* alanını ekrana basılmaktadır.

```
List<byte []> certs = smartCard.getSignatureCertificates(session);
for (byte[] bs : certs) {
      ECertificate cert = new ECertificate(bs);
      if(cert.isQualifiedCertificate())
```

```
            System.out.println(cert.getSubject().stringValue());
}
```

Eğer ESYA API asn sınıflarına erişim yoksa, *ECertificate* yerine java'nın *x509Certificate* sınıfı kullanılabilir. *ECertificate* sınıfının *isQualifiedCertificate()* fonksiyonu yerine aşağıdaki örnek kodda gösterildiği gibi kontrol yapılabilir. Sertifikaları birbirinden ayırt etmek amacıyla *x509Certificate* sınıfının *getSubjectDN().toString()* metodu kullanılabilir.

Aşağıdaki kod bloğu akıllı kart içinden imzalama sertifikalarını alıp nitelikli olanların *Subject* alanını ekrana basmaktadır.

```
List<byte []> certs = smartCard.getSignatureCertificates(session);
CertificateFactory cf = CertificateFactory.getInstance("X.509");
String qcStatement = "1.3.6.1.5.5.7.1.3";
for (byte[] bs : certs) {
      X509Certificate cert = (X509Certificate)cf.generateCertificate(new
ByteArrayInputStream(bs));
      if( cert.getExtensionValue(qcStatement) != null)
            System.out.println(cert.getSubjectDN().toString());
}
```

5 Akıllı Karttaki Nesne Adlarının Okunması

Akıllı kartta bulunan sertifika, açık anahtar ve özel anahtarın her biri nesne olarak adlandırılır. Akıllı karttaki nesnelerin adı ile de işlem yapılabilir. Nesne adları değişken olabileceğinden nesne adları ile işlem yapmak önerilmez. Yalnız bazı durumlarda nesne adları kullanıcıya daha anlamlı gelebilir.

SmartCard sınıfının *getSignatureKeyLabels(...) ve getEncryptionKeyLabels(...)* fonksiyonları ile anahtarların adları okunabilir. Eğer anahtarın sertifikasının adı, anahtar adı ile aynı ise bu ad ile sertifika da okunabiir. Sertifikanın okunması için *readCertificate(long aSessionID,String aLabel)* fonksiyonu kullanılabilir.

6 Akıllı Karta İmzalama-Şifreleme İşlemlerinin Yapılması

Akıllı kartta şifreleme ve imzalama işlemlerinin yapılması için *login* olunması gerekmektedir. *SmartCard* sınıfının *decryptDataWithCertSerialNo(...),decryptData(...), signDataWithCertSerialNo(...), signData(...)* fonksiyonları kriptografik işlemleri yerine getirmek için kullanılabilir. Akıllı kart ile yapılacak işlemler, özel anahtar (private key) ile yapılacak işlemler olmalıdır. Açık anahtar ile yapılan işlemlerin herhangi bir güvenlik kısıtı olmadığından akıllı kartta yapılmasının anlamı yok. Özel anahtar kullanan işlemler ise imza atma ve şifrelenmiş verinin şifresinin çözülmesi işlemidir.

Yalnız imzalama ve şifeleme işlemlerini kullanan modüller *BaseSigner* veya *BaseCipher* arayüzünde imzacılar ve şifreleyiciler istemektedir. Bu yüzden *SCSignerWithCertSerialNo, SCSignerWithKeyLabel, SCCipherWithCertSerialNo, SCCipherWithKeyLabel* sınıfları daha çok kullanılacaktır.

