CMS TABANLI KÜTÜPHANE KULLANARAK ETSI UYUMLU ELEKTRONİK İMZA MODÜLÜ VE ÇEVRİMİÇİ UYGULAMA GELİŞTİRMEK

Hasan GÖLLE

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

TEMMUZ 2009
ANKARA

Hasan GÖLLE tarafından hazırlanan CMS TABANLI KÜTÜPHANE KULLANARAK ETSI UYUMLU ELEKTRONİK İMZA MODÜLÜ VE ÇEVRİMİÇİ UYGULAMA GELİŞTİRMEK adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

Prof. Dr. Şeref SAĞIROĞLU
Tez Danışmanı, Bilgisayar Mühendisliği ABD

Bu çalışma, jürimiz tarafından oy birliği ile Bilgisayar Mühendisliği Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir.

Prof. Dr. Sezai DİNÇER
Elektrik Elektronik Mühendisliği ABD, Gazi Üniversitesi
Prof. Dr. Şeref SAĞIROĞLU
Bilgisayar Mühendisliği ABD, Gazi Üniversitesi
Doç Dr. M.Ali AKÇAYOL
Bilgisayar Mühendisliği ABD, Gazi Üniversitesi

Tarih : 08/07/2009

Bu tez ile G.Ü. Fen Bilimleri Enstitüsü Yönetim Kurulu Yüksek Lisans derecesini onamıştır.

Prof. Dr. Nail ÜNSAL
Fen Bilimleri Enstitüsü Müdürü

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

Hasan GÖLLE

CMS TABANLI KÜTÜPHANE KULLANARAK ETSI UYUMLU ELEKTRONİK İMZA MODÜLÜ VE ÇEVRİMİÇİ UYGULAMA GELİŞTİRMEK

(Yüksek Lisans Tezi)

Hasan GÖLLE

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Temmuz 2009

ÖZET

İmzalanacak veri formatının, imzanın kendisinin ve elektronik belgenin geniş anlamda yaygın bir standardı yoktur. Bu da gerçek iş ortamlarında duruma uygun çözümlere, yetersiz özel yazılım uygulamalarına neden olmaktadır. Belgeleri elektronik olarak imzalamanın farklı özel yöntemleri vardır. Fakat farklı formatlarla uygunluk ve imzanın uzun dönemde geçerliliği sağlanmamaktadır.

Bu tezde, farklı standartlarda oluşturulan elektronik imzalı belgelerin uyumluluğunun sağlanmasına yönelik olarak incelemeler yapılmıştır. Uzun dönemli elektronik imza için önerilen Avrupa Standardı elektronik imza uygulaması geliştirmek için mimari araştırılmıştır. Elektronik imzalı belgenin yönetimi için kullanılan mimari tanıtılmıştır. İşlemlerin hızlı yapılabilmesi için bir elektronik imza kütüphanesi geliştirilmiştir. Önerilen metodun test edilmesi için çevrimiçi çalışan bir uygulama geliştirilmiştir ve geliştirilen işlem adımları tanıtılmıştır.

Sonuç olarak ETSI 101733 standardı ile uyumlu elektronik imza modülü ve bu modülü kullanarak çalışan çevrimiçi bir elektronik imza uygulaması başarıyla geliştirilmiş ve kullanıma sunulmuştur.

Bilim Kodu : **902.1.011**
Anahtar Kelimeler : **Elektronik imza, CMS, ETSI , imzalama sertifikası**
Sayfa Adedi : **116**
Tez Yöneticisi : **Prof. Dr. Şeref Sağıroğlu**

DEVELOPING ETSI COMPLIANT ELECTRONIC SIGNATURE MODULE AND ONLINE APPLICATION BY USING CMS-BASED LIBRARY

(M.Sc. Thesis)

Hasan GÖLLE

GAZİ UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY

July 2009

ABSTRACT

There is no common standart of data to be signed, the signature itself and electronic document in the broad sense. This causes ad hoc solutions and insufficient specific software applications in real business environments. There are special techniques of electronically signing documents. However, compatibility with different formats and long term signature validity is not available.

In this thesis, review of compatibility of electronically signed documents with different standards have been made. Recommended for long term electronic signatures, general architecture required for electronic signature application in European Standart is investigated. Electronic signature library is developed to process quickly. Online application is developed to test the proposed method and developed process steps are introduced.

As a result, the electronic signature module which is compatible with ETSI 101733 standard and online application running with that module are developed successfully and presented for usage.

Science Code : **902.1.011**
Key Words : **Electronic signature, CMS, ETSI, signing certificate**
Page Number : **116**
Adviser : **Prof. Dr. Şeref Sağıroğlu**

TEŞEKKÜR

Tezin hazırlanması sırasında beni yönlendiren ve bilgilerinden faydalandığım Prof. Dr. Şeref Sağıroğlu'na ve Muhammet Serdar Soran'a teşekkür ederim. Yüksek lisans yapmamda beni teşvik eden TÜBİTAK UEKAE yöneticilerine ve Ersin Gülaçtı'ya teşekkür ederim.

Çalışmayı gerçekleştirirken yükümü hafifletmek için bana her zaman özverili bir şekilde yardımcı olan eşim Reyhan GÖLLE'ye sonsuz teşekkür ederim.

İÇİNDEKİLER

Sayfa

Sayfa

Sayfa

Sayfa

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

Şekil **Sayfa**

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış bazı kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Kısaltmalar	Açıklama
AAA	Açık Anahtar Altyapısı
AES	İleri Elektronik İmza (Advanced Electronic Signature)
AIDA	İleri İnteraktif Sayısal Yönetim (Advanced Interactive Digital Administration)
API	Uygulama Programlama Arayüzü (Application Programming Interface)
ASN	Soyut Sözdizim Gösterimi (Abstract Syntax Notation)
ATL	Aktif Şablon Kütüphanesi (Active Template Library)
BER	Temel Kodlama Kuralları (Basic Encoding Rules)
BES	Temel Elektronik İmza (Basic Electronic Signature)
CAB	Cabinet Dosya Formatı
CadES	CMS İleri Elektronik İmza (CMS Advanced Electronic Signature)
CEN	Avrupa Standartlaştırma Komitesi (European Committee for Standardization)
CMS	Kriptografik Mesaj Sözdizimi (Cryptographic Message Syntax)
COM	Bileşen Nesne Modeli (Component Object Model)
CWA	CEN Çalışma Uzlaşması (CEN Workshop Agreements)
ÇİSDUP	Çevrimiçi Sertifika Durum Protokolü
DER	Seçkin Kodlama Kuralları (Distinguished Encoding Rules)
EESSI	Avrupa Elektronik İmza Standartlaştırma İnsiyatifi
EN	Avrupa Normu (European Norm)

Kısaltmalar	Açıklama
EPES	Açık Politikalı Elektronik İmza (Explicit Policy Electronic Signatures)
ES-C	Doğrulama Verisi Referanslı Elektronik İmza (Electronic Signature with Complete Validation Data)
ESHS	Elektronik Sertifika Hizmet Sağlayıcı
ESS	S/MIME için Geliştirilmiş Güvenlik Servisleri (Enhanced Security Services for S/MIME)
ES-T	Zaman Damgalı Elektronik İmza (Electronic Signature with TimeStamp)
ES-X	Genişletilmiş Doğrulama Verili Elektronik İmza (Electronic Signature with Extended Validation Data)
ETSI	Avrupa Telekomünikasyon Standartları Enstitüsü
HTML	Hyper Text Markup Language
IETF	İnternet Mühendisliği Görev Gücü (Internet Engineering Task Force)
JAI	Java İleri Görüntüleme Kütüphanesi (Java Advanced Imaging)
JVM	Java Sanal Makinesi (Java Virtual Machine)
KM	Kayıt Makamı
LDAP	Lightweight Directory Access Protocol
MSI	MSI Dosya Formatı
NM	Nitelik Makamı
OID	Nesne Belirteci (Object ID)
OSI	Açık Sistemler Bağlantısı (Open System Interconnection)
PDF	PDF Dosya Formatı
PIN	Personal Identification Number
PKCS	Açık Anahtar Kriptografi Standartları (Public Key Cryptography Standarts)

Kısaltmalar	Açıklama
RFC	Request For Comments
S/MIME	Güvenli/ Çok Amaçlı İnternet Posta Uzantıları (Secure / Multipurpose Internet Mail Extensions)
SİL	Sertifika İptal Listesi
SM	Sertifika Makamı
TSA	Zaman Damgası Makamı (TimeStamp Authority)
URI	Uniformed Resource Identifier
XML	Extensible Markup Language

1. GİRİŞ

Günümüzün ticari ortamında, bilgisayar tabanlı bilginin doğrulanması ve elektronik imzanın hukuksal altyapısının sağlanması amacıyla sistem oluşturmak için hem hukuksal hem de bilgisayar güvenliği alanında profesyonel yetkinliğe sahip olmak gerekir. Bu iki disiplini birarada kullanabilmek zordur. Bilgi güvenliği açısından bakılırsa sayısal imza, özel verilere uygulanan özel teknik işlemlerdir. Fakat hukuksal yönü daha geniş bir anlam ifade eder.

Sayısal imza ile elektronik imza arasındaki farkı da bilmek gereklidir. Sayısal imza, belgeyi alan kişiye o belgenin kaynağını ve bütünlüğünü ispatlayan ve sahtekarlıktan koruyan bir veridir. Bu veri, belgeye eklenebilir yada belgeyle mantıksal bir bağı vardır. Avrupa Elektronik İmza Direktifi, elektronik imzayı, "belgeye eklenen yada mantıksal bağ kurulan veri değil, kimlik doğrulama sağlayan elektronik veri" olarak tanımlar [1]. Taranmış ıslak imza yada elektronik postanın sonuna konan isim elektronik imza olabilir. Yani, sayısal imza, kriptografi ve özetleme tekniklerini kullanarak kimlik doğrulama, bütünlük ve inkar edememezlik sağlarken, elektronik imza ise sayısal imza kavramını da içine alan daha geniş bir kavramdır.

