ESYA API CMS Signature

Kullanım Kılavuzu

Tübitak BİLGEM

İçindekiler

Bölüm 1 Sozluk 1

Bölüm 2 Giriş 1

Bölüm 3 İmza Atma İşlemleri 1

1 İmzasız Bir Verinin İmzalanması ... 1
2 İmzalı Bir Veriye İmza Eklenmesi ... 3
Paralel İmza Ekleme ... 3
Seri İmza Ekleme ... 5
3 Ayrık İmza Atılması ... 7
4 Zorunlu Olmayan Özelliklerin Eklenmesi ... 9
5 Sertifika Doğrulaması ... 11
6 Akıllı Kart İşlemleri ... 11
SmartCardManager ... 12
Slot İşlemleri ... 13
Karttan Sertifika Okuma ... 14
Karttan Anahtar Etiketlerinin Okunması ... 15
Akıllı Kart İmzacılar Oluşturma ... 15

Bölüm 4 İmza Doğrulama İşlemleri 16

1 İmza Doğrulama Sonucu ... 17
2 Ön Doğrulama ... 18
3 Ayrık İmzanın Doğrulanması ... 19
4 Sertifika Doğrulama ... 20
Politika Dosyası ... 20
Bulucular ... 21
Eşleştiriciler ... 22
Doğrulayıcılar ... 22
Sertifika Deposu ... 23
5 İmzacıların Alınması ... 23

Bölüm 5 İmza Tipleri 24

1 BES ... 24
2 EST ... 26
3 ESXLong ... 27
4 ESA ... 27

Bölüm 6 İmza Tipleri Arasında Dönüşüm 28

Bölüm 7 Zaman Damgası 29

1 İmza Zamanı Alma ... 30
2 Zaman Damgası Sunucusunun Test Edilmesi ... 31
3 Zaman Damgası Alma ... 32

Bölüm 8 Parametreler 32

Bölüm 9 Lisans Ayarları 34

1 Deneme Lisansı ile Çalışma 34

Bölüm 10 Döküman Versiyonları 34

1 Sozluk

ÇiSDUP: Çevrimiçi Sertifika İptal Kontrolü (OCSP)
SİL : Sertifika İptal Listesi (CRL)
yürürlükteki dizin : Programınızın koştuğu dizin (working directory)
bağıl adres : Belirli bir adres parametre alınarak verilen adres (relative path)

2 Giriş

Bu doküman ESYA API CMS Signature kütüphanesinin nasıl kullanılacağı hakkında bilgi vermektedir. CMS Signature kütüphanesi ile akıllı kartları yönetebilir, sertifika doğrulayabilir, elektronik imza atabilir ve elektronik imza doğrulayabilirsiniz. Kütüphaneyi kullanabilmeniz için adınıza üretilmiş bir lisansa veya test lisansına ihtiyacınız vardır.

İmza atma işlemi için kullanıcının sertifikaya ve özel anahtarını güvenli bir şekilde muhafaza edebileceği bir öğeye ihtiyacı vardır. Özel anahtarların muhafazası için genel olarak akıllı kart kullanıldığından, dokümanda bu öğe için "akıllı kart" kavramı kullanılacaktır. İmza atma hakkında ayrıntılı bilgi için İmza Atma İşlemleri 1 bölümüne bakınız.

İmza doğrulama işlemleri sertifikanın doğrulanmasından ve imzanın yapısal olarak doğrulamasından oluşmaktadır. Sertifika doğrulama için sertifika deposuna ve sertifika doğrulamanın nasıl yapılacağını belirten politika dosyasına ihtiyaç vardır. Daha ayrıntılı bilgi için İmza Doğrulama İşlemleri 16 bölümüne bakınız.

3 İmza Atma İşlemleri

İmzalama işlemi genel olarak iki aşamada gerçekleşmektedir. Öncelikle imzacının sertifikasının doğrulanması yapılmaktadır, sonra imza atma işlemi gerçekleşmektedir.

3.1 İmzasız Bir Verinin İmzalanması

Veriyi imzalama işleminden *BaseSignedData* sınıfı sorumludur. Bu sınıfa öncelikle *addContent(…)* fonksiyonu ile imzalanacak veri eklenmelidir. *AddContent(…)* fonksiyonu yalnızca bir kere çağrılmalıdır. İmzalanacak veri *addContent(…)* ile eklendikten sonra değiştirilemez. *AddSigner(…)* fonksiyonu ile veriye imza bilgileri eklenir.

İmza eklenirken imzanın türü, imzacının sertifikası, imza işlemini gerçekleştirecek kripto nesnesi, varsa ekstra imza özellikleri ve imza üretiminde kullanılması gereken parametreler *addSigner(…)* fonksiyonuna parametre olarak geçilmelidir. İmza atan örnek kod bloğu:

Java:

```
BaseSignedData bs = new BaseSignedData();
bs.addContent(new SignableByteArray("test".getBytes()));
```

```
//create parameters necessary for signature creation
HashMap<String, Object> params = new HashMap<String, Object>();
ValidationPolicy policy=  PolicyReader.readValidationPolicy(new
FileInputStream(POLICY_FILE));
/*necessary for certificate validation.By default,certificate validation
is done.But if the user does not want certificate validation,he can add
P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to
false*/
params.put(EParameters.P_CERT_VALIDATION_POLICY, policy);
/*By default, QC statement is checked,and signature wont be created if it
is not a qualified certificate.  By setting this parameter to false,user
can use test certificates*/
params.put(EParameters.P_CHECK_QC_STATEMENT, false);

ECertificate cert = new ECertificate(new File(SIGNING_CERTIFICATE_PATH));


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

BaseSigner signer = new SCSignerWithCertSerialNo(sc,session,slots[0],
cert.getSerialNumber().toByteArray(),
            SignatureAlg.RSA_SHA1.getName());

            /*add signer.  Since the specified attributes are mandatory for
bes,null is given as parameter for optional attributes*/
bs.addSigner(ESignatureType.TYPE_BES, cert , signer, null, params);

            //write the contentinfo to file
AsnIO.dosyayaz(bs.getEncoded(),SIGNATURE_FILE);
sc.logout(session);
sc.closeSession(session);
```

C#:

```
BaseSignedData bs = new BaseSignedData();
bs.addContent(new SignableByteArray(ASCIIEncoding.ASCII.GetBytes
("test")));

//create parameters necessary for signature creation
Dictionary<String, Object> parameters = new Dictionary<String, Object>();
ValidationPolicy policy;
using (FileStream fs = new FileStream(@"C:\policy.xml", FileMode.Open,
FileAccess.Read))
{
     policy = PolicyReader.readValidationPolicy(fs);
}
/*necessary for certificate validation.By default,certificate validation
is done.But if the user does not want certificate validation,he can add
P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to
false*/
```

```
parameters[EParameters.P_CERT_VALIDATION_POLICY] = policy;
/*By default, QC statement is checked,and signature wont be created if it
is not a qualified certificate. By setting this parameter to false,user
can use test certificates*/
parameters[EParameters.P_CHECK_QC_STATEMENT] = false;

ECertificate cert = new ECertificate(new FileInfo
(SIGNING_CERTIFICATE_PATH));

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

Pair<SignatureAlg,IAlgorithmParams> algParams = SignatureAlg.
fromAlgorithmIdentifier(cert.getSignatureAlgorithm());
            BaseSigner signer = new SCSignerWithCertSerialNo(sc, session,
slots[0], cert.getSerialNumber().GetData(), algParams.first().getName());


//add signer. Since the specified attributes are mandatory for bes,null is
given as parameter for optional attributes
bs.addSigner(ESignatureType.TYPE_BES, cert, signer, null, parameters);

//write the contentinfo to file
AsnIO.dosyayaz(bs.getEncoded(), SIGNATURE_FILE);
sc.logout(session);
sc.closeSession(session);
```

3.2 İmzalı Bir Veriye İmza Eklenmesi

Bir veri birkaç kişi tarafından imzalanabilir. İmzalar iki şekilde atılabilir.

Paralel İmza Ekleme
Seri İmza Ekleme

3.2.1 Paralel İmza Ekleme

Bu tür imzalarda bütün imzacıların imzaladıkları veri aynı veridir. Bütün imzalar aynı seviyededir. Bir imzacının imzası dokümandan çıkartılırsa fark edilemez.

Java:

```
byte [] signature = AsnIO.dosyadanOKU(SIGNATURE_FILE);
BaseSignedData bs = new BaseSignedData(signature);

//create parameters necessary for signature creation
HashMap<String, Object> params = new HashMap<String, Object>();
ValidationPolicy policy =  PolicyReader.readValidationPolicy(new
FileInputStream(POLICY_FILE));
```

```
/*necessary for certificate validation. By default, certificate validation
is done. But if the user does not want certificate validation, he can add
P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to
false*/
params.put( EParameters.P_CERT_VALIDATION_POLICY, policy);
/*By default, QC statement is checked, and signature wont be created if it
is not a qualified certificate. By setting this parameter to false, user
can use test certificates*/
params.put( EParameters.P_CHECK_QC_STATEMENT, false);

ECertificate cert = new ECertificate( new File( SIGNING_CERTIFICATE_PATH));


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

BaseSigner signer = new SCSignerWithCertSerialNo ( sc, session, slots[0],
cert.getSerialNumber().toByteArray
          (), SignatureAlg.RSA_SHA1.getName());

//add signer. Since the specified attributes are mandatory for bes, null is
given as parameter for
//optional attributes
bs.addSigner( ESignatureType.TYPE_BES, cert , signer, null, params);

//write the contentinfo to file
AsnIO.dosyayaz( bs.getEncoded(), NEW_SIGNATURE_ADDED_FILE);
sc.logout( session);
sc.closeSession( session);
```

C#:

```
//Load signed document
BaseSignedData bs = new BaseSignedData( SIGNED_DATA);

//create parameters necessary for signature creation
Dictionary<String, Object> parameters = new Dictionary<String, Object>();
ValidationPolicy policy;

using ( FileStream fs = new FileStream( @"C:\policy.xml", FileMode.Open,
FileAccess.Read))
{
     policy = PolicyReader.readValidationPolicy( fs);
}

/*necessary for certificate validation. By default, certificate validation
is done. But if the user does not want certificate validation, he can add
P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to
false*/
parameters[ EParameters.P_CERT_VALIDATION_POLICY] = policy;
/*By default, QC statement is checked, and signature wont be created if it
```

```
is not a qualified certificate. By setting this parameter to false,user
can use test certificates*/
parameters[EParameters.P_CHECK_QC_STATEMENT] = false;

ECertificate cert = new ECertificate(new File(SIGNING_CERTIFICATE_PATH));

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

Pair<SignatureAlg,IAlgorithmParams> algParams = SignatureAlg.
fromAlgorithmIdentifier(cert.getSignatureAlgorithm());
            BaseSigner signer = new SCSignerWithCertSerialNo(sc, session,
slots[0], cert.getSerialNumber().GetData(), algParams.first().getName());
//add signer. Since the specified attributes are mandatory for bes,null is
given as parameter for optional attributes
bs.addSigner(ESignatureType.TYPE_BES, cert, signer, null, parameters);

//write the contentinfo to file
AsnIO.dosyayaz(bs.getEncoded(), NEW_SIGNATURE_ADDED_FILE);
sc.logout(session);
sc.closeSession(session);
```

3.2.2 Seri İmza Ekleme

Seri imza eklerken, imzanın eklendiği seviyeye kadar olan bütün imzacıların imzası ve imzalanmak istenen veri imzalanır. Dolayısıyla bir imzacının imzası çıkartılırsa o imzacıdan sonra imza atan imzacıların da imzalarının çıkartılması gerekmektedir.