Aşağıdaki örnek kodda sertifika seri numarası ile işlem yapan sınıflar vardır.

```
SCSignerWithCertSerialNo signer = new SCSignerWithCertSerialNo(sc,
session, slot,
            signatureCert.getSerialNumber().toByteArray(), Algorithms.
SIGNATURE_RSA_SHA1);

SCCipherWithCertSerialNo cipher = new SCCipherWithCertSerialNo(sc,
session,
            encCert.getSerialNumber().toByteArray());
```

Aşağıdaki örnek kodda anahtar adı ile işlem yapan sınıflar vardır.

```
SCSignerWithKeyLabel signer = new SCSignerWithKeyLabel(sc, session, slot,
"yasemin.akturk#SIGN0",
            Algorithms.SIGNATURE_RSA_SHA1);

SCCipherWithKeyLabel cipher = new SCCipherWithKeyLabel(sc, session, slot,
"yasemin.akturk#ENCR0");
```

7 PKCS7 Yapısında İmza Atılması

PKCS7 yapısı en basit imza yapılarından biridir. *PKCS7Signature* sınıfı PKCS7 formatında imza atılmasından sorumlu sınıftır. Ayrık imza veya bütünleşik imza atılabilir. *signExternalContent* fonksiyonu ile ayrık imza, *signInternalContent* fonksiyonu ile bütünleşik imza atılabilir.

Aşağıdaki örnek kodda PKCS7 yapısında imzanın nasıl atılacağı gösterilmiştir. Örnekte ayrık imza atılmıştır. *signInternalContent* fonksiyonu kullanılırsa bütünleşik imza atılacaktır. Bütünleşik imzadan içerik *PKCS7* nesnesinin *getContentInfo().getContentBytes()* fonksiyonu ile alınabilir.

```
public void testPKCS7() throws Exception
{
      PKCS7Signature pkcsSignature = new PKCS7Signature();
      ByteArrayOutputStream signature = new ByteArrayOutputStream();

      SmartCard sc = new SmartCard(CardType.AKIS);
      long [] slots = sc.getSlotList();
      //sc.getSlotInfo(slots[0]).slotDescription;
      long session = sc.openSession(slots[0]);
      sc.login(session, "12345");

      //Gets first certificate, it must be asked to user if it is more
than one certificate.
      byte [] certBytes = sc.getSignatureCertificates(session).get(0);
      CertificateFactory cf = CertificateFactory.getInstance("X.509");
      X509Certificate cert = (X509Certificate)cf.generateCertificate(new
```

```
ByteArrayInputStream(certBytes));

      BaseSigner signer = new SCSignerWithCertSerialNo(sc, session, slots
[0]
                              ,cert.getSerialNumber().toByteArray(),
Algorithms.SIGNATURE_RSA_SHA1);

      ByteArrayInputStream bais = new ByteArrayInputStream(toBeSigned);
      pkcsSignature.signExternalContent(bais, cert, signature, signer);

      Assert.assertEquals(true, validate(new ByteArrayInputStream
(signature.toByteArray()), cert));
}

//For this case, there is one signature
private boolean validate(InputStream signature, X509Certificate cert)
throws Exception
{
      PKCS7 p = new PKCS7(signature);
      //validates the signature, not the person.
      SignerInfo  [] signerInfo = p.verify(toBeSigned);
      if(signerInfo == null)
            return false;
      else
      {
            //Checks whether the expected person signed the data.
            return signerInfo[0].getCertificateSerialNumber().equals(cert.
getSerialNumber()) == true;
      }
}

private final static byte [] toBeSigned = "Test".getBytes();
```

8 Lisans Ayarları

LicenseUtil. setLicenseXml(InputStream ...) fonksiyonu ile lisans dosyası verilebilir. Eğer bu fonksiyon ile bir lisans bilgisi verilmemişse, 'working directory' altında lisans klasörü altında "lisans.xml" dosyası aranır.

Lisans dosyasının satış yapılan müşteri dışında başkasının eline geçmemesi için bu dosyanın sunucuda tutulması, istemci yazılımlarının lisansa sunucudan erişmesi tavsiye edilmektedir.

8.1 Deneme Lisansı ile Çalışma

ESYA API ile denemeler yapmanız için test lisansı edinebilirsiniz. Lisans için verilen xml dosyasında, test alınının değeri "test" ise deneme lisansına sahipsinizdir.

Bu lisans ile yaptığımız çalışmalarda ancak "common name" alanında "test" metni

geçen sertifikalar kullanabilirsiniz. Ayrıca akıllı kart ile yapacağınız işlemlerde birkaç saniye gecikme olacaktır.