Sayısal imzanın matematiksel altyapısı iyi tanımlanmıştır ve açık anahtar altyapısının sağladığı kriptografik metodlar, gizlilik, bütünlük, inkar edememezlik ve kimlik doğrulamada etkin olarak kullanılabilmektedir. Sayısal imza için tanımlı genel standartlar (PKCS#7 veya CMS gibi) varsa da imzalanacak verinin formatını, imza formatını ve elektronik belge formatını tanımlayan bir standarda gereksinim vardır [2, 3]. Farklı ortamlarda uyumlu olarak çalışan elektronik imza çözümleri için uzun dönemli geçerliliği olan elektronik imza standardına ihtiyaç vardır.

ETSI 101733 elektronik imza standart dokümanı, uzun dönemli geçerliliğin önemli olduğu farklı işlem türleri için (satın alma, sözleşme, fatura uygulamaları gibi) elektronik imzayı ele almaktadır ve inkar etme durumunda geçerlilik kanıtlamayı da amaçlamaktadır [4].

Bu standart, kişiler ve şirketler arasında, iki şirket arasında, kişiler ve kamu kurumları arasında vs. tüm işlemler için kullanılabilir. Bu standart, tüm ortamlardan bağımsızdır ve tüm ortamlarda uygulanabilir. Akıllı kartlar, GSM SIM kartları, elektronik imza için özel programlar buna örnek olarak verilebilir .

Elektronik imza "Bir elektronik mesaj yada iletiye eklenen ve göndereni emsalsiz şekilde tanımlayan veya taklit edilmesi çok zor olan bir sayısal kod" şeklinde tanımlanmaktadır [5].

Farklı ülkeler ve iş alanları arasında uyumluluk için geniş kullanımı olan elektronik imza standardına sahip modüler bir altyapıya gereksinim vardır. Bu durumda özel amaçlar için hazırlanmış çözümlerin kullanılmasına gerek kalmayacaktır.

Tezimin 2. bölümünde uzun dönemli elektronik imza için önerilen Avrupa Standardı Elektronik İmza Yönetimi mimarisi açıklanmıştır. 3. bölümde ASN.1 BER ve DER yapılarına değinilmiş ve 4. bölümde ise ETSI 101 733 standardı kullanılarak geliştirilmiş bazı örnek uygulamalara yer verilmiş olup geliştirilmiş olan bir applet uygulaması da anlatılmıştır [6].

Bu tez çalışmasında Microsoft'un ürettiği CMS tabanlı bir kütüphane olan MS CryptoApi ile oluşturulan elektronik imzaların ETSI (Avrupa Telekomunikasyon Standartları Enstitüsü) 101 733 elektronik imza standardı ile olan uyumsuzlukları sunulmuştur [3, 4, 7].

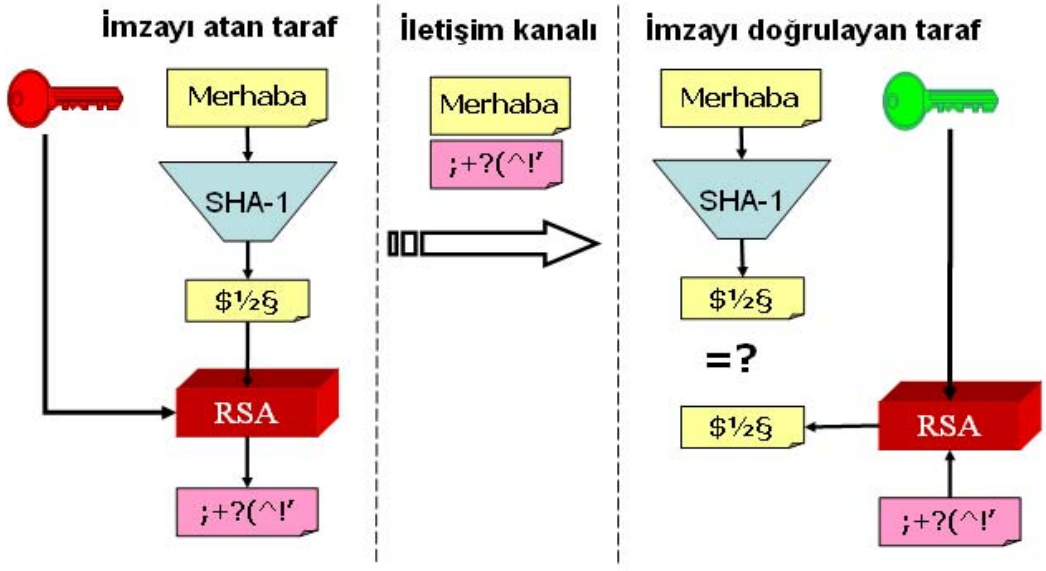
Bu uyumsuzluğu gidermek amacıyla yapılması gereken çalışmalar 6. bölümde açıklanmıştır. Bu amaçla BesImza.dll adında ETSI 101 733 uyumlu bir elektronik imza modülü geliştirilmiştir [4]. BesImza.dll elektronik imza modülünün sağladığı metodları kullanarak çalışan web tabanlı bir activeX uygulaması ve geliştirme adımları da 7. bölümde anlatılmıştır [8].

Son bölümde ise çalışma genel olarak değerlendirilmiş, sonuç ve öneriler sunulmuştur.

2. ELEKTRONİK İMZA

Elektronik imza, mesajın özeti ve imzalayan kişinin asimetrik özel anahtarının beraber kullanıldığı kriptografik bir dönüşümdür. Elektronik imzanın genel çerçevesi Şekil 2.1'de verilmiştir. Elektronik imza aşağıdaki özelliklere sahiptir;

- Mesajın sonuna eklenir.
- Mesaj alıcısının, mesajın bütünlüğünü kontrol etmesine ve mesajı gönderenin kimliğini doğrulamasına olanak tanır.
- İnkar edememezlik hizmeti sağlar.
- Asimetrik kriptografi (AAA-Açık anahtar altyapısı) kullanır.
- Islak imza gibi elektronik ortamlarda dokümanların imzalanmasını sağlar.
- Yüksek güvenlik sağlar.



Şekil 2.1 Elektronik imza genel çerçevesi

Veriyi alıp işleyen ve bu veriye özgü, sabit uzunlukta bir değer üreten algoritmalara mesaj özeti algoritması denir. Bu algoritmanın ürettiği değer, mesajın özetidir. En çok bilinen özetleme algortimaları MD5 ve SHA ailesidir. Mesaj özeti elde etmek için kullanılan fonksiyonların özellikleri aşağıda verilmiştir;

- Özetleme (parmakizi) fonksiyonları sabit uzunlukta sonuç üretirler. MD5 fonksiyonu 128 bit uzunluğunda, SHA-1 fonksiyonu ise 160 bit uzunluğunda özet değeri üretir.
- Mesajdaki küçük değişiklikler bile özette büyük değişikliklere neden olur.
- Özetleme fonksiyonları, kriptografik tek yönlü fonksiyonlardır. Mesaj özetinden asıl mesajı elde etmek çok zordur.

Sayısal imza formatlarını üçe ayırmak mümkündür: sarılmış imza (enveloped signature-imza imzalanan verinin içine gömülü), saran imza (enveloping signature-imzalanan veri imzanın içinde), ve ayrı imza (detached signature- imzalar imzalanan içerikten ayrı) [9].

Enveloped yada enveloping imzalar aynı dosyada bulunan veri üzerine oluşturulurken, detached imzalarda ise imza nesnesi, veri ile ayrı dosyalarda saklanır [9].

2.1. PKCS#7 / CMS

En önemli kriptografik standartlardan birisi, RSA veri güvenliği firmasının geliştirdiği PKCS#7'dir [2]. Bu standart, S/MIME eposta güvenliği, kredi kartı ödemeleri için (Secure Electronic Specification Transaction-SET) standartı yada gizli anahtar ve sertifikanın güvenli taşımasında kullanılan PKCS#12 standartlarında geniş kabul görmüş ve temel alınmıştır [10, 11].

PKCS#7 standardının evrimleşmiş hali, İnternet Mühendisliği Görev Gücü (Internet Engineering Task Force-IETF)'un S/MIME çalışma grubunun ürettiği ve sıradan bir mesajın sayısal imzalanması, özetinin alınması, doğrulanması yada şifrelenmesi amacıyla kullanılan Kriptografik Mesaj Sözdizimi (Cryptographic Message Syntax-CMS) standardıdır [2, 3]. Temelde CMS, elektronik verilere sayısal imza gibi kriptografik düzenlemeler eklemek için farklı kullanışlı biçimler önermektedir [3]. Standart, imzalama zamanı, mesaj içeriği ile birlikte doğrulanabilme, çoklu imza gibi niteliklere izin verir. Standart, PKIX çalışma grubunun tanımladığı X.509

tabanlı AAA gibi farklı sertifika tabanlı mimarilere destek verecek şekilde tasarlanmıştır. Tüm veri tipleri ASN.1 sözdizimine uygun olarak açıkça tanımlanmıştır ve kodlama (encode) kuralları (DER ve/veya BER kodlama nerede ve ne zaman kullanılır gibi.) da ayrıca belirlenmiştir [12, 13].

Herhangi bir içeriğe sayısal imza eklenmesi gerektiğinde CMS standardında bulunan imzalı veri (signed-data) içerik tipini paralel olarak çok sayıda imzacı kullanabilir [3]. İmzalanan veri formatı blobdur (binary large object- ikili veri kütlesi). CMS de belge eklerine sayısal imza desteklenmemiştir [3]. Mesaj kendi içinde imza doğrulama için gerekli sertifikaları içerebilir.

Standartla ortaya çıkabilecek olumsuz bir durum da içerikte imzalayan olmadığı durumda sertifikalar ve sertifika iptal listeleri (SİL) anlamsızlaşabilir [14]. Ayrıca veri ve imzasının imzalı veri içerik tipiyle birarada gruplanmamış ayrık imzalar da ortaya çıkabilir. Bu son durumda imza doğrulama işlemi uygulamaya bağımlıdır.