Aşağıdaki kod örneğinde ilk imzacıya seri imza ekleniyor.

Java:

```
byte [] signature = AsnIO.dosyadanOKU(SIGNATURE_FILE);
BaseSignedData bs = new BaseSignedData(signature);

//create parameters necessary for signature creation
HashMap<String, Object> params = new HashMap<String, Object>();
ValidationPolicy policy =  PolicyReader.readValidationPolicy(new
FileInputStream(POLICY_FILE));
/*necessary for certificate validation. By default,certificate validation
is done. But if the user does not want certificate validation,he can add
P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to
false*/
params.put(EParameters.P_CERT_VALIDATION_POLICY, policy);
/*By default, QC statement is checked,and signature wont be created if it
is not a qualified certificate. By setting this parameter to false,user
can use test certificates*/
params.put(EParameters.P_CHECK_QC_STATEMENT, false);
```

```
ECertificate cert = new ECertificate( new File( SIGNING_CERTIFICATE_PATH));


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

BaseSigner signer = new SCSignerWithCertSerialNo ( sc, session, slots[ 0],
cert. getSerialNumber().
          toByteArray(), SignatureAlg. RSA_SHA1. getName());

Signer firstSigner = bs. getSignerList(). get( 0);

firstSigner. addCounterSigner( ESignatureType. TYPE_BES, cert , signer, null,
params);

//write the contentinfo to file
AsnIO. dosyayaz( bs. getEncoded(), NEW_SIGNATURE_ADDED_FILE);
sc. logout( session);
sc. closeSession( session);
```

C#:

```
//Load signed document
BaseSignedData bs = new BaseSignedData( SIGNED_DATA);
//create parameters necessary for signature creation
Dictionary<String, Object> parameters = new Dictionary<String, Object>();
ValidationPolicy policy;
using ( FileStream fs = new FileStream( @"C: \policy. xml", FileMode. Open,
FileAccess. Read))
{
     policy = PolicyReader. readValidationPolicy( fs);
}
/*necessary for certificate validation. By default, certificate validation
is done. But if the user does not want certificate validation, he can add
P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to
false*/
parameters[ EParameters. P_CERT_VALIDATION_POLICY] = policy;
/*By default, QC statement is checked, and signature wont be created if it
is not a qualified certificate. By setting this parameter to false, user
can use test certificates*/
parameters[ EParameters. P_CHECK_QC_STATEMENT] = false;

ECertificate cert = new ECertificate( new FileInfo
( SIGNING_CERTIFICATE_PATH));

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

Pair<SignatureAlg, IAlgorithmParams> algParams= SignatureAlg.
```

```
fromAlgorithmIdentifier(cert.getSignatureAlgorithm());
            BaseSigner signer = new SCSignerWithCertSerialNo(sc, session,
slots[0], cert.getSerialNumber().GetData(), algParams.first().getName());

Signer firstSigner = bs.getSignerList()[0];
firstSigner.addCounterSigner(ESignatureType.TYPE_BES, cert, signer, null,
parameters);

//write the contentinfo to file
AsnIO.dosyayaz(bs.getEncoded(), NEW_SIGNATURE_ADDED_FILE);
sc.logout(session);
sc.closeSession(session);
```

Örnek kodlar içerisinde (*SerialSign.java*) bir veriye hiç imzacı eklenmediyse ilk imzayı ekleyen, sonra eklenen imzaları ise son imzacıya seri olarak ekleyen örneği bulabilirsiniz.

3.3 Ayrık İmza Atılması

Ayrık imzada imzalanacak veri BaseSignedData.addContent(…) fonksiyonunun ikinci parametresi false verilerek atanır.

Dahili imza ile büyük boyutlu dosyalar imzalanamaz. İmzanın yapısı gereği imzalanacak verinin hepsi belleğe alınmaktadır. Bundan dolayı büyük boyutlu dosyaların imzalanması için ayrık imza kullanmak gerekmektedir.

Java:

```
BaseSignedData bs = new BaseSignedData();

File file = new File(MOVIE_FILE);
ISignable signable = new SignableFile(file,32768);
bs.addContent(signable,false);

//create parameters necessary for signature creation
HashMap<String, Object> params = new HashMap<String, Object>();

ValidationPolicy policy =  PolicyReader.readValidationPolicy(new FileInputStream(POLICY_FILE));
params.put(EParameters.P_CERT_VALIDATION_POLICY, policy);
/*By default, QC statement is checked,and signature wont be created if it is not a qualified certificate.
By setting this parameter to false,user can use test certificates*/
params.put(EParameters.P_CHECK_QC_STATEMENT, false);

ECertificate cert = new ECertificate(new File(SIGNING_CERTIFICATE_PATH));

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

BaseSigner signer = new SCSignerWithCertSerialNo (sc, session, slots[0],      cert.getSerialNumber().
    toByteArray(), SignatureAlg.RSA_SHA1.getName());
```

```
bs.addSigner(ESignatureType.TYPE_BES, cert, signer, null, params);

AsnIO.dosyayaz(bs.getEncoded(), SIGNATURE_FILE);

sc.logout(session);
sc.closeSession(session);
```

C#:

```
BaseSignedData bs = new BaseSignedData();

FileInfo file = new FileInfo( MOVIE_FILE);
ISignable signable = new SignableFile( file, 32768);
bs. addContent( signable, false);

//create parameters necessary for signature creation
Dictionary<String, Object> params_ = new Dictionary<String, Object>();

ValidationPolicy policy;
using ( FileStream fs = new FileStream( POLICY_FILE, FileMode. Open,
FileAccess. Read))
{
   policy = PolicyReader. readValidationPolicy( fs);
}
params_[ EParameters. P_CERT_VALIDATION_POLICY] = policy;
/*By default, QC statement is checked,and signature wont be created if it
is not a qualified certificate.
By setting this parameter to false,user can use test certificates*/
params_[ EParameters. P_CHECK_QC_STATEMENT] = false;

ECertificate cert = new ECertificate( new FileInfo
( SIGNING_CERTIFICATE_PATH));

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

BaseSigner signer = new SCSignerWithCertSerialNo( sc, session, slots[ 0],
cert. getSerialNumber(). GetData(), SignatureAlg. RSA_SHA1. getName());

bs. addSigner( ESignatureType. TYPE_BES, cert, signer, null, params_);

AsnIO. dosyayaz( bs. getEncoded(), SIGNATURE_FILE);

sc. logout( session);
sc. closeSession( session);
```

3.4 Zorunlu Olmayan Özelliklerin Eklenmesi

İmzaya adres bilgisi, imza zamanı bilgisi gibi opsiyonel bilgileri özellik olarak ekleyebilirsiniz. API'de tanımlı olan kullanabileceğiniz özellikler;

SigningTimeAttr; beyan edilen imza zamanını içerir. Ancak beyan edilen bu tarih güvenilir bir tarih olmadığından imzanın bu tarihte atıldığını garanti etmez, bilgi amaçlı kullanılabilir.

SignerLocationAttr*;* imzacının adresi hakkında bilgiler içerir. İmzacının ülkesi, şehri ve posta adresi belirtilebilir. Bilgi amaçlı olduğundan bu bilgilerden bazılarının değeri *null* olabilir.

CommitmentTypeIndicationAttr; imza amacını belirtmek için kullanılabilir. İmzalanan verinin tarafınızdan oluşturulmuş olduğunu, sadece imzanın içeriğini onayladığınızı vs. belirtebilirsiniz. *CommitmentType* sınıfında tanımlanmış aşağıdaki değerler verilebilir.

RECEIPT, imza sahibinin imzalı belgeyi aldığını (bir yerden geliyor ise) belirtmek için kullanılır.
SENDER, imzalı veriyi gönderenin (imzalı veri bir yere gönderiliyor ise) veriyi gönderen kişi olduğunu belirtmek için kullanılır. Yani imza sahibinin gönderilen verinin içeriğini onayladığı anlamına gelmez sadece bunu ben gönderdim demektir.
APPROVAL, imza sahibinin belgenin içeriğini onayladığını belirtmek için kullanılır.
DELIVERY, bir mesaj gönderildiğinde, bu mesajın karşı tarafa iletildiğini belirtmede kullanılır. Bu tür bir imza genelde güvenilir servis sağlayıcılar (TSP - Trusted Service Provider) tarafından kullanılır.
CREATION, imza sahibinin belgeyi oluşturan kişi olduğunu belirtmek için kullanılır. Belge içeriğini onayladığı veya gönderdiği anlamına gelmez.
ORIGIN, imza sahibinin belgeyi oluşturduğunu, içeriğini onayladığını ve gönderenin de kendisi olduğunu belirtmek için kullanılır.

ContentIdentifierAttr; imzalanan içeriği tanımlamak için kullanılır. Özellikle ayrık imzada imzalanan dökümanı imza ile eşleştrimek için kullanılabilir. *byte []* olacak şekilde herhangi bir değer olabilir.

ContentHintsAttr; imzalanan içerik hakkında alıcıya fikir vermek amacıyla kullanılır.

SignerAttributesAttr; imzalayan kişi hakkında bilgiler içerir. İmzalayanın iddia ettiği özellikleri veya imzalayanın yetki sertifikasını barındırabilir.

İmza atma sırasında imzaya eklenmek istenen özellikler bir listeye konularak kütüphaneye verilir. Eklenecek alanın değeri alanın yaratılması sırasında kurucu fonksiyona verilir. Aşağıdaki örnekteki gibi imzaya eklenirler ve imzadan okunurlar.