2.2. XML İmza

XML, tüm belge yada veriler için veri değişimi standardı olma potansiyeline, bilgi yapısını belirleyip tanımlama yetisine, ve tüm belgeler için verinin yapısını açıkça gösterme yetisine sahiptir [15]. Bu dillere örnek olarak; belgenin yapısını tanımlayan XML Schema, iletim amaçlı kullanılan XSLt ve stylesheetleri belirten XSL verilebilir [16-18]. XML imza, XML 1.0 meta-dili uygulamasıdır ve XML dahil tüm sayısal içeriklere (veri nesneleri) uygulanabilir [15, 19]. Veri nesnesinin özeti alınır, sonuç değeri diğer bilgilerle birlikte bir elemente konur, ve bu elementin özeti alınıp kriptografik olarak imzalanır. Sarılmış imza (enveloped signature) veya saran imza (enveloping signature), verinin üzerinde ve imza ile aynı XML belgesinde iken ayrık imza (detached signature) ise imza belgesinin dışında tutulur ve veri nesnesine URI'ler ile ilişkilendirilir [20]. Eğer imzalanan belge XML ile yazılmışsa XML imza, eklerin imzalanmasını da destekler.

2.3. PDF Açık Anahtar Sayısal İmza Çözümü

Adobe Acrobat 4.0 ile PDF belgelere sayısal imza ekleyebilme desteği verilmeye başlanmıştır [21]. Bu format, veriye gömülü imzayı (enveloped signature) desteklemektedir, böylece belgenin bir kısmını imzalayabilme ve çoklu imza özelliklerine sahiptir.

Bir kullanıcı bir belgeyi imzalamak istediğinde, öncelikle login olmalıdır. Acrobat, imza eklemeden önce imzalama nedeni, yerellik (locality) ve şifre gibi birtakım parametreler sorar. İmzalı belgeyi alan kişi gerekli sertifikalar belgede olduğu için imzayı doğrulayabilir. Fakat bu yeterli değildir çünkü imza geçerliliğinin de doğrulanması gereklidir, bu da sertifika parmakizi (fingerprint) ile belgeyi gönderenden güvenli yolla alınan sertifikanın parmakizlerinin karşılaştırılması ile yapılır. Eğer sonuç olumlu ise alıcı gönderen kişinin sertifikasını adres defterine ekler.

Güven modeli yalnızca küçük gruplar için uygundur ve belge değişimi yapan kullanıcı sayısının fazla olduğu kolektif kullanım için en iyi çözüm değildir. Ayrıca, AAA için güvenilir SM (sertifika makamı-certificate authority), sertifika iptali vs. gibi uyumlu eklentiler geliştirilmelidir. Bu şekilde ham imza (Raw Signature) ve PKCS#7 formatı ile sayısal imza oluşturabilir [2]. Son olarak, yalnızca PDF'e çevrilen belgeler imzalanabilir. Acrobat, blob (genel belge veya doküman) belgeler üstüne sayısal imzayı desteklemez. Bahsedilen imzalı veri ve sayısal imza formatları Çizelge 2.1' de özetlenmiştir.

2.4. Elektronik İmza Uygulamasının Modüler Çerçevesi

Elektronik imza uygulaması geliştirmek için dikkat edilmesi gerekli işlevsel ve temel gereksinimlere aşağıda yer verilmektedir.

Çizelge 2.1. İmzalanan veri ve sayısal imza nesnesi formatları

Format	Elektronik imza eklenebilir mi?	Çoklu imza desteği var mı?	Hangi format imzaya izin verir ?	Belge ekleri de İmzalana bilir mi ?	İmzalanan veri ile sayısal imza ilişkisi?
PDF 1.3	PKCS#7 kullanılıyorsa EVET	EVET	Yalnız PDF	Yalnızca PDF de	İmzalar veriye gömülü. Yalnızca enveloped signature oluşturulabilir.
XML imza	EVET	EVET	Blob	Yalnızca XML de	Enveloped, enveloping yada detached signature oluşturulabilir.
PKCS# 7/CMS	EVET	EVET	Blob	HAYIR	Enveloping yada detached signature oluşturulabilir.

2.4.1 İşlevsel gereksinimler

Islak imza gibi hukuksal bağlayıcılığı olan elektronik imza ile tüm işlemler için aşağıdaki gereksinimler karşılanmalıdır:

- Elektronik imzayı tanımlayıp anlamak için genel bir yöntem olmalıdır.
- Alan kişi mesajı kimin gönderdiğini anlayabilmelidir. (kimlik doğrulama).
- Mesaj bütünlüğünü doğrulayabilmenin bir yolu olmalıdır. Alan kişi mesajın değiştiğini, yönlendirildiğini yada tamamlanmadığını anlayabilmelidir. (veri bütünlüğü).
- Mahkemede bu işlemler ispatlanabilmelidir. Gönderen kişi mesajı gönderdiğini yada mesaj içeriğini inkar edememelidir. (inkar edilemezlik).

Elektronik belge yönetiminde elektronik imzanın yerini tanımlayabilmek için aşağıdaki gereksinimler belirlenmelidir:

- Uygulamanın tipine göre elektronik imzanın farklı hukuksal değerleri için destek sağlanmalıdır. Örneğin gerçek hayatta kitap satın almak için ıslak imzaya gerek

yoktur. E-ticaret işlemlerinde, benzer durumlarda elektronik imzanın ıslak imza ile aynı hukuksal değere sahip olma zorunluluğu olmayabilir.

- Elektronik imzanın anlamını tanımlayıp açıklamak gerekir. Yani kişinin organizasyon içindeki rolüne uygun olarak elektronik imzanın yönetilmesi, çoklu imza uygulanabilmesi yada uzun dönemde elektronik imzanın doğrulanabilmesi vs. gibi kavramlar belirlenmeli ve açıklanabilmelidir.
- İmzalama zamanı kabul edilebilir ve iyi tanımlanmış şekilde belirlenmelidir.
- Belge bölümlerinin elektronik imzalanması da opsiyonel olarak desteklenmelidir.

2.4.2. Temel gereksinimleri

Elektronik imza uygulamalarını gerçekleştirmek için birtakım proje gereksinimlerinin karşılanması gerekir.

Gereksinimlerden biri, sayısal imza mekanizmasının AAA (açık anahtar altyapısı) üzerine kurulu olmasıdır. Bu sayede kimlik doğrulama, bütünlük, inkar edememezlik, zaman damgası gibi işlevsel gereksinimleri karşılanır.

Bir diğer gereksinim, geniş anlamda tanınan standart formatların tanımlanması, kullanılması ve elektronik imzaya adapte edilmesidir. Böylece birlikte çalışabilirlik sağlanmış olur. İmzalanacak veri formatının elektronik imzadan bağımsız olmasının yolu budur. Adobe gibi özel teknolojiler için çözümler, yalnızca kendi sayısal imzalarını anlayıp gerçekleştirme yetisine sahiptirler ve bu da elektronik imza gerçekleştirimi için tatmin edici değildir. İmza formatlarının temel karakteristikleri arasında aşağıdaki ayrım yapılabilir:

- Adobe PDF 1.3 formatında olduğu gibi imzaların verinin içinde olduğu durumda *enveloped signature* desteği verilmelidir. Aynı şekilde, veri ve imzayı birlikte içeren tek nesneyi bulmak gerektiği durumlarda *enveloping signature* desteği verilmesi gerekir. Benzer olarak arşivleme ve “full text” sorgular ile veri getirme

işlemi için ideal olan, imzalı veri ve imzanın ayrı tutulduğu *ayrık imza* için de destek verilmelidir.

- Uzun dönemde doğrulama için gerekli veri yapılarının tanımlanması gerekir. Ayrıca, bu veri yapılarının, imza nesnesine SİL ve ÇİSDUP cevap verisi ve referansının ve zaman damgalarının eklenebilmesine izin vermesi gerekir [14, 22].
- Bilgisayar korsanlığına karşı koruma sağlayacak veri yapılarının tanımlanması gerekir. Elektronik imza üretmek ve doğrulamak için elektronik imza açık anahtar sertifikası kullanılmalı ve güvenilmeyen sertifikalar elektronik imza amaçlı kullanılmamalıdır.
- Çoklu imza desteği verilmelidir. (paralel ve seri imzalar).
- İmzanın anlamını kavramaya yarayacak veri yapılarının tanımlanması gerekir. Örneğin kullanıcı rolü yada imzalama zamanı, sayısal imza yapısına eklenebilir.

İmza politikası tanımı ve yönetimi de diğer bir önemli tasarım gereksinimidir. İmza politikası, elektronik imza tarafından karşılanması gereken kurallar kümesini (sözdizim, encoding-kodlama formatları, protokollar, ve diğer özel bilgiler) tanımlar ve imzalama esnasında imzalayan tarafından gönüllü olarak kabul edilmiş sayılır [4].

Son olarak "gördüğünü imzala-what you see is what you sign" probleminin yönetilmesi gerekir. Kullanıcı neyi imzalayacağının ve imza doğrulamanın yapıldığı metnin içeriğinin farkında olmalıdır [23].

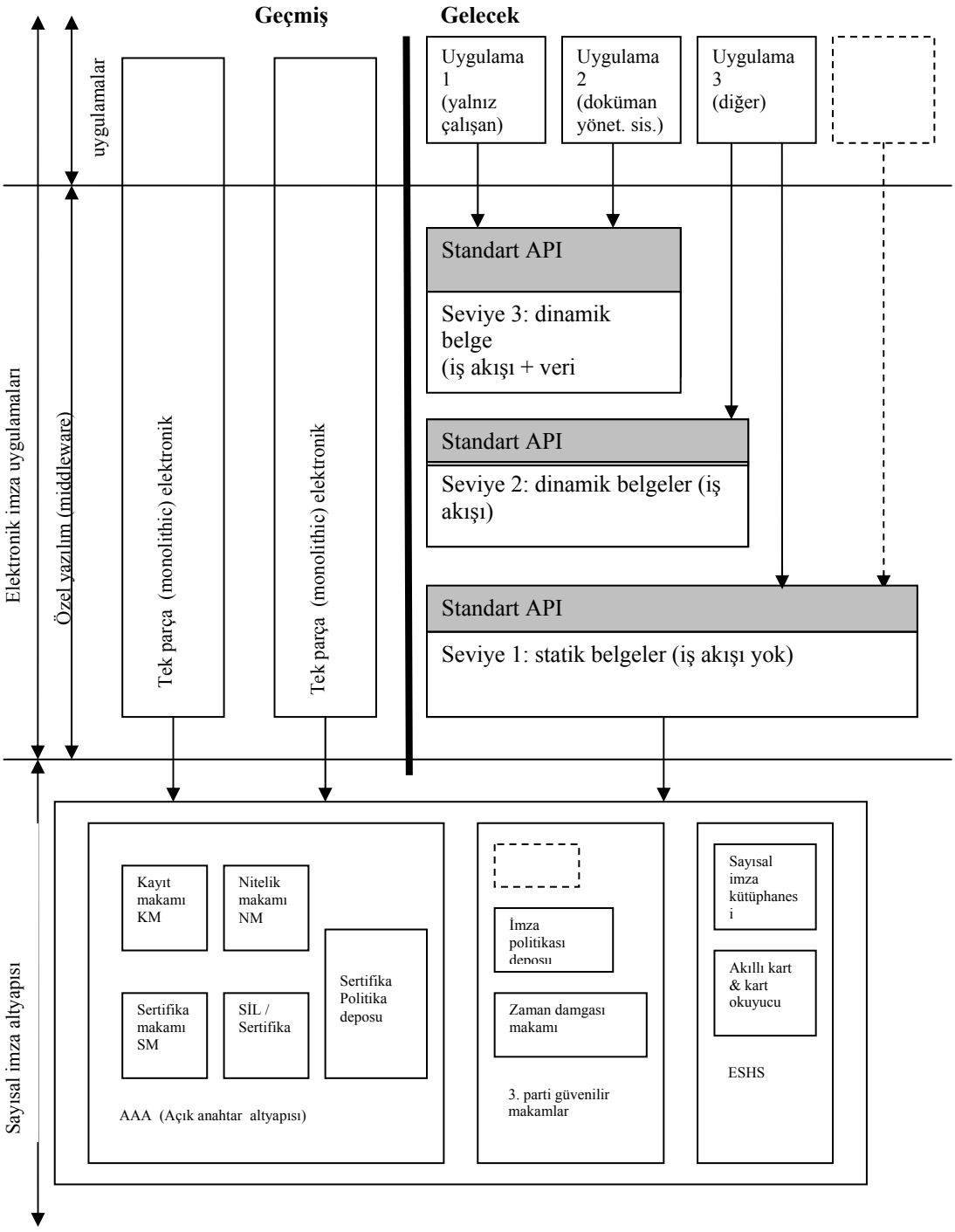
2.5. Elektronik İmza Uygulamaları Geliştirmek İçin Genel Mimari

Belge yönetimi için elektronik imza uygulaması gerçekleştirirken sayısal imza gereksinimlerine ve elektronik imzaya özgü gereksinimlere uyumluluk içinde bir sistem tasarlanmalıdır. Sayısal imza gerçekleştiriminde gerekli modüler altyapı için AAA, üçüncü parti güvenilir makamlar (Trusted Third Party) ve ESHS (Elektronik Sertifika Hizmet Sağlayıcı) gereklidir.

Şekil 2.2'de "Geçmiş" bölümünde görüldüğü gibi, önceleri gerçek elektronik imza geliştirmek için gerekli mimari tanımlanmadığı için tescilli ve belirli bir amaca yönelik çözümler kullanılmaktaydı.

Birlikte çalışabilir elektronik imza uygulamaları geliştirmek için genel, geniş kapsamlı ve modüler bir çerçevenin belirlenmesi gerekir ve bu çerçevenin de sayısal imza altyapısını ve uygulama alanını da kapsayan geniş tabanlı bir standart üzerinde oluşturulması gereklidir (Şekil 2.2 "Gelecek" bölümü).

Çok katmanlı bir mimari ile geliştiricilere gerekli sayıda soyutlama seviyesi verilirse, bu sayede elektronik imza gerçekleştirimi ile ilgili detaylar atlanıp uygulamanın mantığı üstünde yoğunlaşmak mümkün olabilir. Üstte bahsedilen çerçeve, elektronik imza uygulamalarının geliştirilmesi sürecini basitleştirecek ve elektronik imzalı belgelerin yönetilmesini kolaylaştıracaktır. Şekil 2.2'de, karakteristiği yukarda verilen genel mimari gösterilmektedir.



Şekil 2.2. Genelleştirilmiş elektronik imza geliştirme mimarisi

2.6. Sayısal İmza Altyapısı Tasarım Gereksinimleri

Sayısal imza mekanizması, açık anahtar altyapısına ve özetleme (hashing) tekniklerine dayanır. Bu tarz kriptografiyi kullanabilmek için açık anahtar ile iyi tanımlanmış bir varlığı, onaylanmış ve belirsiz olmayan bir yolla ilişkilendirmek gerekir. Bu ilişki için kullanılan veri yapısı açık anahtar sertifikasıdır. Açık anahtar sertifikası, sertifika makamının (SM) imzasını içerir. Sertifika makamı, çok daha karmaşık ve hiyerarşik bir yapı olan AAA (Açık Anahtar Altyapısının) bir parçasıdır. AAA, açık anahtar sertifikalarının oluşturulması, yönetilmesi, saklanması, dağıtılması ve iptal edilmesi için gerekli tüm donanım cihazları, yazılım, kullanıcılar, politikalar ve prosedürleri içerir [24].

Önemli AAA kavramlarına , SM (sertifika makamı-certificate authority), KM (kayıt makamı-registration authority), NM (nitelik makamı-attribute authority), sertifika politikaları, sertifika havuzları ve sertifika iptal listeleri (SİL), ÇİSDUP (Çevrimiçi Sertifika Durum Protokolü) örnek verilebilir [14, 22].

Sayısal imza sisteminin diğer önemli bileşenleri ise, Elektronik Sertifika Hizmet Sağlayıcılarıdır (ESHS). ESHS'ler, veri nesnelerinin imzalanmasında kullanılan matematiksel algoritmaları gerçekleştirmek için yazılım (sayısal imza kütüphanesi ve API) ve donanım (akıllı kartlar ve kart okuyucular) çözümleri sunarlar.

Elektronik imza gerçekleştirimi için sayısal imza altyapısı en az iki bileşenle genişletilmelidir. Bunlar, zaman damgası makamı (Time stamp authority- TSA) ve imza politikaları havuzu. İlki, imzalama zamanını onaylayan güvenilir bir hizmet sağlayıcıyı makamdır, diğeri ise elektronik imza oluşturulurken farklı imza politikalarının alınabileceği bir yerdir.

2.7. Elektronik İmza Uygulamaları Alanı İçin Tasarım Gereksinimleri

Elektronik belgelerle yapılan işlemleri üç başlıkta toplayabiliriz.

- Süreçlere dahil olmayan statik belgeler (iş akış yok).

- Yalnızca belge işlemleriyle bağlantılı durum değişimleri olan dinamik belgeler (iş akışı)
- Belge işlemleri ve farklı bilgi havuzları ile bilgi değişimi ile ilgili durum değişimleri olan belgeler. (iş akışı + harici veri tabanlarına erişim)

Şekil 2.2'de gösterilen çok katmanlı mimaride özel yazılım (middleware) bölümünde, yukarıda bahsedilen her elektronik belge için özel modül tanımlanmıştır. Farklı modüllerin gerçekleştirimi için imza formatı ile ilgili problemlerin çözülmesi ve imzalama politikasının seçimi yada zaman damgası ekleme gerekliliği gibi konuları da dikkate alarak elektronik imza nesnesinin oluşturulup doğrulanmasının tanımlanması gereklidir. 5.1'de (pratik uygulamalar) statik belgelerin işlenmesi ile sınırlı (Şekil 5.2) bir model anlatılmıştır. Elektronik imza formatı için ETSI 101733 standardı, belirtilen tüm gereksinimleri karşılamaktadır [4].

2.8. Avrupa Birliği Elektronik İmza Direktifi (EESSI) ve ETSI

Elektronik ticaretin gelişiminin hızlanması amacıyla Avrupa Parlementosunda 13 Aralık 1999 tarihinde Avrupa Birliği Elektronik İmza Direktifi kabul edilmiştir [1].

Avrupa Elektronik İmza Standart İnsiyatifi (EESSI), elektronik imzanın özellikle iş alanında, gelecekteki standartlaşma ihtiyaçlarını belirlemeyi hedef almıştır [1]. Elektronik imza alanındaki tüm yerel insiyatifler ve tüm teknolojik standartlar biraraya getirilmiştir. Standart belirlemenin gerekli olduğu tüm alanlar belirlenmiştir.

ETSI SEC ESI çalışma grubunun geliştirdiği standartda, elektronik sertifika hizmet sağlayıcıların (ESHS) güvenlik yönetimi ve sertifika politikaları özellikleri, sözdizimi tanımları, imza ve imza politikası formatları, nitelikli X509 sertifikaları özellikleri ve zaman damgası protokolü ve formatları vardır. CEN/ISS E-SIGN çalışma grubunun geliştirdiği standartda ise nitelikli sertifika yayımlayan elektronik sertifika hizmet sağlayıcıların (ESHS) güvenlik gereksinimleri, imza oluşturma

cihazlarının güvenlik gereksinimleri, elektronik imza geliştirmenin kullanıcı arayüzü belirtimleri, imza doğrulama gereksinimleri ve uygunluk değerlendirmeleri belirtilmiştir.
Son olarak ETSI ve CEN üyelerinin EESSI gözetimi altında ürettikleri standartlar geliştirilmiş ve Avrupa Normu (EN) haline gelmiştir. İmza formatları alanında ilk standart Mart 2000 de kabul edilmiş ve Mayıs 2000 de yayımlanmıştır.

Avrupa Elektronik İmza Standart İnsiyatifi Direktifinde kimlik doğrulama, inkar edememezlik ve bütünlük için ileri elektronik imza (Advanced Electronic Signature-AES) kapsanmaktadır [1]. Ayrıca ıslak imza ile aynı hukuki geçerliliği olan "nitelikli" elektronik imza da tanımlanmıştır. Nitelikli elektronik imzalar nitelikli sertifikalarla ve güvenli imzalama cihazlarıyla oluşturulmaktadır.