Java

```
List<IAttribute> optionalAttributes = new ArrayList<IAttribute>();
optionalAttributes.add( new SigningTimeAttr(Calendar.getInstance()));
optionalAttributes.add( new SignerLocationAttr("TURKEY", "KOCAELİ",
                          new String[]{"TUBITAK UEKAE","GEBZE"}));
optionalAttributes.add( new CommitmentTypeIndicationAttr(CommitmentType.
```

```
CREATION));
optionalAttributes.add( new ContentIdentifierAttr("PL123456789".getBytes(
"ASCII")));

bs.addSigner( ESignatureType.TYPE_BES, cert , signer, optionalAttributes,
params);

sc.logout(session);
sc.closeSession(session);

//reading Attributes
BaseSignedData bs2 = new BaseSignedData( bs.getEncoded());
List<EAttribute> attrs ;
Signer aSigner = bs2.getSignerList().get(0);

attrs = aSigner.getAttribute( SigningTimeAttr.OID);
Calendar st = SigningTimeAttr.toTime(attrs.get(0));
System.out.println("Signing time: " + st.getTime());

attrs = aSigner.getAttribute( SignerLocationAttr.OID);
ESignerLocation sl = SignerLocationAttr.toSignerLocation(attrs.get(0));
StringBuilder sb = new StringBuilder();
for (String address : sl.getPostalAddress())
      sb.append(" " + address);
System.out.println("\nCountry: " + sl.getCountry() +
                   "\nCity: " + sl.getLocalityName() +
                   "\nAdress: "+ sb);

attrs = aSigner.getAttribute( ContentIdentifierAttr.OID);
byte [] ci = ContentIdentifierAttr.toIdentifier(attrs.get(0));
System.out.println("\n" + Arrays.toString(ci));

attrs = aSigner.getAttribute( CommitmentTypeIndicationAttr.OID);
CommitmentType ct = CommitmentTypeIndicationAttr.toCommitmentType(attrs.get
(0));
System.out.println("\n" + ct);
```

C#

```
List<IAttribute> optionalAttributes = new List<IAttribute>();
optionalAttributes.Add( new SigningTimeAttr( DateTime.UtcNow));
optionalAttributes.Add( new SignerLocationAttr("TURKEY", "KOCAELİ",
                                        new String[] { "TUBITAK UEKAE",
"GEBZE" }));
optionalAttributes.Add( new CommitmentTypeIndicationAttr( CommitmentType.
CREATION));
optionalAttributes.Add( new ContentIdentifierAttr( ASCIIEncoding.ASCII.
GetBytes("PL123456789")));

bs.addSigner( ESignatureType.TYPE_BES, cert, signer, optionalAttributes,
params_);

sc.logout(session);
```

```
sc.closeSession(session);

//reading Attributes
BaseSignedData bs2 = new BaseSignedData(bs.getEncoded());
List<EAttribute> attrs;
Signer aSigner = bs2.getSignerList()[0];

attrs = aSigner.getAttribute(SigningTimeAttr.OID);
DateTime? st = SigningTimeAttr.toTime(attrs[0]);
Console.WriteLine("Signing time: " + st.Value);

attrs = aSigner.getAttribute(SignerLocationAttr.OID);
ESignerLocation sl = SignerLocationAttr.toSignerLocation(attrs[0]);
StringBuilder sb = new StringBuilder();
foreach (String address in sl.getPostalAddress())
   sb.Append(" " + address);
Console.WriteLine("\nCountry: " + sl.getCountry() +
                          "\nCity: " + sl.getLocalityName() +
                          "\nAdress: " + sb);

attrs = aSigner.getAttribute(ContentIdentifierAttr.OID);
byte[] ci = ContentIdentifierAttr.toIdentifier(attrs[0]);
Console.WriteLine("\n" + BitConverter.ToString(ci));

attrs = aSigner.getAttribute(CommitmentTypeIndicationAttr.OID);
CommitmentType ct = CommitmentTypeIndicationAttr.toCommitmentType(attrs
[0]);
Console.WriteLine("\n" + ct);
```

3.5 Sertifika Doğrulaması

İmza atma işleminden önce sertifika doğrulaması yapılmaktadır. Bunun amacı iptal edilmiş veya hatalı bir sertifika ile imza atılmasının önlenmesidir. İmzacının sertifikasının doğrulanması parametreler yardımıyla yaptırtmayılabilir. Bu işlem için Parametreler bölümünde *P_VALIDATE_CERTIFICATE_BEFORE_SIGNING* parametresini inceleyebilirsiniz.

Eğer sertifika doğrulamasında bir hata çıkarsa *CertificateValidationException* hatası fırlatılmaktadır. Sertifika doğrulama ile ilgili ayrıntılı bilgi için İmza Doğrulama İşlemleri bölümünün altındaki Sertifika Doğrulaması 20 bölümüne bakınız.

3.6 Akıllı Kart İşlemleri

Akıllı kart işlemleri ile ilgili ayrıntılı bilgi için "ESYA API SmartCard Kullanım Kılavuzu" dökümanına bakınız. İmza atmanın matematiksel kısmı akıllı kart içinde gerçekleşir. İmza atmak isteyen kullanıcının bilgisayarına birden fazla akıllı kart takılı olabilir. Aynı zamanda bir akıllı kart içinde imza atabilecek birden fazla sertifika olabilir. Bu durumda hem akıllı kartın hem de imzayı atacak sertifikanın seçilmesi gerekmektedir.

Akıllı kart işlemlerini kolaylaştırmak amacıyla örnekler içindeki *SmartCardManager* sınıfından yararlanabilirsiniz. Bu sınıf genel imza işlemleriniz için yeterli olacaktır. Yalnız Java 1.6 ile çalışmaktadır.

Akıllı kart işlemleri akıllı kart üreticisinin geliştirdiği bir PKCS11 standardındaki sürücüler üzerinden yapılmaktadır. Windows için dll, linux için so tipindeki dosyalar aracılığı ile akıllı karta erişilir.

Akis kart PKCS11 dll'in yanında aynı zamanda java tabanlı kütüphane (AkisCIF.x.x.x.jar) de sunmaktadır. Doğrudan akıllı karta komut (APDU) gönderilebilmektedir. Böylelikle akıllı kart işlemlerinin süresi dolayısıyla imza süresi kısalmaktadır. Yalnız AkisCIF üzerinden karta erişildiğinde diğer programlar karta erişememektedir. Aynı zamanda kullandığınız AkisCIF, Akis'in bir yan ürünü olduğundan gelecekte her kartı desteklemeyebilir. Yeni bir sürüm akis kart kullanmaya başladığınızda AkisCIF'i de yenilemeniz gerekecektir.

3.6.1 SmartCardManager

Dağıtılan paket içinde örnek kodlar bölümünde *SmartCardManager* sınıfını bulabilirsiniz. Kendinize göre uyarlayabilmeniz için açık kaynak olarak dağıtılmaktadır. Bu sınıf ile temel imza işlemlerinizi gerçekleştirebilirsiniz. Sınıf aşağıdaki işlemleri sağlayabilir.

- Sisteme bir kart takılı ve kartta belirtilen özellikte bir sertifika varsa doğrudan bu kart ve bu sertifika üzerinden işlem yapar.
- Birden fazla kart takılı ise kullanıcıya kart seçtirir. Birden fazla belirtilen özellikte sertifika yüklü ise kullanıcıya sertifika seçtirir.
- Eğer APDU ile karta erişilmek isteniyorsa ve kart APDU ile erişimi destekliyorsa APDU ile karta erişim sağlar.
- Aynı kart ile imzalama işlemlerinde sertifikayı ve imzacıyı bellekten çekerek hızlanma sağlar.
- Bir kart ile işlem yaptıktan sonra eğer yeni bir kart takılmışsa veya işlem yapılan kart çıkartılmışsa kart ve sertifika seçme işlemlerini tekrarlar.

Örnek bir kullanım aşağıdaki gibi olabilir.

```
//Enable APDU usage
SmartCardManager.useAPDU( true);
//Connect a smartcard. If more than one smart card connected, user selects
```

```
one of them
SmartCardManager scm = SmartCardManager.getInstance();

//Get qualified certificate. If more than one qualified certificate, user
selects one of them.
ECertificate cert = scm.getSignatureCertificate(true, false);
//Create signer
BaseSigner signer = scm.getSigner("12345", cert);

/**
 * Create signature
 */

//If not sign again with selected card logout.
scm.logout();
//To select new card and new certificate, call reset.
scm.reset();
```

Yeni bir kartın takılıp takılmadığı seçili kartın çıkartılıp çıkartılmadığı *getInstance()* methodu içinde kontrol edilmektedir. Her imzalama işleminden önce *SmartCardManager* nesnesini *getInstance()* methodu ile alınız. Yukarıdaki örnek kodda işlemler kısa zamanda ardışıl olarak yapıldığından nesne bir kere alınmış ve o nesne üzerinden işlem yapılmıştır.

3.6.2 Slot İşlemleri

Akıllı kart işlemleri akıllı kart firmalarının kütüphaneleri yardımıyla gerçekleşmektedir. Dolayısıyla hangi kart takılı ise o kartın kütüphanesi kullanılmalıdır. Java 1.5'te takılı kart bilgilerine ulaşılamadığından, takılı olan kart türünün API'ye verilmesi gerekmektedir. Eğer çalışma ortamı Java 1.6 ise kart türü belirlenebilmektedir. Kart türünü belirlemek için *SmartOp* sınıfındaki *findCardTypeAndSlot(), findCardTypesAndSlots(), getCardTerminals()* ve *getSlotAndCardType(…)* fonksiyonlarından uygun olan kullanılabilir. Fonksiyonlar hakkında gerekli açıklamayı JavaDoc'ta bulabilirsiniz.

Eğer Java 1.5 ile çalışıyorsanız, AKİS kütüphanesini kullanarak kart okuyucuları listeleyebilirsiniz. Yalnız eğer kart akis değilse bağlanmaya çalıştığınızda hata alırsınız.

Java:

```
SmartCard sc = new SmartCard(CardType.AKIS);
long [] slots = sc.getSlotList();
for (long slot : slots)
{
      CK_SLOT_INFO slotInfo = sc.getSlotInfo(slot);
      System.out.println(new String(slotInfo.slotDescription).trim());
}
```

C#:

```
SmartCard sc = new SmartCard(CardType.AKIS);
long [] slots = sc.getSlotList();
foreach (long slot in slots)
{
      CK_SLOT_INFO slotInfo = sc.getSlotInfo(slot);
      Console.WriteLine(new String(slotInfo.slotDescription).Trim());
```

```
}
```

3.6.3 Karttan Sertifika Okuma

Karttaki sertifikaları okumak için aşağıdaki kod örneği kullanılabilir.

Java:

```
SmartCard sc = new SmartCard(CardType.AKIS);
long [] slots = sc.getSlotList();
long slot = 0;
if(slots.length == 1)
   slot = slots[0];
else
   slot = selectSlot();
long session = sc.openSession(slot);
sc.login(session, "12345");
List<byte[]> certBytes = sc.getSignatureCertificates(session);
for (byte[] bs : certBytes)
{
      ECertificate cert = new ECertificate(bs);
      //cert.isQualifiedCertificate()
      System.out.println(cert.getSubject().getCommonNameAttribute());
}
sc.logout(session);
sc.closeSession(session);
```

C#:

```
SmartCard sc = new SmartCard(CardType.AKIS);
long [] slots = sc.getSlotList();
long slot = 0;
if(slots.Length == 1)
      slot = slots[0];
else
      slot = selectSlot(…);
long session = sc.openSession(slot);
sc.login(session, "12345");
List<byte[]> certBytes = sc.getSignatureCertificates(session);
foreach (byte[] bs in certBytes)
{
      ECertificate cert = new ECertificate(bs);
      //cert.isQualifiedCertificate()
      Console.WriteLine(cert.getSubject().getCommonNameAttribute());
}
sc.logout(session);
sc.closeSession(session);
```

Eğer nitelikli sertifikalar imzada kullanılacaksa, *Ecertificate.isQualifiedCertificate()* fonksiyonu kullanılabilir.