Avrupa Elektronik İmza Standart İnsiyatifi Direktifinde ayrıca elektronik sertifika hizmet sağlayıcıların (ESHS) karşılaması gereken gereksinimler, imzalayan ve doğrulayan gereksinimleri de belirtilmiştir [1]. Bu gereksinimlerin detaylı standartlarla desteklenmesi ve iş hayatının gereksinimlerini de karşılaması gerekir.

3. ASN.1 , BER ve DER

Günümüzün en karmaşık ve geniş anlamda soyutlama gerektiren sistemlerinden biri, X.200 de tanımlı OSI (Open Systems İnterconnection-Açık sistemler bağlantısı)'dır [25]. OSI, bilgisayarlar arasında fiziksel katmandan kullanıcı uygulamaları katmanına kadar bağlantıları yöneten uluslararası standartlaşmış bir mimaridir. Üst katmandaki nesneler soyut olarak tanımlanmıştır ve alt katmandaki nesnelerle birlikte gerçeklenmesi öngörülmüştür. OSI, her katmandaki çok farklı uygulamalara destek verir.

OSI, soyut nesneleri belirtmek için X.208'de tanımlı ASN.1 (Abstract Syntax Notation) metodunu kullanır [26]. Bu nesneleri '1' ve '0'lar dizgisi şeklinde tanımlamak için kullanılan ve X.209'da tanımlı kurallar kümesine BER (Basic Encoding Rules) denir [27]. ASN.1 esnek bir gösterim şekline sahiptir. ASN.1 kullanan biri çok farklı türde veri tipleri tanımlayabilir. Bunlar basit tipler de olabilir, kümeler ve diziler gibi bileşik de olabilir, başka tiplerle tanımlanan karmaşık tipler de olabilir. BER ise ASN.1 tip değerlerinin 8 bitlik oktetlerle nasıl kodlanacağını gösteren kurallar kümesidir. Verilen bir değeri BER ile kodlamanın genelde birden fazla şekli vardır. BER'in bir alt kümesi olan DER (Distinguished Encoding Rules) ise her ASN.1 değerinin tek bir oktet kodlamasını verir. Bu hususlar farklı başlıklar altında detaylı olarak açıklanacaktır.

3.1. ASN.1

ASN.1 (Abstact Syntax Notation), soyut tip ve değerleri belirtmek için kullanılan bir sözdizim kuralıdır. ASN.1 de veri tipi bir değerler kümesidir. Bazı tipler için sonlu sayıda değer varken bazıları içinse sonsuz sayıda değer vardır. ASN.1 de 4 temel veri tipi vardır [28].

- Basit tipler; "atomik"dir ve bileşeni yoktur.
- Bileşik tipler; "Structured", bileşenleri vardır.
- Etiketli tipler; "Tagged"; diğer tiplerden elde edilmiştir.

- Diğer tipler; “CHOICE” ve “ANY” .

Tip ve değerlere ASN.1 atama operatörü (::=) ile isim verilebilir ve bu isimler diğer tip ve değerleri tanımlamada kullanılabilir. CHOICE ve ANY dışında tüm ASN.1 tiplerinin bir etiketi “tag” vardır. Etiket, sınıf numarası ve ‘0’ dan büyük etiket numarasından oluşur. ASN.1 tipleri yalnızca etiket numaraları aynıysa birbirinin aynıdır. Yani ASN.1 tipinin adı, soyut anlamını ifade etmez. 4 tane etiket sınıfı vardır [28].

- Evrensel (universal), tüm uygulamalarda aynı anlam taşıyan tipler içindir; Bu tipler yalnız X.208’de tanımlıdır [26]. Çizelge 3.1’de bazı evrensel veri tipleri görülmektedir.
- Uygulama (Application), uygulamaya özel anlam taşıyan tipler içindir.
- Özel (Private), projeye özel anlam taşıyan tipler içindir.
- Bağlam-özel (Context-specific), Verilen birleşik tipe özel anlam taşıyan tipler içindir.

Çizelge 3.1. Bazı evrensel sınıfı tipler ve etiketleri

Tip	Etiket numarası (decimal)	Etiket numarası (hexadecimal)
Integer	2	02
Bit string	3	03
Octet string	4	04
Null	5	05
Object identifier	6	06
Sequence & sequence of	16	10
Set & set of	17	11
PrintableString	19	13
T61String	20	14
IA5String	22	16
UTCTime	23	17

ASN.1'de veri tipleri ve değerleri, esnek programlama diline benzer bir şekilde ve aşağıdaki kurallara uygun olarak gösterilir:

- Yerleştirme çok önemli değildir. Çok sayıda boşluk yada yeni satır tek bir boşluk olarak düşünülür.
- Yorumlar (--) kısa çizgi çiftleri ile gösterilir.
- Kimliklendiriciler (alan ve değer adları) ve veri tipi referansları, büyük ve küçük harfler, rakamlar, kısa çizgiler içerebilir. Kimliklendiriciler küçük harfle başlar, veri tipi referansları ise büyük harfle başlar.

3.1.1. Basit tipler

Bileşeni olmayan "atomik" tiplerdir. PKCS standartları ile ilgili olanlar Çizelge 3.2 de gösterilmektedir.

Çizelge 3.2. Basit tipler

Basit Tipler	Açıklama
Integer	Tamsayı
Bit string	Rastgele bit dizisi (0 ve 1 ler)
Octet string	Rastgele oktet dizisi (8-bit değer)
Null	Boş değer
Object identifier	Algoritma yada nitelik tipi gibi, bir nesneyi tanımlayan tamsayı dizisi.
PrintableString	Rastgele basılabilir karakter dizisi.
T61String	Rastgele T.61 (8-bit) karakter dizisi
IA5String	Rastgele IA5 (ASCII) karakterleri dizisi
UTCTime	Greenwich Ortalama Zamanı (GMT)

Basit tipler ikiye ayrılır. Dizgi (String) tipleri ve dizgi olmayan tipler. BIT STRING, IA5String, OCTET STRING, PrintableString, T61String ve UTCTime dizgi tipleridir.

Dizgi (string) tipleri, kodlama (encoding) amaçlı olarak altdizgi (substring) bileşenlerine ayrılabilir. Böylece uzunluğu bilinemeyen değerlerin (örneğin dosyadan oktet stringe aktarılan değer) kodlaması mümkün olur. Dizgi tiplerinin büyüklüğü sınırlandırılabilir.

3.1.2. Bileşik (structured) tipler

Bileşik tipler bileşenleri olan tiplerdir. ASN.1, PKCS standartları ile ilişkili 4 bileşik tip tanımlar.

Çizelge 3.3. Bileşik tipler

Bileşik tipler	Açıklama
Sequence	Bir yada daha çok tipin düzenli birleşimi
Sequence of	Verilen bir tipin 0 yada daha fazla sayıda düzenli olarak bir araya getirilmesi
Set	Bir yada daha çok tipin düzensiz birleşimi
Set of	Verilen bir tipin 0 yada daha fazla sayıda düzensiz olarak bir araya getirilmesi

Bileşik tiplerin isteğe bağlı (optional) bileşenleri olabilir.

3.1.3. Dolaylı yada açık etiketli tipler

Etiketleme, uygulama içinde tipleri ayırt etmek için ve bileşik tip içinde bileşenleri ayırt etmek için kullanılır. Etiketleme, dolaylı yada açık olmak üzere 2 şekilde yapılır.

- Tiplerin etiketleri değiştirilerek dolaylı etiketlenmiş tipler elde edilir. Dolaylı etiketleme, [sınıf numarası] IMPLICIT ile gösterilir.
- Tiplere harici etiket ekleyerek açık etiketlenmiş tipler elde edilir. Aslında açık etiketlenmiş tipler bir bileşeni olan bileşik tiplerdir. Açık etiketleme, [sınıf numarası] EXPLICIT ile gösterilir.

3.1.4. Diğer tipler

Diğer tipler CHOICE ve ANY tipleridir. CHOICE tipi bir yada daha fazla alternatifin birleşimini ifade eder. ANY tipi rastgele bir tipin rastgele bir değerini gösterir.

3.2. BER (Basic Encoding Rules)

BER (Basic Encoding Rules), ASN.1 değerini oktet dizgisi olarak bir yada daha fazla şekilde ifade etmeye yarar. ASN.1 değerini BER ile kodlamanın veri tipinin değerine ve değerin uzunluğuna bağlı olarak 3 farklı metodu vardır. Bunlar;

- İlkel (primitive), sabit uzunlukta kodlama (encoding).
- Bileşik (constructed), sabit uzunlukta kodlama.
- Bileşik, değişken uzunlukta kodlamadır.

Basit, dizgi olmayan tipler, ilkel ve sabit uzunlukta kodlama kullanır. Bileşik tipler bileşik metodlardan birini kullanır. Basit, dizgi olan tipler uzunluğun bilinip bilinmemesine bağlı olarak metodlardan herhangi birini kullanabilir.

Her kodlama metodunda BER kodlamanın 3 yada 4 parçası vardır:

Belirleyici (Identifier) oktetler: Bunlar, ASN.1 değerinin sınıf ve etiket numarasını ve metodun ilkel mi yada bileşik mi olduğunu belirler.

Uzunluk (Length) octetler: Sabit uzunluktaki metodlar için içerik oktetlerinin sayısını verir. Bileşik, değişken uzunlukta metod için uzunluğun değişken olduğunu belirtir.

İçerik (Content) octetler: İlkel, sabit uzunlukta metod için değerin gerçek gösterimini verir. Bileşik metodlar için ise bileşenlerin BER kodlamalarının birbirine bağlanmış şeklini verir.

İçerik oktetlerinin sonu: Bileşik, değişken uzunlukta metod için içerik oktetlerinin sonunu gösterir.

3.3. DER (Distinguished Encoding Rules)

DER (Distinguished Encoding Rules), BER'in altkümesidir ve herhangi bir ASN.1 değerini oktet dizgileri ile ifade etmenin tek bir yolunu verir. ASN.1 değeri üzerinde oluşturulan, sayısal imzada olduğu gibi tek bir oktet dizgisi kodlaması gerektiren durumlarda kullanılır [13].

BER de verilen kurallara DER ile aşağıdaki eklemeler yapılır:

- Uzunluk (length), 1 ile 127 arasında ise uzunluğun kısa formu kullanılmalıdır.
- Uzunluk 128 yada daha büyükse uzunluğun (length) uzun (long) formu kullanılmalıdır ve uzunluk minimum sayıda oktet ile kodlanmalıdır.
- Basit dizgi tipleri için ilkel, sabit uzunlukta metod kullanılmalıdır.
- Bileşik tipler için bileşik, sabit uzunlukta metod kullanılmalıdır.

3.4. Örnek

X.501'de tanımlı Name tipinin ASN.1 gösterimi ve DER kodlaması örnek olarak verilebilir [29].

3.4.1. ASN.1 gösterimi

X.501'de tanımlı Name tipinin ASN.1 gösterimi aşağıdaki gibidir [29];

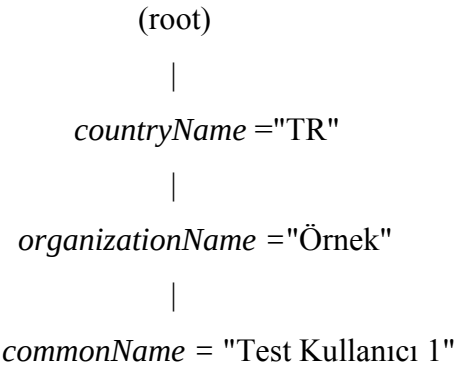
```
Name ::= CHOICE {
  RDNSequence }

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::=
  SET OF AttributeValueAssertion

AttributeValueAssertion ::= SEQUENCE {
   AttributeType,
   AttributeValue }

AttributeType ::= OBJECT IDENTIFIER

AttributeValue ::= ANY
```

Name, RDNSequence alternatifi içeren CHOICE tipinde bir nesnedir. RDNSequence, bir yada daha fazla sayıda RelativeDistinguishedName içerir. RelativeDistinguishedName, 0 yada daha fazla sayıda AttributeValueAssertion içerir.

AttributeValueAssertion tipinin iki bileşeni AttributeType ve AttributeValue'dur. AttributeType nesne belirteci ile nitelik belirtir. AttributeValue ise rastgele bir nitelik değeri gösterir.

3.4.2. DER kodlama

Name tipi Şekil 3.1'deki gibi gösterilebilir.



Şekil 3.1. X.501 de tanımlı Name tipinin gösterimi

Her seviye bir RelativeDistinguishedName değerine karşılık gelir ve bunların da her biri AttributeValueAssertion değerine sahiptir. AttributeTypevalue, eşittir (=) işaretinden önce gelir ve AttributeValue ise sonra gelir.

CountryName, organizationName, ve commonUnitName, X.520'de ASN.1 gösterimi aşağıdaki gibi tanımlı nitelik tipleridir [30]:

```
attributeType OBJECT IDENTIFIER ::=
  { joint-iso-ccitt(2) ds(5) 4 }
```

countryName OBJECT IDENTIFIER ::= { attributeType 6 }
organizationName OBJECT IDENTIFIER ::=
{ attributeType 10 }

commonUnitName OBJECT IDENTIFIER ::=
{ attributeType 3 }

4. ETSI ELEKTRONİK İMZA STANDARDI (CaDES)

Bu bölümde, ETSI tarafından önerilen uzun dönem geçerli elektronik imza formatları anlatılmaktadır [4]. Elektronik imzalı belgelerde imzalayan yada doğrulayan tarafın inkarı durumunda imza geçerliliğini kanıtlamak mümkün olmalıdır. Standartta, elektronik sertifika hizmet sağlayıcıların (ESHS) sağlaması gerekli olan özellikler ve uzun dönemli doğrulama için gerekli olan arşiv verilerinin (örn. çapraz sertifikalar ve iptal listeleri) özellikleri de belirtilmektedir.

Elektronik imza, imzalayanla doğrulayan arasında uzun yıllar sonra bile oluşabilecek anlaşmazlıklarda kullanılabilir. ETSI elektronik imza standartında, elektronik imza üretmede açık anahtar altyapısı temel alınmıştır ve kullanılan yazım kuralı olarak ASN.1 (Abstract Syntax Notation 1) belirtilmiştir [4]. Bu format, CMS (Cryptographic Message Syntax) tabanlı olup RFC 3852'de tanımlıdır [3]. Bu elektronik imzalar, bu nedenle CadES (CMS Advanced Electronic Signature-CMS İleri Elektronik İmza) olarak adlandırılır. Bu standartta CMS (RFC 3852) tabanlı Elektronik imza formatları tanımlanmıştır [3]. Bu bağlamda temel olarak aşağıdaki tanımlara yer verilmiştir:

- İmzalayan
- Doğrulayan
- Elektronik sertifika hizmet sağlayıcısı (ESHS)
- Hakem

Hakem, imzalayanla doğrulayan arasında anlaşmazlık çıkması durumunda hakemlik yapan varlıktır. Elektronik sertifika hizmet sağlayıcılar (ESHS), imzalayanla doğrulayan arasında güven ilişkisi sağlamaya yarayan varlıklardır. Bu güven ilişkisini aşağıdaki hizmetleri sürdürerek sağlarlar:

- Kullanıcı sertifikaları.
- Çapraz sertifikalar.
- Zaman damgası.

- SİL (Sertifika İptal Listesi) sorguları [14].
- ÇİSDUP (Çevrimiçi Sertifika Durum Protokolü- Online Certificate Status Protocol) sorguları [22].

Bu hizmetler aşağıdaki güvenilir servis sağlayıcıları vasıtasıyla sürdürülür:

- Sertifika makamı, SM
- Kök sertifika makamı
- Kayıt makamı, KM
- SİL yayımlayıcı
- ÇİSDUP yayımlayıcı
- İptal makamı
- Zaman damgası makamı
- İmza politikası yayımlayıcı

4.1 Elektronik İmza Formatları

ETSI 101733 elektronik imza standartında elektronik imzaya eklenen imzalı yada imzasız niteliklerin türüne göre elektronik imza formatları tanımlanmıştır [4]. Temel elektronik imza (BES-Basic electronic signature), sayısal imzayı ve imzacının sağladığı diğer temel bilgileri içerir, temel kimlik doğrulama, bütünlük koruma sağlar ve çevrimiçi (online) servislere (zaman damgası gibi) erişilmeden de oluşturulabilir. Zamandamgalı elektronik imza (ES with Timestamp, ES-T), imzaya zaman damgası ekler ve uzun dönemli doğrulamanın ilk adımını oluşturur. Doğrulama referanslı elektronik imza (ES with complete validation data, ES-C), elektronik imzanın geçerliliğini belirten veriyi ES-T ye ekler. Bu veri, sertifika referansı ile birlikte iptal kontrolü durum bilgisi yani sertifika iptal listesi (SİL) referansı ve/veya ÇİSDUP cevabı olabilir [14, 22]. ES-C ye imzalayan sertifikası ile birlikte tüm nitelikli sertifikalar eklenerek tüm sertifikasyon yolu oluşturulup ayrıca iptal kontrolü bilgisi de eklenirse ES-X (Genişletilmiş doğrulama verisi ile Elektronik imza- ES with extended validation data) elde edilir. Doğrulama verisi için de zaman damgası alınır.

4.1.1. CadES BES ve CaDES BES'de bulunması zorunlu nitelikler

Cades BES (Basic Electronic Signature-Temel Elektronik İmza) Şekil 4.1'de gösterilmektedir. Cades BES yapısında bulunan bileşenler:

- RFC 3369'da tanımlı imzalayan verisi (signer's document) [31]
- RFC 3369 ve ESS'de tanımlı zorunlu imzalı nitelikler (signed attributes) [31, 32]
- Bu standartta tanımlı zorunlu ek imzalı nitelikler
- Kullanıcı verisinden ve imzalı niteliklerden hesaplanan RFC 3369'da tanımlı sayısal imza değeri [31].

Ayrıca Cades BES de,

- Ek imzalı nitelikler
- İsteğe bağlı imzasız nitelikler bulunabilir.

Zorunlu imzalı nitelikler:

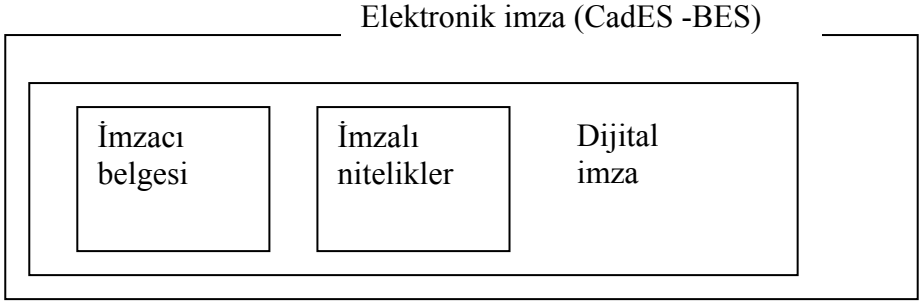
- İçerik tipi (content type). RFC 3369'da tanımlıdır ve imzalanan içerik tipini (EncapsulatedContentInfo) belirtir [31].
- Mesaj özeti (message digest). RFC 3369'da tanımlıdır ve imzalanan (encapContentInfo) içinde (eContent OCTET STRING)in özet değerini belirtir [31].
- İmzalama sertifikası (ESS signing certificate yada ESS signing certificate-V2). İmzalama sertifikası (ESS signing certificate) niteliği (Enhanced Security Services -ESS) RFC 2634'de tanımlanmıştır ve özetleme algoritması olarak yalnızca SHA-1'e izin verir [32]. İmzalama sertifikası-V2 (ESS signing certificate-V2) niteliği ise (ESS update: Adding CertID algorithm agility) RFC 5035'de tanımlanmıştır ve tüm özetleme algoritmalarına izin verir [33].