Her seferinde karttan sertifika okuma işlemi zaman alabilir. Bu API'yi kullanan

uygulama içinde kullanıcının imzalamada kullanmak istediği sertifika veya sertifikalar tanımlanabilir. Böylelikle her imzalama işlemi için karttan sertifika okunmak zorunda kalınmaz.

3.6.4 Karttan Anahtar Etiketlerinin Okunması

Anahtar etiketleri imzacı oluştururken kullanılabilir. Aşağıdaki örnek kod bloğu ile anahtar etiketleri okunabilir.

Java:

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
long session = sc.openSession(slot);
sc.login(session, "12345");
String [] labels = sc.getSignatureKeyLabels(session);
for (String label : labels)
{
      System.out.println(label);
}
```

C#:

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
long session = sc.openSession(slot);
sc.login(session, "12345");
String [] labels = sc.getSignatureKeyLabels(session);
foreach (String label in labels)
{
      Console.WriteLine(label);
}
```

3.6.5 Akıllı Kart İmzacılar Oluşturma

Akıllı karttan imzacı oluştururken sertifika seri numarası veya anahtarın etiketi kullanılabilir. *SCSignerWithCertSerialNo* ve *SCSignerWithKeyLabel* sınıfları kullanılabilir.

Sertifika seri numarası ile imzacı aşağıdaki şekilde oluşturulabilir.

Java:

```
SmartCard sc = new SmartCard(CardType.AKIS);
long [] slots = sc.getSlotList();
long session = sc.openSession(slots[0]);
sc.login(session, "12345");
ECertificate cert = new ECertificate(new File(SIGNING_CERTIFICATE_PATH));


BaseSigner signer = new SCSignerWithCertSerialNo (sc, session, slots[0],
cert.getSerialNumber().toByteArray
          (), SignatureAlg.RSA_SHA1.getName());
```

C#:

```
SmartCard sc = new SmartCard(CardType.AKIS);
long[] slots = sc.getSlotList();
long session = sc.openSession(slots[0]);
sc.login(session, "12345");
ECertificate cert = new ECertificate(new FileInfo
(SIGNING_CERTIFICATE_PATH));

BaseSigner signer = new SCSignerWithCertSerialNo(sc, session, slots[0],
cert.getSerialNumber().GetData(), SignatureAlg.RSA_SHA1.getName());
```

Anahtar etiketi ile imzacı ise aşağıdaki şekilde oluşturulabilir.

Java:

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
long session = sc.openSession(slot);
sc.login(session, "12345");
BaseSigner signer = new SCSignerWithKeyLabel(sc,session, slot,"yasemin.
akturk#ug.netSIGN0",
          SignatureAlg.RSA_SHA1.getName());
```

C#:

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
long session = sc.openSession(slot);
sc.login(session, "12345");
BaseSigner signer = new SCSignerWithKeyLabel(sc,session, slot,"yasemin.
akturk#ug.netSIGN0", SignatureAlg.RSA_SHA1.getName());
```

4 İmza Doğrulama İşlemleri

İmza doğrulama işlemi birçok kontrolcünün bir araya gelmesiyle yapılmaktadır. Bu kontrolcüler imzanın yapısal kontrollerinden ve sertifika kontrollerinden ulaşmaktadır. Sertifika kontrolleri sertifikanın yapısal kontrollerinden, sertifikanın güvenilir bir kökten verilmiş olmasından ve sertifika iptal kontrolünden oluşmaktadır. Sertifika iptal kontrolü dışında kalan kontroller matematiksel işlemlerden oluşmaktadır, ekstra bir bilgiye ihtiyaç duymamaktadır. Sertifika iptal kontrolü için ise sertifikayı verenin yayınlandığı uygun zamanlı SİL veya ÇİSDUP bilgisine ihtiyaç duyulmaktadır. Sertifika doğrulama işlemi için sertifika doğrulama işleminin konfigurasyonunu barındıran xml formatında politika dosyasına ihtiyaç vardır. Bu xml dosyası dışında birçok parametre kullanılabilir. Bu parametrelere Parametreler 32 bölümünden veya *EParameters* sınıfından bakabilirsiniz.

Bir imzalanmış verinin doğrulama işleminden *SignedDataValidation* sınıfı sorumludur. *SignedDataValidation* sınıfının *verify(…)* methodu kullanılarak imza doğrulama işlemi gerçeklenir. v*erify(…)* fonksiyonu imzalanmış verinin byte [] halini ve imzalanmış verinin hangi parametrelere göre doğrulanacağı bilgisini alır. *SignedDataValidation* sınıfındaki *verify(…)* fonksiyonu ile imzalanmış döküman bir bütün halinde doğrulanır. İçindeki imzalardan biri doğrulanamamışsa *verify(…),* başarısız değer döner. İmza doğrulama yapan örnek kod bloğu;

Java:

```
byte[] signedData = AsnIO.dosyadanOKU(SIGNATURE_FILE);
ValidationPolicy policy=  PolicyReader.readValidationPolicy(new
FileInputStream(POLICY_FILE));

Hashtable<String, Object> params = new Hashtable<String, Object>();
params.put(EParameters.P_CERT_VALIDATION_POLICY, policy);

SignedDataValidation sdv = new SignedDataValidation();

SignedDataValidationResult sdvr = sdv.verify(signedData, params);

if(sdvr.getSDStatus() != SignedData_Status.ALL_VALID)
      System.out.println("Imzalarin hepsi dogrulanamadi");

System.out.println(sdvr.toString());
```

C#:

```
byte[] signedData = AsnIO.dosyadanOKU(SIGNATURE_FILE);
ValidationPolicy policy;
using (FileStream fs = new FileStream(POLICY_FILE, FileMode.Open,
FileAccess.Read))
{
      policy = PolicyReader.readValidationPolicy(fs);
}
Dictionary<String, Object> parameters = new Dictionary< String, Object>();
parameters[EParameters.P_CERT_VALIDATION_POLICY] = policy;

SignedDataValidation sdv = new SignedDataValidation();
SignedDataValidationResult sdvr = sdv.verify(signedData, parameters);

if(sdvr.getSDStatus() != SignedData_Status.ALL_VALID)
      Console.WriteLine("İmzaların hepsi doğrulanamadı");

Console.WriteLine(sdvr.ToString());
```

Eğer imzalanan içerik imza içersinde ise içerik, *BaseSignedData* sınıfının *getSignedData().getEncapsulatedContentInfo().getContent()* fonksiyonu ile alınabilir.

4.1 İmza Doğrulama Sonucu

SignedDataValidation sınıfının verify(...) fonksiyonu doğrulama sonucu olarak *SignatureValidationResult* tipinde bir nesne döner. *getSDStatus()* fonksiyonu eğer bütün imzalar doğrulanmış ise *ALL_VALID,* eğer imzalardan en az bir tanesi doğrulanamamışsa *NOT_ALL_VALID* döner.*SignedDataValidation* nesnesinin *toString()* methodu bütün imzaların kontrol sonucu açıklamalarını döner. Eğer özellikle bir imzanın sonucu elde edilmek isteniyorsa imza ağacında gezerek elde edilebilir. Bu ağaç yapısı, BaseSignedData yapısındaki imzaların veri yapısı ile aynıdır. Örneğin birinci paralel imzanın, birinci seri imzasına ve bu imzanın doğrulama sonucuna aşağıdaki örnek kod ile ulaşılabilir.

Java:

```
BaseSignedData bs = getBaseSignedData();
SignedDataValidationResult sdvr = getValidationResult();
Signer firstCounterSigner = bs.getSignerList().get(0).getCounterSigners().
get(0);
SignatureValidationResult firstCounterSignerVR = sdvr.
getSDValidationResults().get(0).getCounterSigValidationResults().get(0);
```

C#:

```
Signer firstCounterSigner = bs.getSignerList()[0].getCounterSigners()[0];
SignatureValidationResult firstCounterSignerVR = sdvr.
getSDValidationResults()[0].getCounterSigValidationResults()[0];
```

Herbir imzanın doğrulama sonucu *SignatureValidationResult* nesnelerinde tutulur. Bir imzacının doğrulama sonucundan seri imzacılarının doğrulama sonuçlarına da erişilebilir. *SignatureValidationResult* nesnelerinin *toString()* fonksiyonu imzacının ve seri imzacıların doğrulama kontrolcülerinin sonuçlarını döner. Eğer sadece o imzaya ait kontrolcülerin açıklamaları elde edilmek isteniyorsa *getValidationDetails()* fonksiyonunu kullanınız. İmza tipi geliştikçe kontrol edilmesi gereken yapı ve sertifika artmaktadır. Zaman damgaları imza yapısına bir imza olarak eklendiklerinden ayrıca bir imza olarak kontrol edilmektedirler.

SignatureValidationResult nesnesinden bir imzanın doğrulama sonucu *getSignatureStatus()* fonksiyonu ile *Signature_Status* yapısında alınabilir. Eğer imza sonucu *INCOMPLETE* ise sertifika doğrulama verisine ulaşılamamıştır.

4.2 Ön Doğrulama

Bir imzanın doğrulanması için imza atıldıktan sonra sertifika iptal bilgilerinin güncellenebilmesi için belirli bir süre geçmesi gerekmektedir. Bu süreye "kesinleşme süresi"(grace period) denilmektedir. API'de bu süre *P_GRACE_PERIOD* parametresi ile ayarlanabilir; varsayılan değeri 86400 saniye yani 24 saattir.

Bir imza doğrulanmaya çalışıldığında kesinleşme süresi geçmese bile imza ve sertifika doğrulanabilmektedir. Öndoğrulama denilen bu doğrulama bir kesinlik içermemektedir. Kesin bir doğrulama yapılabilmesi için kesinleşme süresinin geçmesi gerekmektedir. Bir imzanın doğrulanmasının ön doğrulama olduğunu aşağıdaki örnek kod ile öğrenebilirsiniz. Ön doğrulama için olgulaşmamış anlamına gelen "*PREMATURE*" kelimesi, kesinleşmiş imza için olgunlaşmış anlamına gelen "*MATURE*" kelimesi kullanılmaktadır.

Java:

```
If(sdvr.getSDValidationResults().get(0).getValidationState() ==
ValidationState.PREMATURE)
      System.out.println("Ön doğrulama yapıldı.");
```

C#:

```
if(sdvr.getSDValidationResults()[0].getValidationState() ==
ValidationState.PREMATURE)
Console.WriteLine("Ön doğrulama yapıldı");
```

Ön doğrulama ile kesin doğrulama arasında doğabilecek fark; ön doğrulama sırasında geçerli olan bir sertifikanın, kesin doğrulamada iptal edilmiş olmasıdır. Kullanıcı akıllı kartını çaldırdığında bu senaryo ile karşılaşılabilir.

ÇiSDuP için kesinleşme süresi kısa olabilmektedir. Yalnız iptal bilgileri için SİL kullanılıyorsa bu süre uzayabilmekte.

"E-İmza Profilleri" dökümanı ÇiSDuP kullanıldığında kesinleşme süresinin çok kısa olduğundan yok sayılabileceğini öngörüyor. ÇiSDuP kullanarak sertifika doğrulama yapıldığında, kesinleşme süresi kadar beklenmediğinden büyük avantaj sağlanmaktadır.

4.3 Ayrık İmzanın Doğrulanması

Ayrık imza doğrulanırken imzalanan dökümanının parametre olarak verilmesi gerekmektedir. *P_EXTERNAL_CONTENT* parametresine *ISignable* türünden nesne verilmelidir.

Java:

```
byte[] signedData = AsnIO.dosyadanOKU(SIGNATURE_FILE);
ISignable content = new SignableFile(new File(CONTENT_FILE));

ValidationPolicy policy=  PolicyReader.readValidationPolicy(new
FileInputStream(POLICY_FILE));
Hashtable<String, Object> params = new Hashtable< String, Object>();
params.put(EParameters.P_CERT_VALIDATION_POLICY, policy);
params.put(EParameters.P_EXTERNAL_CONTENT, content );

SignedDataValidation sdv = new SignedDataValidation();
SignedDataValidationResult sdvr = sdv.verify(signedData, params);

if(sdvr.getSDStatus() != SignedData_Status.ALL_VALID)
      System.out.println("Imzalarin hepsi dogrulanamadi");

System.out.println(sdvr.toString());
```

C#:

```
Byte[] signedData = AsnIO.dosyadanOKU(SIGNATURE_FILE);
ISignable content = new SignableFile(new FileInfo(CONTENT_FILE));
ValidationPolicy policy;
using (FileStream fs = new FileStream(POLICY_FILE, FileMode.Open,
FileAccess.Read))
{
   policy = PolicyReader.readValidationPolicy(fs);
}
Dictionary<String, Object> parameters = new Dictionary<String, Object>();
parameters[EParameters.P_CERT_VALIDATION_POLICY] = policy;
parameters[EParameters.P_EXTERNAL_CONTENT] = content;

SignedDataValidation sdv = new SignedDataValidation();
SignedDataValidationResult sdvr = sdv.verify(signedData, parameters);
```

```
if (sdvr.getSDStatus() != SignedData_Status.ALL_VALID)
   Console.WriteLine("Imzalarin hepsi dogrulanamadi");

Console.WriteLine(sdvr.ToString());
```

4.4 Sertifika Doğrulama

İmza atarken ve imza doğrulama sırasında CMS Signature kütüphanesi imzacıların sertifikalarını doğrular. Sertifikaların doğrulama işlemleri sertifika doğrulama politikasına göre yapılmaktadır. Politika bilgisi *Eparameters.P_CERT_VALIDATION_POLICY* parametresi ile *ValidationPolicy* nesnesi tipinde verilir.

Politika bilgileri xml dosyasında saklanır. Xml dosyasındaki verilerin okunup *ValidationPolicy* nesnesine dönüştürülmesinden *PolicyReader* sınıfı sorumludur.

İmza doğrulama sırasında sertifika doğrulama işlemi API tarafından yapılmaktadır. Bazı durumlarda sadece sertifikanın doğrulanmasına ihtiyaç olabilir. Bu gibi durumlar için aşağıdaki örnek kod kullanılabilir.

Java

```
ValidationSystem vs = CertificateValidation.createValidationSystem(policy);
vs.setBaseValidationTime(Calendar.getInstance());
CertificateStatusInfo csi = CertificateValidation.validateCertificate(vs,
cert);
if(csi.getCertificateStatus() != CertificateStatus.VALID)
      System.out.println("Sertifika dogrulanamadi");
```

C#

```
ValidationSystem vs = CertificateValidation.createValidationSystem(policy);
vs.setBaseValidationTime(DateTime.UtcNow);
CertificateStatusInfo csi = CertificateValidation.validateCertificate(vs,
cert);
if (csi.getCertificateStatus() != CertificateStatus.VALID)
   Console.WriteLine("Sertifika dogrulanamadi");
```

4.4.1 Politika Dosyası

Politika dosyası, sertifika doğrulama sırasında kullanılacak verilerin nasıl bulunacağını (find), doğru verilerin bulunup bulunmadığının kontrolü için nasıl eşleştirme yapılacağını (match) ve doğrulama sırasında hangi kontrollerin yapılacağını (validate) belirten sınıfları bulunduran dosyadır.

Politika dosyasının değiştirilmemesi imza doğrulama için hayati önem taşır. Politika dosyasında hangi kontrollerin yapılacağı ve hangi sertifikalara güvenileceği bilgisi yer almaktadır. Kötü niyetli kişiler politika dosyasını değiştirerek, kötü niyetle yaratılmış imzaların doğrulanmasını sağlayabilirler.

Politika dosyası için aşağıdaki güvenlik önlemleri uygulanabilir;

1. Politika dosyası sunucudan çekilir ve bellekte tutulur. Böylelikle politika dosyasına dışardan müdahale olasılığı azalır.

2. Politika dosyasının özeti sunucuda tutulur. İstemcideki politika dosyasının özeti alınır ve sunucudan çekilen özet ile karşılaştırılır. DigestUtil sınıfı ile özet işlemini gerçekleştirebilirsiniz.
3. Politika dosyası istemcide parola tabanlı şifrelenerek tutulur ve şifresi bellekte çözülür. Bunun nasıl yapıldığı gösteren örneği "http://www.java2s.com/Tutorial/Java/0490__Security/PBEFileEncrypt.htm" adresinde bulabilirsiniz.
4. Politika dosyası imzalı bir şekilde tutulur. Politika dosyasına karar verildikten sonra politika dosyası bir kere imzalanır ve bu imzalı dosya istemcilerde tutulur. İmzalanan politika dosyasının imzası doğrulanırsa ve imza doğru kişi tarafından atılmışsa; politika dosyasına güvenilebilir. Doğru kişi imzalamış mı kontrolünün yapılması için, kodunuzun içine imzacı sertifikasının gömülmesi gerekmektedir. Kodunuzu, devamlı değiştiremeyeceğinizden imzalama işleminde kullandığınız akıllı kartı veya pfx dosyasını kesinlikle kaybetmemeniz gerekmektedir. Politika dosyasını PKCS7 standardında imzalayabilirsiniz. PKCS7 tipindekiyi imza işlemlerine smartcard modülünden erişebilirsiniz.

Yukarıdaki güvenlik önlemleri birlikte veya tek olarak uygulanabilir.

Politika dosyasının güvensiz bir şekilde istemcilerde durmasını kesinlikle önermiyoruz.

4.4.1.1 Bulucular

Sertifika doğrularken gerekli olan verilerin hangi yöntemlerle bulunacağını belirleyen sınıfları içerir. Buraya yazılacak sınıfların *Finder* sınıfından türemiş olmaları gerekmektedir. Bulucular hangi bilginin erişiminde kullanılacaklarına göre alt kategorilere ayrılmaktadırlar. Bunlar güvenilir kabul edilen sertifikalar (trustedcertificate), sertifikalar (certificate), iler (crl) ve ÇiSDuP (ocsp) kategorileridir.

Güvenilir Kabul Edilen Sertifikalar

Sertifika doğrulama sırasında güvenilirlikleri sorgulanmayacak sertifikaların bulunmasından sorumlu sınıfların belirtildiği alandır. Buradaki bulucular tarafından bulunan sertifikaların doğrulaması yapılmaz. Buraya yazılacak bulucular *TrustedCertificateFinder* sınıfından türemelidir.

***TrustedCertificateFinderFromECertStore*:** Sertifika deposundaki güvenilir sertifikaları getiren sınıftır. Güvenilir sertifikalar, sertifika deposunda güvenli bir şekilde saklandıklarından bu bulucunun kullanılması önerilmektedir.

***TrustedCertificateFinderFromFileSystem*:** Belirtilen klasör içindeki sertifikalar güvenilir sertifika olarak kabul edilir. Test sertifikalarının kökleri Sertifika Deposunda bulunmadığından test kökleri klasör yoluyla belirtilebilir.

Sertifikalar

Doğrulama sırasında gerekli sertifikaların bulunmasından sorumlu sınıfların belirtildiği alandır. Bir sertifika güvenilir bir sertifika değilse, sertifika yolu takip edildiğinde güvenilir bir sertifikaya ulaşılması gerekmektedir. Buraya yazılacak bulucular *CertificateFinder* sınıfından türemelidirler.

***CertificateFinderFromHTTP*:** Sertifika içindeki 'Authority Info Access' bilgisinden sertifikayı verenin sertifikasına http'den erişir.

CertificateFinderFromLDAP: Sertifika içindeki 'Authority Info Access' bilgisinden sertifikayı verenin sertifikasına ldap'tan erişir.

CertificateFinderFromECertStore: Sertifika deposundan sertifikaya erişir.

CertificateFinderFromFile: Dosya yolundan sertifikaya erişir.

Siller

Doğrulama sırasında gerekli ilerli bulunmasından sorumlu sınıfların belirtildiği alandır. Buraya yazılacak bulucular *CRLFinder* sınıfından türemelidirler.

CRLFinderFromHTTP: Sertifika içindeki 'Authority Info Access' bilgisinden sertifikaya ait güncel sil dosyasına http'den erişir.

CRLFinderfromLDAP: Sertifika içindeki 'Authority Info Access' bilgisinden sertifikaya ait güncel sil dosyasına ldap'tan erişir.

CRLFinderFromECertStore: İmzalanmış veri içinde referansı bulunan sil dosyasını sertifika deposundan getirir.

CRLFinderFromFile: İşaret edilen sil dosyasını getirir.

CRLFinderFromHTTPAddress: İşaret edilen http adresindeki sil dosyasını getirir.

CRLFinderFromLDAPAddress: İşaret edilen ldap adresindeki sil dosyasını getirir.

ÇiSDuP'lar

Doğrulama sırasında gerekli ÇisDuP'ların bulunmasından sorumlu sınıfların belirtildiği alandır. Buraya yazılacak bulucular *OCSPResponseFinder* sınıfından türemelidirler.

OCSPResponseFinderFromAIA: Sertifika içindeki 'Authority Info Access' bilgisinden sertifikaya ait ÇisDuP sorgusunu yapar.

OCSPResponseFinderFromECertStore: İmzalanmış veri içinde referansı bulunan ÇisDuP cevabını sertifika deposundan getirir.

4.4.1.2 Eşleştiriciler

Sertifika ve ilerli hangi yöntemlerle eşleştirileceğini belirten sınıfları içeren alandır. Buraya yazılacak bulucular *Matcher* sınıfından türemelidirler.

4.4.1.3 Doğrulayıcılar

Sertifika, sil, ÇisDuP'ların doğrulanmasında kullanılacak sınıfları içeren alandır. Buraya yazılacak kontrolcüler *Checker* sınıfından türemelidir.

4.4.2 Sertifika Deposu

Sertifika deposu güvenilir olduğu varsayılan sertifikaların saklanmasından ve sertifika doğrulama verilerinin saklanmasından sorumludur.