RFC 3369, ESS ve ETSI CaDES'de tanımlı isteğe bağlı imzalı nitelikler de Cades BES'e eklenebilir [4, 31, 32]. Aşağıda bu nitelikler belirtilmiştir:

- İmzalama zamanı (signing time): RFC 3369'da tanımlıdır ve imzalama zamanını gösterir [31].
- İçerik işaretleri (content hints): ESS (RFC 2634)'de tanımlıdır ve imzalı içeriğin formatı hakkında bilgi verir [32].
- İçerik referansı (content reference): ESS'de (RFC 2634) tanımlıdır. İstek ve cevap mesajları arasında ilişki kurmak için kullanılır [32].
- İçerik belirteci (content identifier): ESS'de (RFC 2634) tanımlıdır ve content reference (içerik referansı) niteliği için bir belirteç içerir [32].
- Taahhüt-tip-belirteci (commitment-type-indication): Cades BES'de tanımlıdır ve imzalayanın imzalama esnasında kabul ettiği sorumluluğu belirtmek için kullanılır [4].
- İmzalayan yeri (signer location): Cades BES'de tanımlıdır ve imzalayana imzalama yerini belirtme imkanı tanımaktadır [4].
- İmzalayan nitelikleri (signer attributes): Cades BES'de tanımlıdır ve imzalayan rolü hakkında bilgi verir [4].
- İçerik zaman damgası (content time stamp): Cades BES'de tanımlıdır ve imzalı bilgiye imzalı zaman damgasının ilişkilendirilmesini sağlar [4]. Belgenin belirtilen zamandan önce imzalandığını ispatlar.

Cades-BES, RFC 3369 ve ESS (RFC 2634)'de belirtildiği gibi ayrıca imzasız nitelikler de içerebilir [4, 31, 32]. Seri imza ise (CounterSignature) RFC 3369'da tanımlıdır ve gömülü imzalar (imza üstünde imza) gerektiğinde kullanılabilir [31].

Cades-BES genel gösterimi Şekil 4.1'de verilmiştir.



Şekil 4.1. Cades-Bes gösterimi

Cades-BES, Avrupa Elektonik İmza Direktifi yasal gereksinimlerini karşılar, temel kimlik doğrulama ve bütünlük denetimi sağlar [1, 4]. Tek başına uzun dönemli doğrulama için yeterli değildir. Uzun dönemli doğrulama için imza oluşturulduğu zamandaki sertifika iptal kontrolü bilgisinin de imza paketinde bulunması gereklidir. Cades-BES de bulunması zorunlu nitelikler aşağıda açıklanmıştır [4].

A-İçerik tipi (Content-type)

Content-type niteliği imzalı içeriğin tipini belirtir. Content-type niteliğinin sözdizimi RFC 3369'da tanımlıdır [31]. Content-type niteliği, signed-data yada authenticated-data içindeki ContentInfo'nun içerik tipini gösterir. Content-type niteliği imzalı-veride (signed-data) imzalı nitelikler varsa yada doğrulanmış-veride (authenticated-data) doğrulanmış nitelikler varsa mutlaka bulunmalıdır. Content-type niteliği imzalı-veride yada doğrulanmış verideki encapContentInfo eContentType değeriyle uyuşmalıdır. Content-type niteliği imzalı olmalıdır.

Aşağıdaki nesne belirteci (oid) content-type niteliğini belirtir:

id-contentType OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 3 }

ContentType ::= OBJECT IDENTIFIER

B-Mesaj özeti (message-digest)

Mesaj özeti (Message-digest) niteliğinin sözdizimi RFC 3369'da tanımlıdır [31]. Mesaj-özeti (message-digest) niteliği imzalı-veride (signed-data) imzalanacak encapContentInfo eContent OCTET STRING verisinin özetini gösterir [31]. İmzalı-veri için mesaj-özeti, imzalayanın mesaj özeti algoritması ile hesaplanır. İmzalı-veride imzalı niteliklerin bulunması durumunda mutlaka mesaj özeti niteliği de bulunmalıdır. Mesaj özeti mutlaka imzalı olmalıdır.

Aşağıdaki nesne belirteci ile content-type niteliği belirtilir:

id-messageDigest OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 4 }

MessageDigest ::= OCTET STRING

C-İmzalama sertifikası (SigningCertificate) niteliği

İmzalama sertifikası (SigningCertificate) niteliği, ESS imzalama-sertifikası (ESS signing-certificate) yada ESS imzalama-sertifikası-v2 (ESS signing-certificate-v2) şeklindedir [32, 33]. Bu niteliklerde imzalayan sertifikasına kaynak bulunmalıdır ve bu nitelikler yerine koyma ve yeniden yayımlama saldırılarına karşı koyan ve imza doğrulamada kısıtlı sayıda sertifikaya izin veren yapıda tasarlanmalıdır. Tam bir sertifikadan daha yoğun ve küçük bir yapıdadır ve kolayca ayırdedilebilir.

- İmzalama sertifikası (ESS signing certificate) niteliği (Enhanced Security Services -ESS) RFC 2634'de tanımlanmıştır ve özetleme algoritması olarak yalnızca SHA-1 e izin verir [32].
- İmzalama sertifikası-V2 (ESS signing certificate-V2) niteliği ise (ESS update: Adding CertID algorithm agility) RFC 5035'de tanımlanmıştır ve diğer özetleme algoritmaları kullanıldığında kullanılmalıdır [33].

İmza doğrulamada kullanılacak sertifika, sertifika zincirinde bulunmalıdır ve sertifika zinciri de boş olmamalıdır.

İmzalamaSertifikası (SigningCertificate) niteliğinin ASN.1 gösterimi ve nesne belirteci aşağıdaki gibidir:

SigningCertificate ::= SEQUENCE {
certs SEQUENCE OF ESSCertID,
policies SEQUENCE OF PolicyInformation OPTIONAL
}

id-aa-signingCertificate OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) id-aa(2) 12 }

```
ESSCertID ::= SEQUENCE {
        certHash Hash,
        issuerSerial IssuerSerial OPTIONAL
}

Hash ::= OCTET STRING -- SHA1 hash of entire certificate

IssuerSerial ::= SEQUENCE {
        issuer GeneralNames,
        serialNumber CertificateSerialNumber
}
```

Sertifika zincirindeki ilk sertifika imza doğrulama sertifikası olmalıdır. Bu sertifika için EssCertID kodlamasında issuerSerial alanı mutlaka bulunmalıdır. İmzalamaSertifikası (SigningCertificate) niteliği mutlaka imzalı olmalıdır.

4.1.2. Cades EPES (Explicit Policy Based Electronic Signature)

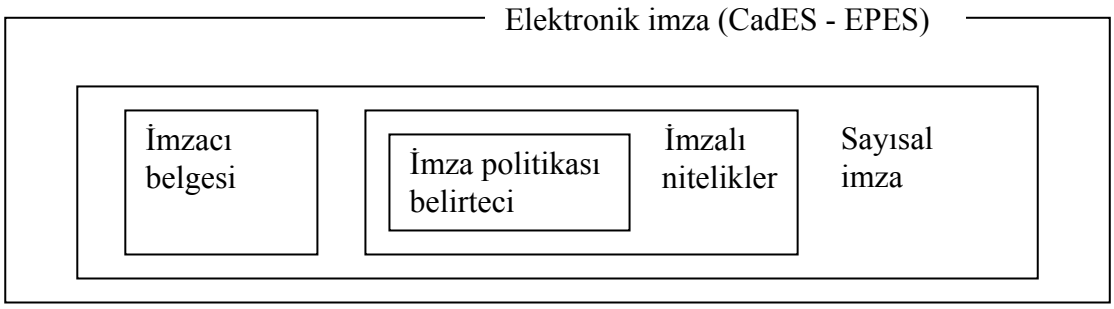
İmza politikası (Signature policy), elektronik imza oluşturma ve doğrulamada gereksinimleri karşılamak ve imzanın geçerliliğini doğrulamak için uyulması gerekli kurallar kümesidir. İmza politikası yayımlayıcı (Signature policy issuer) ise imza politikasını belirleyen ve yayımlayan varlıktır.

Cades EPES, imza politikasını gösteren imza-politikası-belirteci (signature-policy-identifier) imzalı niteliğini içerir (Şekil 4.2) ve bu nitelik elektronik imzanın doğrulanması için gereklidir [4],. Bu imzalı nitelik imza tarafından korunmaktadır. Elektronik imza, imza politikasına uyması için başka imzalı niteliklere de sahip olabilir. Cades-EPES yapısı Şekil 4.2'de verilmiştir [4]:

4.2 Doğrulama Verisi İçeren Elektronik İmza Formatları

Elektronik imzanın doğrulanabilmesi için ek bilgilere ihtiyaç vardır. Bunlara doğrulama verisi denir. Bunlar;

- Açık anahtar sertifikası (PKC- Public Key Certificate)
- Her sertifika için iptal durum bilgisi
- Sayısal imzaya uygulanan güvenilir zaman damgası
- İmza doğrulamada kullanılan imza politikası



Şekil 4.2. Cades-EPES gösterimi [4]

Doğrulama verisi imzalayan yada doğrulayan tarafından elde edilebilir. İmza politikası belirteci varsa, imza politikası gereksinimleri karşılanmalıdır. Doğrulama verisi, SM (sertifika makamı) sertifikalarıyla birlikte iptal kontrolü bilgisi içerir. İptal kontrolü bilgisi ise SİL yada ÇİSDUP sorgusu şeklinde olabilir [14, 22]. Doğrulama verisi ayrıca zaman damgası da içerebilir. Zaman damgası, imzanın verilen tarihten önce oluşturulduğunu kanıtlar.

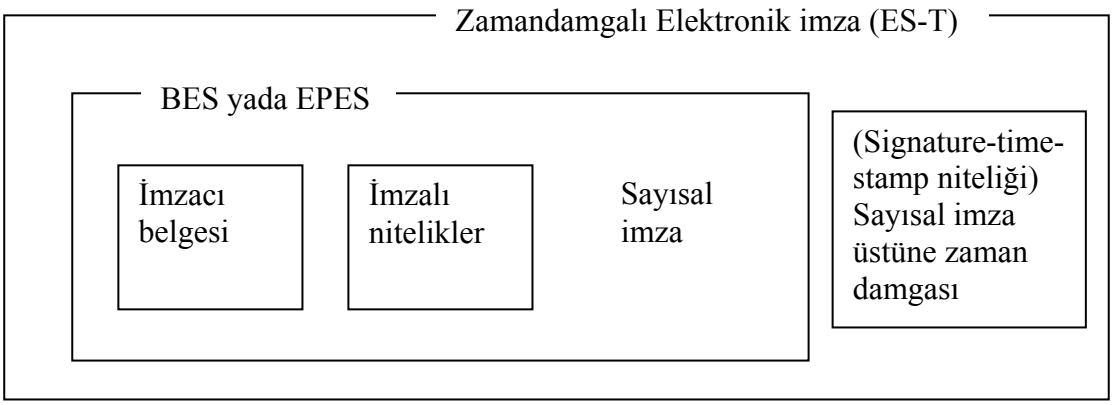
4.2.1. Zaman damgalı elektronik imza (ES-T)

Güvenilir zaman bilgisi (signature-time-stamp), imzasız nitelik (unsigned attribute) olarak elektronik imzaya eklenir. Signature-time-stamp imzasız niteliği, elektronik imzaya alınan zaman damgasını gösterir. Güvenilir zaman bilgisi, uzun dönemli doğrulamanın ilk adımıdır. Şekil 4.3 de ES-T gösterilmiştir [4].