Sertifika deposunun varsayılan yeri, kullanıcının ana klasörünün(user home directory) altındaki ".sertifikadeposu" klasörünün içidir. Varsayılan dosya ismi ise "SertifikaDeposu. svt"dir. Eğer farklı bir dosya yolu ve dosya ismi kullanılması isteniyorsa, sertifika doğrulama politika dosyasında belirtilmelidir. Bunun için *storepath* parametresi kullanılmalıdır.

```
<class name="tr.gov.tubitak.uekae.esya.api.certificate.validation.find.
certificate.trusted.TrustedCertificateFinderFromECertStore">
     <param name="storepath" value="C:\SertifikaDeposu\SertifikaDeposu.
svt"/>
 </class>
```

Test sertifikalarıyla çalışırken sertifikanın doğrulanması sırasında "*PATH_VALIDATION_FAILURE*" hatası alabilirsiniz. Bunun sebebi test sertifikalarının köklerinin sertifika deposunda güvenilir sertifika olarak tanımlanmadığından olabilir. Bu kök sertifikaları dosya yoluyla belirtebilirsiniz. Yalnız bu kullanım sadece test amacıyla yapılmalıdır.

```
<trustedcertificate>
      <class name="tr.gov.tubitak.uekae.esya.api.certificate.validation.
find.certificate.trusted.TrustedCertificateFinderFromECertStore"/>
      <class name="tr.gov.tubitak.uekae.esya.api.certificate.validation.
find.certificate.trusted.TrustedCertificateFinderFromFileSystem">
            <param name="dizin" value="T:\MA3\api-
cmssignature\testdata\cmssignature\imza\creation\plugtests\certscrls\root\"
></param>
      </class>
</trustedcertificate>
```

Bazı durumlarda kesin bir dosya yolu belirtemeyebilirsiniz. Bu gibi durumlarda programınızın yürürlükteki dizinine göre bağıl dizin adresi verilebilir.

```
<class name="tr.gov.tubitak.uekae.esya.api.certificate.validation.find.
certificate.trusted.TrustedCertificateFinderFromFileSystem">
      <param name="dizin" value="certscrls/root"></param>
</class>
```

4.5 İmzacıların Alınması

İmza yapısı *BaseSignedData* sınıfı tarafından işlenmektedir. Bu yapıda seri ve paralel imzacılar bir ağaç yapısında bulunmaktadırlar. *BaseSignedData* sınıfının *getSignerList*() fonksiyonu ile birinci seviye imzacılar alınmaktadır. Sadece paralel imza atılmışsa *getSignerList*() fonksiyonu ile bütün imzacılar alınmış olur. Seri imzacıları almak için ise seri imzacıları alınmak istenen imzacının *getCounterSigners()* fonksiyonu çağrılmalıdır. Eğer imza seviyeleri önemli değilse *BaseSignedData* sınıfının *getAllSignerList()* fonksiyonu kullanılarak bütün imzacılar alınabilir. Daha ayrıntılı bilgi için örnek kodlar içinde yer alan *SignersInJTree* sınıfını inceleyebilirsiniz.

İmza işlemlerinde imza atan kişiyi tanımlama işlemi kişinin sertifikası üzerinden yapılmaktadır.

Standartlara göre imza atanın sertifikasının imza yapısında olması zorunlu değildir. Yalnız genel davranış olarak sertifika imza yapısına konur. ESYA kütüphanesinde de sertifika imza yapısına eklenmektedir. Sertifikadan, kişinin ismi ve Türkiye için TC. kimlik numarası alınabilir.

Java

```
BaseSignedData bsd = new  BaseSignedData(signedData);
ECertificate cert = bsd.getSignerList().get(0).getSignerCertificate();

if(cert == null){
      System.out.println("İmzaci bilgisi yok");
} else {
      System.out.println("İsim & Soyisim: " + cert.getSubject().
getCommonNameAttribute());
      System.out.println("TC Kimlik No: " + cert.getSubject().
getSerialNumberAttribute());
}
```

C#

```
BaseSignedData bsd = new BaseSignedData(signedData);
ECertificate cert = bsd.getSignerList()[0].getSignerCertificate();

if (cert == null)
{
   Console.WriteLine("İmzaci bilgisi yok");
}
else
{
   Console.WriteLine("İsim & Soyisim: " + cert.getSubject().
   getCommonNameAttribute());
   Console.WriteLine("TC Kimlik No: " + cert.getSubject().
   getSerialNumberAttribute());
}
```

5 İmza Tipleri

API tarafından desteklenen imza tiplerinden çok kullanılanları hakkında kısa açıklamaları burada bulabilirsiniz. Hangi imzanın size uygun olduğuna karar vermek için "E-imza Profilleri" dökümanını inceleyebilirsiniz. İmza tipleri hakkında daha geniş bilgi için ise "ETSI TS 101 733" dökümanına bakınız.

5.1 BES

BES imza, en basit imza türüdür. BES, imza sadece o kişinin imzayı attığını garanti eder. İmza zamanıda belli olmadığından ancak sertifika geçerli iken imza doğrulanabilir. Sertifika iptal edildiğinde veya sertifika süresi dolduğunda imza doğrulanamaz. BES imza içersine zaman bilgisi eklenebilir, eklenen zamanın herhangi bir hukuki yükümlülüğü, kesinliği yoktur. Beyan edilen zaman şeklinde kullanılabilir.

Eğer beyan edilen zamana güvenmiyorsanız *Eparameters.P_TRUST_SIGNINGTIMEATTR* parametresine *false* değerini atayınız.

Beyan edilen zamanı eklemek için *SigningTimeAttr* özelliği kullanılmalıdır. *SigningTimeAttr* özelliği eğer bir zaman verilmeden kullanılırsa zaman bilgisini imzanın atıldığı bilgisayarın zamanından alır. Eğer kullanıcı zamanına güvenmiyorsanız *P_SIGNING_TIME* parametresi ile zaman ataması yapabilirsiniz.

Java:

```
BaseSignedData bs = new BaseSignedData();
bs.addContent(new SignableByteArray("test".getBytes()));

//Since SigningTime attribute is optional,add it to optional attributes
list
List<IAttribute> optionalAttributes = new ArrayList<IAttribute>();
optionalAttributes.add(new SigningTimeAttr(Calendar.getInstance()));

HashMap<String, Object> params = new HashMap<String, Object>();
params.put(EParameters.P_CERT_VALIDATION_POLICY, VALIDATION_POLICY);

bs.addSigner(ESignatureType.TYPE_BES, getSignerCertificate(),
getSignerInterface(SignatureAlg.RSA_SHA1), optionalAttributes, params);
```

C#:

```
BaseSignedData bs = new BaseSignedData();
bs.addContent(new SignableByteArray(ASCIIEncoding.ASCII.GetBytes
("test")));

//Since SigningTime attribute is optional,add it to optional attributes
list
List<IAttribute> optionalAttributes = new List<IAttribute>();
optionalAttributes.Add(new SigningTimeAttr(DateTime.UtcNow));

Dictionary<String, Object> parameters = new Dictionary<String, Object>();
parameters[EParameters.P_CERT_VALIDATION_POLICY] = POLICY_FILE;
parameters[EParameters.P_SIGNING_TIME] = DateTime.Now;

bs.addSigner(ESignatureType.TYPE_BES, getSignerCertificate(),
getSignerInterface(SignatureAlg.RSA_SHA1), optionalAttributes,
parameters);
```

İmzanın beyan edilen zamanını almak için aşağıdaki örnek kod kullanılabilir.

Java:

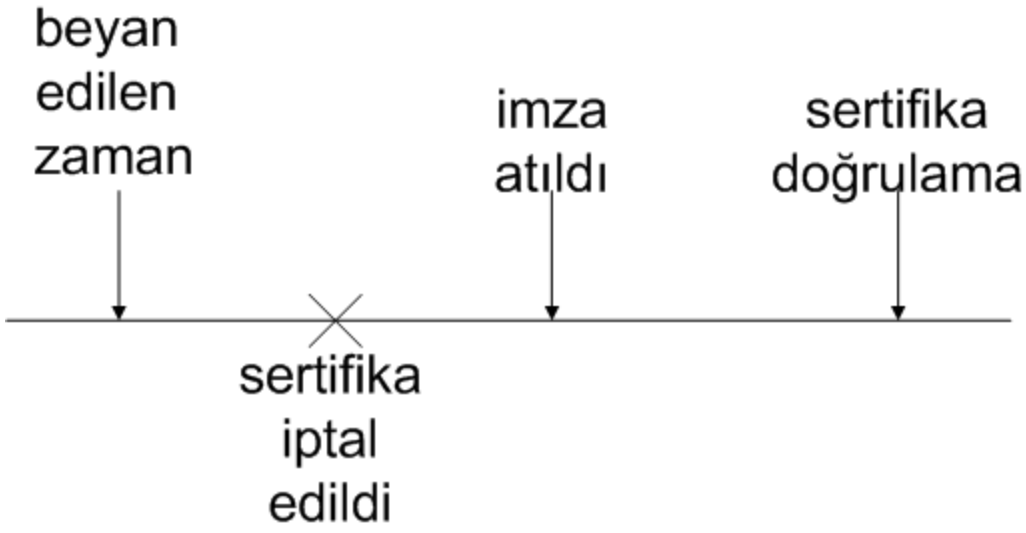
```
BaseSignedData bs = new BaseSignedData(input);
List<EAttribute>  attrs = bs.getSignerList().get(0).getSignedAttribute
(SigningTimeAttr.OID);
Calendar time = SigningTimeAttr.toCalendar(attrs.get(0));
System.out.println(time.getTime().getTime().toString());
```

C#:

```
BaseSignedData bs = new BaseSignedData(input);
List<EAttribute>  attrs = bs.getSignerList()[0].getSignedAttribute
```

```
(AttributeOIDs.id_signingTime);
ETime time = new ETime(attrs[0].getValue(0));
Console.WriteLine(time.getTime().Value.ToString());
```

Beyan edilen zamanın kullanımında oluşabilecek kötü senaryo aşağıdaki gibidir. İmza atıldığı sırada imzacı sertifikası iptal edilmiştir; yalnız kullanıcı imza zamanı olarak daha önceki bir zamanı beyan etmiştir. Beyan edilen zamana güvenildiği durumda geçersiz olan bu imza doğrulanacaktır.



5.2 EST

EST imza, BES imzadan türemiştir. İçerisinde imzalama zamanını gösterir zaman damgası bulundurmaktadır. İmza atılırken zaman damgası ayarlarının verilmesi gerekmektedir. Aşağıdaki örnek kodda bir EST imzanın nasıl atılacağını bulabilirsiniz.

Java:

```
BaseSignedData bs = new BaseSignedData();
bs.addContent(new SignableByteArray("test".getBytes()));

HashMap<String, Object> params = new HashMap<String, Object>();

TSSettings tsSettings = new TSSettings("http://zd.ug.net", 21, "12345678".
toCharArray());
params.put(EParameters.P_TSS_INFO, tsSettings);
params.put(EParameters.P_TS_DIGEST_ALG, DigestAlg.SHA1);
params.put(EParameters.P_CERT_VALIDATION_POLICY, TestConstants.getPolicy
());

//add signer
bs.addSigner(ESignatureType.TYPE_EST, getSignerCertificate(),
getSignerInterface(SignatureAlg.RSA_SHA1), null, params);
```

C#:

```
BaseSignedData bs = new BaseSignedData();
```

```
bs.addContent(new SignableByteArray(ASCIIEncoding.ASCII.GetBytes
("test")));
Dictionary<String, Object> parameters = new Dictionary<String, Object>();

//for getting signaturetimestamp
TSSettings tsSettings = new TSSettings("http://zd.ug.net", 21,
"12345678");
parameters[EParameters.P_TSS_INFO] = tsSettings;
parameters[EParameters.P_TS_DIGEST_ALG] = DigestAlg.SHA1;
parameters[EParameters.P_CERT_VALIDATION_POLICY] = TestConstants.getPolicy
();

//add signer
bs.addSigner(ESignatureType.TYPE_EST, TestConstants.getSignerCertificate
(), TestConstants.getSignerInterface(SignatureAlg.RSA_SHA1), null,
parameters);
```

İmza zamanı belli olduğundan, BES tipi imzadan farklı olarak sertifika doğrulamada kesin sonuçlara ulaşılabilir. Aşağıdaki örnek kod ile imza zamanını alabilirsiniz.

Java:

```
BaseSignedData bs = new BaseSignedData(input);
EST estSign = (EST)bs.getSignerList().get(0);
Calendar time = estSign.getTime();
System.out.println(time.getTime().toString());
```

C#:

```
BaseSignedData bs = new BaseSignedData(input);
EST estSign = (EST)bs.getSignerList()[0];
DateTime? time = estSign.getTime();
Console.WriteLine(time.Value.ToString());
```

5.3 ESXLong

ESXLong imza aynı zamanda bir EST imzadır. ESXLong, ESTimzadan farklı olarak imza içerisinde sertifika doğrulamada kullanılacak doğrulama verisini içerir. Böylelikle çevrim dışı durumlarda bile imza doğrulabilir. Sertifika ömrü dolduktan sonra doğrulamada kullanılacak veriye ulaşmada sorunlar çıkabilir. ESXLong imza doğrulama verisini içinde barındırdığından bu tür sorunların çıkmasını engeller.

5.4 ESA

ESA tipi kriptografik algoritmaların zamanla güvenilirliğini kaybetmesine karşı geliştirilmiş bir imza türüdür. Şu anda güvenerek kullandığımız algoritmalar, 5-10 yıl sonra güvenilemez olabilir. Bu algoritmalar güvenilmez duruma geçmeden önce, imzaların ESA tipine çevrilmesi gerekmektedir. Yalnız imza atıldıktan hemen sonra imzanın ESA'ya çevrilmesi fazladan bir güvenlik sağlamaz. Algoritmaların bir kısmı güvenilmez duruma geçerken, daha güvenilir yeni algoritmalar kullanılmaya başlanacaktır. ESA tipine çevrim sırasında bu yeni algoritmalar kullanılmalıdır.

ESA'yı kullanmanın bir diğer amacı imza üzerinde değişiklik yapılmasını engellemek olabilir. Eğer seri imza kullanılıyorsa; ESA'ya çevrilen imzanın altındaki imzaların veri yapısında bir değişiklik yapılamaz. Bu değişiklikler imza türünün değiştirilmesi, imzanın silinmesi, yeni imza eklenmesi olabilir.

Bir imza atılırken ESA tipinde atılamaz. Öncelikle başka bir tipte atılmalı, daha sonra ESA tipine çevrilmelidir. Nasıl yapılacağına "İmza Tipleri Arasında Dönüşüm" bölümünden bakabilirsiniz.

ESA tipine dönüşüm sırasında arşiv tipi zaman damgası kullanılmaktadır. Bundan dolayı parametreler yardımıyla zaman damgası ayarları verilmelidir.

6 İmza Tipleri Arasında Dönüşüm

İmza tipleri arasında dönüşüm işlemi *BaseSignedData* sınıfı aracılığı ile yapılabilir. Öncelikle imzalanmış verinin imzacıları *BaseSignedData.getSignerList()* fonksiyonuyla imzalanmış veri içinden alınır. İmza tipi değiştrilmek istenen imzacı liste içinden bulunarak, imzacının *convert()* fonksiyonu çağrılır.

BES tipinden EST tipine dönüşüm yapan örnek kod bloğu:

Java:

```
byte[] inputBES = AsnIO.dosyadanOKU(BES_SIGNATURE_FILE);

BaseSignedData sd = new BaseSignedData(inputBES);


HashMap<String, Object> params = new HashMap< String, Object>();

//necessary for getting signaturetimestamp
params.put(EParameters.P_TS_DIGEST_ALG, DigestAlg.SHA1);
params.put(EParameters.P_TSS_INFO, getTSSettings());

ValidationPolicy policy=  PolicyReader.readValidationPolicy(new
FileInputStream(POLICY_FILE));


//necessary for validating signer certificate according to time of //
signaturetimestamp
params.put(EParameters.P_CERT_VALIDATION_POLICY, policy);

sd.getSignerList().get(0).convert(ESignatureType.TYPE_EST, params);

AsnIO.dosyayaz(sd.getEncoded(),CONVERTED_TO_EST_FILE);
```

C#:

```
byte[] inputBES = AsnIO.dosyadanOKU(BES_SIGNATURE_FILE);
BaseSignedData sd = new BaseSignedData(inputBES);
```

```
Dictionary<String, Object> parameters = new Dictionary< String, Object>();

//necessary for getting signaturetimestamp
parameters[EParameters.P_TS_DIGEST_ALG] = DigestAlg.SHA1;
parameters[EParameters.P_TSS_INFO] = getTSSettings();

ValidationPolicy policy;
using (FileStream fs = new FileStream(@"C:\policyBES.xml", FileMode.Open,
FileAccess.Read))
{
      policy = PolicyReader.readValidationPolicy(fs);
}

//necessary for validating signer certificate according to time of //
signaturetimestamp
parameters[EParameters.P_CERT_VALIDATION_POLICY] = policy;
sd.getSignerList()[0].convert(ESignatureType.TYPE_EST, parameters);

AsnIO.dosyayaz(sd.getEncoded(),CONVERTED_TO_EST_FILE);
```

7 Zaman Damgası

Zaman damgası, üzerinde bulunduğu verinin belirli bir tarihteki varlığını garantiler. Bizim uygulamamızda zaman damgası, oluşturduğunuz imzaya bağlı olarak oluşturulduğundan, imzanızın gerçekten o tarihte var olduğunu ispatlamak için kullanılır. Zaman damgası, güvenilir bir servis sağlayıcısı olan zaman damgası otoritelerinden alınır. Tüm ESHS'ler bu hizmeti vermektedirler.

İmza tarihinin önemli olduğu uygulamalarda zaman damgasının kullanımı büyük önem arz etmektedir. Çünkü zaman damgası olmayan bir imza üzerindeki zaman bilgisi, kullanıcının belirleyebildiği ve genelde kullanıcının sistem saatinden alınmış olan zaman bilgisidir. Dolayısıyla imzayı oluşturan kişi zaman bilgisini de istediği gibi belirleyebilir. Sertifika iptal durumlarında da imzanın sertifika iptal edilmeden önce atıldığından emin olunamaz.

Zaman damgası ise imzanın o tarihten (zaman damgası otoritesinin verdiği tarihten) önce var olduğunu garanti eder.

EST ve üzeri imza türleri zaman damgası içermektedir. API'nin zaman damgası alması için zaman damgası sunucusunun ayarlarının parametreler yardımıyla API'ye verilmesi gerekmektedir.

Java:

```
TSSettings tsSettings = new TSSettings("http://zd.ug.net", 21,
"12345678");
params.put(EParameters.P_TSS_INFO, tsSettings);
params.put(EParameters.P_TS_DIGEST_ALG, DigestAlg.SHA1);
```

C#:

```
TSSettings  tsSettings= new TSSettings("http://zd.ug.net", 21,
"12345678");
```

```
parameters[EParameters.P_TSS_INFO] = tsSettings;
parameters[EParameters.P_TS_DIGEST_ALG] = DigestAlg.SHA1;
```

Zaman damgası ayarlarından ilki zaman damgası adresi, ikincisi kullanıcı numarası, üçüncüsü kullanıcı şifresidir.

7.1 İmza Zamanı Alma

İmza zamanın alınabilmesi için imzanın zaman damgası içermesi gerekmektedir. Bunun için imza türünün en az EST olması gerekmektedir.

EST üzeri imza türleri *EST* sınıfından türediğinden *EST* sınıfının fonksiyonunu kullanabiliriz. Bu fonksiyondan dönen zaman *id_aa_signatureTimeStampToken* özelliğinden alınan zaman bilgisidir.

Java:

```
byte[] input = AsnIO.dosyadanOKU(SIGNATURE_FILE);
BaseSignedData bs = new BaseSignedData(input);
EST estSign = (EST)bs.getSignerList().get(0);
Calendar time = estSign.getTime();
```

C#:

```
Byte[] input = AsnIO.dosyadanOKU(SIGNATURE_FILE);
BaseSignedData bs = new BaseSignedData(input);
EST estSign = (EST)bs.getSignerList()[0];
DateTime time = estSign.getTime();
```

Eğer kullanıcının beyan ettiği imza saatine güveniliyorsa *AttributeOIDs.id_signingTime* özelliği kullanılabilir. Yalnız imzadaki *AttributeOIDs.id_signingTime* özelliğini zorunlu bir alan değildir, imza içinde bulunmayabilir.

Java:

```
byte[] input = AsnIO.dosyadanOKU(SIGNATURE_FILE);
BaseSignedData bs = new BaseSignedData(input);
List<EAttribute>  attrs = bs.getSignerList().get(0).getSignedAttribute
(SigningTimeAttr.OID);
Calendar time = SigningTimeAttr.toCalendar(attrs.get(0));
System.out.println(time.getTime().getTime().toString());
```

C#:

```
Byte[] input = AsnIO.dosyadanOKU(SIGNATURE_FILE);
BaseSignedData bs = new BaseSignedData(input);
List<EAttribute>  attrs = bs.getSignerList()[0].getSignedAttribute
(SigningTimeAttr.OID);
ETime time = new ETime(attrs[0].getValue(0));
Console.WriteLine(time.getTime().Value.ToString());
```

Profesyonel kullanıcılar *AttributeOIDs* sınıfında bulunan özelliklerle diğer zaman damgası bilgilerini de alabilirler. Örnek olarak arşiv tipi için kullanılan zaman damgası özelliğini kullanıldı:

Java:

```
byte[] input = AsnIO.dosyadanOKU(ESA);
BaseSignedData bs = new BaseSignedData(input);
List<EAttribute>  attrs = bs.getSignerList().get(0).getUnsignedAttribute
(AttributeOIDs.id_aa_ets_archiveTimestamp);
List<EAttribute>  attrsV2 = bs.getSignerList().get(0).getUnsignedAttribute
(AttributeOIDs.id_aa_ets_archiveTimestampV2);
attrs.addAll(attrsV2);
for (EAttribute attribute : attrs)
{
      Calendar time = ArchiveTimeStampAttr.toTime(attribute);
      System.out.println(time.getTime().toString());
}
```

C#:

```
byte[] input = AsnIO.dosyadanOKU(ESA);
BaseSignedData bs = new BaseSignedData(input);
List<EAttribute> attrs = bs.getSignerList()[0].getUnsignedAttribute
(AttributeOIDs.id_aa_ets_archiveTimestamp);
List<EAttribute> attrsV2 = bs.getSignerList()[0].getUnsignedAttribute
(AttributeOIDs.id_aa_ets_archiveTimestampV2);
attrs.AddRange(attrsV2);
foreach (EAttribute attribute in attrs)
{
   DateTime? time = ArchiveTimeStampAttr.toTime(attribute);
   Console.WriteLine(time.Value.ToString());
}
```

7.2 Zaman Damgası Sunucusunun Test Edilmesi

Zaman damgası ayarları verildikten sonra API zaman damgası alma işlemini kendisi yapmaktadır. Geliştiriciler zaman damgasını test etmek için aşağıdaki örnek kodu kullanabilirler. Zaman dagası işlemlerinden *TSClient* sınıfı sorumludur. Bu sınıf ile zaman damgası alınabilir, kalan kontör miktarı sorgulanabilir.

Java

```
byte [] sha1Digest = new byte [20];
Random rand = new Random();
rand.nextBytes(sha1Digest);

TSClient tsClient = new TSClient();
TSSettings settings  = new TSSettings("http://zd.ug.net", 21, "12345678".
toCharArray());
tsClient.setDefaultSettings(settings);

System.out.println("Remaining Credit: " + tsClient.requestRemainingCredit
(settings));

ETimeStampResponse response = tsClient.timestamp(sha1Digest, settings);
ESignedData sd = new ESignedData(response.getContentInfo().getContent());
ETSTInfo tstInfo = new ETSTInfo(sd.getEncapsulatedContentInfo().getContent
```

```
());

System.out.println("Time Stamp Time" + tstInfo.getTime().getTime());

System.out.println("Remaining Credit: " + tsClient.requestRemainingCredit
(settings));
```

7.3 Zaman Damgası Alma

ESYA kütüpanesini kullanarak sadece zaman damgası da alabilirsiniz. Bunun için asn1rt.jar, log4j.jar, ma3api-asn.jar, ma3api-common.jar, ma3api-crypto.jar, ma3api-crypto-gnuprovider.jar, ma3api-infra.jar dosyalarına ihtiyacınız vardır. Şu anda sadece SHA-1 özet algoritması desteklenmektedir.

Java

```
byte [] data = new byte [] {0,1,2,3,4,5,6,7,8,9};
byte [] dataTbs = DigestUtil.digest(DigestAlg.SHA1, data);
TSSettings settings = new TSSettings("http://zd.ug.net", 21, "12345678");
EContentInfo token = TSClient.getInstance().timestamp(dataTbs, settings).
getContentInfo();
```

C#

```
byte[] data = new byte[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
byte[] dataTbs = DigestUtil.digest(DigestAlg.SHA1, data);
TSSettings settings = new TSSettings("http://zd.ug.net", 21, "12345678");
EContentInfo token = TSClient.getInstance().timestamp(dataTbs, settings).
getContentInfo();
```

8 Parametreler

İmza atarken veya imza doğrulama yaparken uygulama geliştirme arayüzü çeşitli parametrelere ihtiyaç duymaktadır. Bu parametrelerin kullanımı imza türüne veya imza içerisinde yer alan özelliklere göre değişiklik göstermektedir. Kullanılabilecek ana parametreler açıklanacaktır. Kullanılabilecek diğer parametrelerin açıklamalarına *Eparameter* sınıfından ulaşabilirsiniz.

P_EXTERNAL_CONTENT: Ayrık imzanın doğrulaması yapılacaksa doğrulaması yapılacak veriye işaret eder. Atanacak nesne *Isignable* tipinde olmalıdır.

P_CONTENT_TYPE: İmzalanan içeriğin tip bilgisine işaret eder. Varsayılan değeri ise 'veri' anlamına gelen "1, 2, 840, 113549, 1, 7, 1" nesne belirtegeçidir (object identifier).

P_CERT_VALIDATION_POLICY: Sertifika doğrulamada kullanılacak olan politika bilgisine işaret eder. Eğer *P_VALIDATE_CERTIFICATE_BEFORE_SIGNING* parametresine *false* atanmışsa politika parametresinin atanmasına gerek yoktur. Diğer her durumda politika bilgisine ihtiyaç duyulur.

P_CHECK_QC_STATEMENT: Sertifikanın özellikleri arasında nitelikli ibaresinin (qualified certificate) aranıp aranmayacağını belirtmek amacıyla kullanılır. Varsayılan değeri ise *true*'dur. Eğer nitelikli ibaresinin kontrolünün yapılmaması isteniyorsa *false* atanmalıdır.

P_SIGNING_TIME: Doğrulama sırasında imzanın ne zaman atıldığını belirtmek için

kullanılır. Yalnız verilen zaman sadece zaman damgası bulunmayan imzalarda kullanılır. Örneğin BES tipinde bir imza 12.12.2010 tarihinde alındı ve alınma tarihi bir yere kaydedildi. Bu tarih daha sonra imzayı doğrularken kullanılabilir. Yalnız hukuki bir yükümlülüğü yoktur.

P_TRUST_SIGNINGTIMEATTR: İmza doğrulama sırasında kullanılır. İmzaya imzayı atan tarafından eklenen zaman bilgisine olan güveni belirler. *Boolean* tipinde nesne atanır; varsayılan değeri ise *true*'dur. Eğer kullanıcının eklediği zaman bilgisine güvenilmeyecekse *false* atanmalıdır.

P_TS_DIGEST_ALG: İmza atarken zaman damgası (Time Stamp) kullanılacaksa, *DigestAlg* tipinde bir nesne atanmalıdır. Varsayılan değeri *SHA1* algoritmasıdır.

P_TSS_INFO: İmza atarken zaman damgası (Time Stamp) kullanılacaksa, *TSSettings* tipinde bir nesne atanmalıdır. EST ve üzeri imza tiplerinde zaman damgası kullanılması mecburidir.

P_POLICY_ID: Birbirinden bağımsız iki taraf imzaların nasıl doğrulanacağı konusunda kendi aralarında çeşitli politikalar belirleyebilirler. İmzanın hangi politika ile doğrulanacağı bilgisini imzalanmış dosyaya eklemeleri gerekmektedir. Yalnız bu özelliğin kullanılabilmesi için imza türünün EPES veya daha üzeri bir imza türü olması gerekmektedir. EPES türü imzalarda varsayılan davranış olarak *SignaturePolicyIdentifierAttr* imzaya eklenmektedir. Diğer tür imzalarda kullanılabilmesi için *SignaturePolicyIdentifierAttr* imzaya eklenmesi gerekmektedir. *P_POLICY_ID* parametresi ise kullanılacak politika işaret eder.

P_POLICY_VALUE: *P_POLICY_ID* parametresinde anlatıldığı gibi bir politika belirleneceği zaman kullanılır. Kullanılacak politikanın kendisine işaret eder.

P_POLICY_DIGEST_ALGORITHM: *P_POLICY_ID* parametresinde anlatıldığı gibi bir politika belirleneceği zaman kullanılır. *SignaturePolicyIdentifierAttr* imzalanan bir özellik olduğundan, imzalanması sırasında kullanılacak özet bilgisini işaret eder. Varsayılan değer SHA-1'dir.

P_INITIAL_CERTIFICATES: İmzalanın doğrulanması için gerekli olabilecek sertifikaları işaret eder.

P_INITIAL_CRLS: İmzalanın doğrulanması için gerekli olabilecek ilerli işaret eder.

P_INITIAL_OCSP_RESPONSES: İmzalanın doğrulanması için gerekli olabilecek ÇisDuP cevaplarına işaret eder.

P_VALIDATE_CERTIFICATE_BEFORE_SIGNING: Varsayılan durumda imza atarken ve imza doğrulaması yaparken sertifika doğrulaması yapılmaktadır. BES ve EST türü imza atarken sertifika doğrulaması, bu parametreye *false* verilerek es geçilebilir. Yalnız bu kullanım önerilmemektedir.

P_REFERENCE_DIGEST_ALG: *SigningCertificateV1/V2, CompleteCertificateReferences ,CompleteRevocationReferences* özellikleri için kullanılacak özet algoritmasını belirler.

P_GRACE_PERIOD: Sertifika doğrulama verisinin kesinleşmesi için geçecek

kesinleşme süresini saniye olarak belirtir. Varsayılan değeri 86400 saniyedir (24 saat).

P_REVOCINFO_PERIOD: Sertifika doğrulama verisinin hangi süre aralığında toplanacağını saniye olarak belirtir. Örneğin bu süre 2 günü işaret ediyorsa; imzanın atıldığı tarihten 2 gün sonrasına kadarki sürede geçerli olan doğrulama verisi sertifika doğrulamada kullanılır. Bu süre en az *P_GRACE_PERIOD* süresi kadar olmalıdır. Eğer *P_REVOCINFO_PERIOD* parametresine herhangi bir süre verilmezse, bu süre mümkün olduğunca geniş tutulmaya çalışılacaktır. Bu sürenin geniş tutulması sertifika iptal durumlarının yakalanmasında bir dezavantaj getirmez. Bu aralığın kısa tutulması ise askı durumlarının yakalanma ihtimalini arttırır.

9 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.

9.1 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.

10 Döküman Versiyonları

Versiyon 0.1	Döküman yayınlandı.
Versiyon 0.2	BaseSigner arayüzü tr…api.common.crypto paketi altına taşındı.
Versiyon 0.3	Zaman damgası alınması ile ilgili ekleme yapıldı. İmza doğrulama ile ilgili ekleme yapıldı. ESYA API SmartCard Kullanım Kılavuzu'na referans eklendi.
Versiyon 0.4	Deneme lisansı ile çalışma bölümü eklendi. Ön doğrulama kavramı eklendi.
Versiyon 0.5	İmza tipleri eklendi.
Versiyon 0.6	C# örnekleri eklendi.
Versiyon 0.7	Giriş ve İmza Doğrulama Sonucu eklendi. Sertifika doğrulama bölümü İmza Doğrulama Bölümü altına kaydırıldı.
Versiyon 0.8	İsteğe bağlı alanlar eklendi.
Versiyon 0.9	.NET API örnekleri değişen metod imzalarina göre güncellendi.
Versiyon 1.0	SmartCardManager ve APDU işlemleri eklendi.

Versiyon 1.1	C# örnekleri güncellendi.