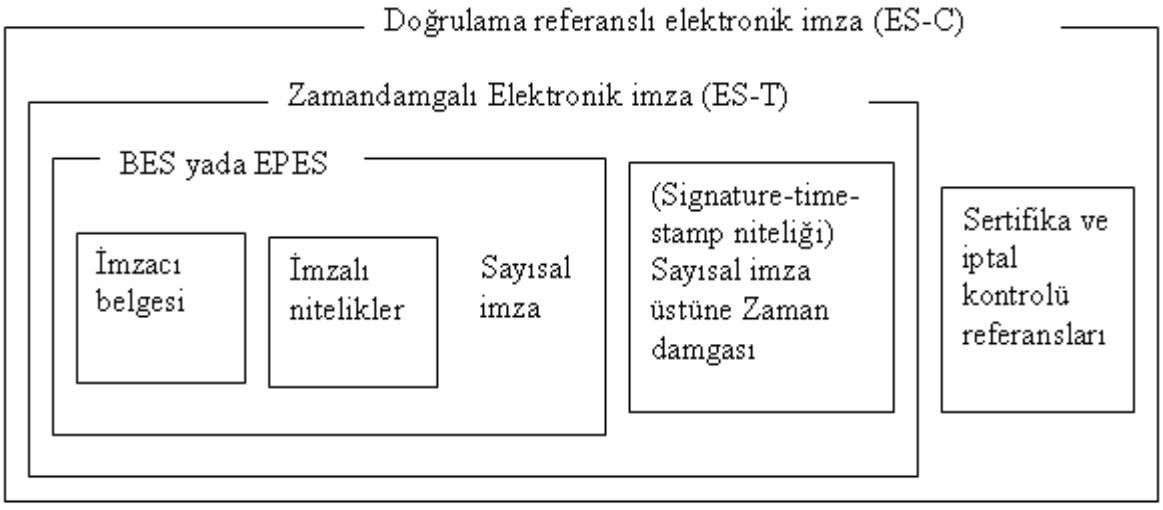
4.2.2. Doğrulama referanslı elektronik imza (ES-C)

Doğrulama referanslı elektronik imza (ES-C, Electronic Signature with Complete validation data references) formatında, ES-T imzasına complete-certificate-references ve complete-revocation-references nitelikleri eklenir [4]. Complete-certificate-references niteliği, sertifika zincirinde bulunan tüm sertifikalar için alınan referansları içerir. Complete-revocation-references niteliği ise imza doğrulamada kullanılan SİL ve/veya ÇİSDUP cevabı referanslarını içerir [14, 22]. Sadece referanslar saklandığı için imza formatının büyüklüğü çok artmaz. Bu referanslar

imzasız nitelik olarak ES-T'ye eklenir. ES-C oluşturmak için "grace period" süresi kadar beklemek gereklidir [4]. Şekil 4.4'de ES-C gösterilmiştir.



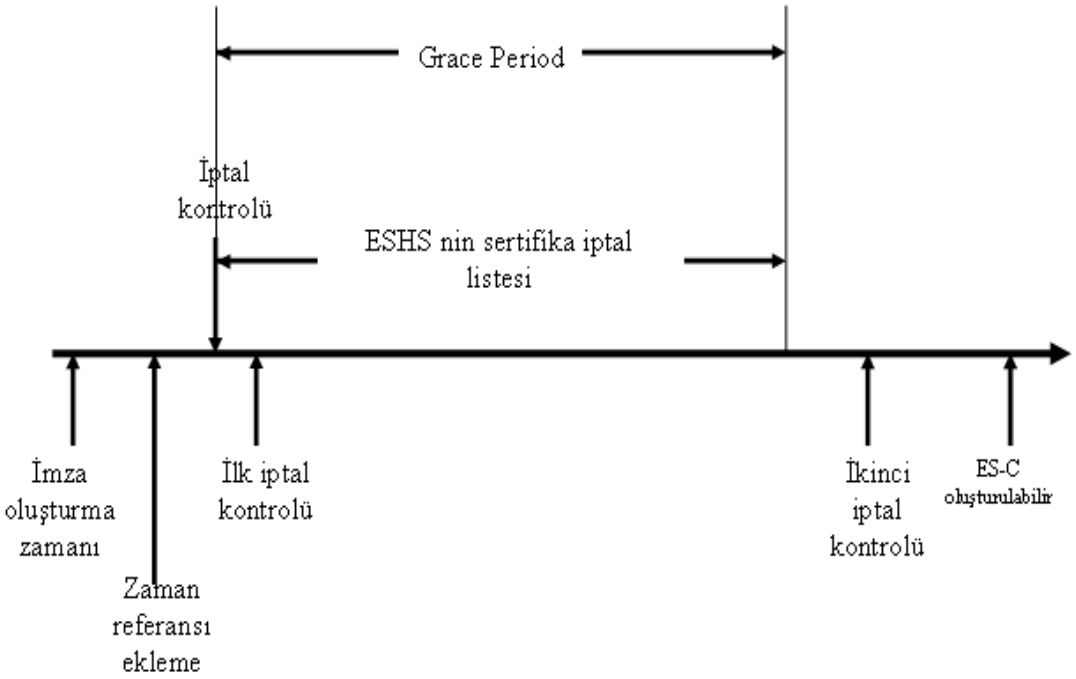
Şekil 4.3. Cades-T gösterimi [4]



Şekil 4.4. Cades-C gösterimi [4]

4.3. Grace Period

Grace period, elektronik imza oluşturulup sertifika iptal kontrolü istendiği andan itibaren ESHS'nin sertifika iptal listesini (SİL) yayımladığı zamana kadar geçen süredir [14, 23]. Grace period, Şekil 4.5 de gösterilmiştir. İmzayı ilk doğrulayan kişinin imza oluşturulduktan sonra beklemesi gerekli minimum zamandır. Bu süre beklenmeden yapılan iptal kontrolleri sağlıklı sonuç vermez. CWA 14171'de "grace period" tanımlanmıştır [23].



Şekil 4.5. Grace period [23]

5. PRATİK UYGULAMALAR

Bu bölümde ETSI 101733 standardı üstüne geliştirilmiş 2 özel uygulamaya yer verilmektedir [4]. İlk uygulama Politecnico di Torino'da Kullanılan Elektronik Doküman Yönetim Sistemi (AIDA) üstünde çalışan elektronik imza uygulamasıdır. İkinci uygulama ise görüntü (image) belgelerinin imzalanması ve imzalarının doğrulanması için geliştirilmiş olan bir applet uygulamasıdır [6].

5.1. AIDA Elektronik Doküman Yönetim Sistemi

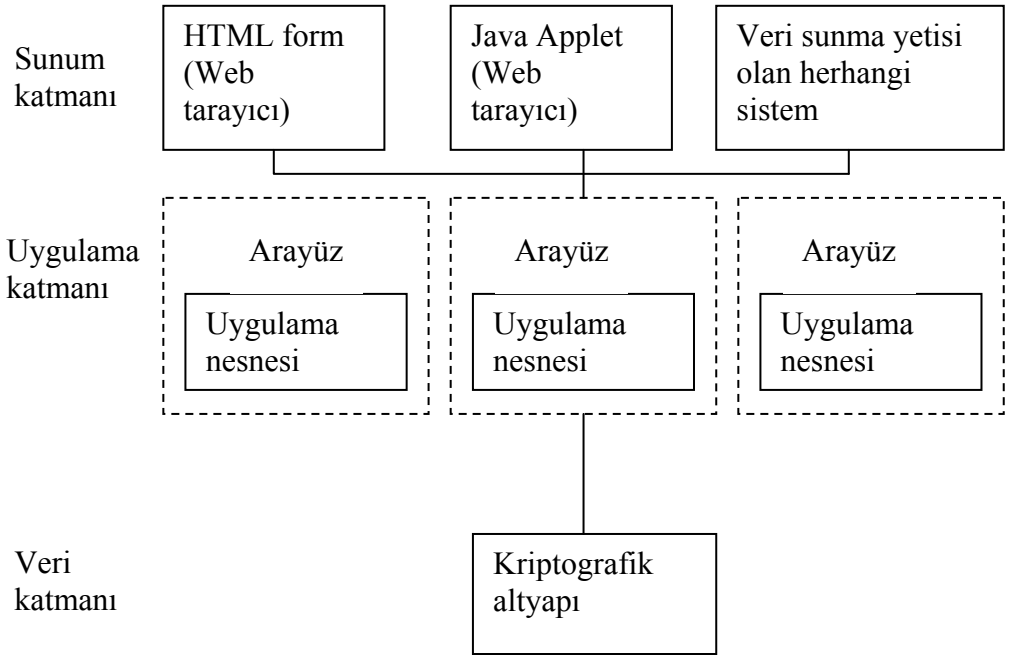
AIDA (Advanced Interactive Digital Administration) projesi, güvenilir elektronik imzaların uygulanabilirliğini göstermek amacıyla teknik bir altyapı oluşturmak için başlatılmıştır [34]. AIDA mimarisi, n-katmanlı bir web uygulaması modeline sahiptir. İlk katman sunum (presentation) katmanıdır. Web tarayıcısı ve veriyi sunulabilir forma getiren bir web sunucusu içerir. İkinci katman uygulama (application) katmanıdır, kullanıcının sunum katmanından istenen veriyi çağırabilmesine olanak verir. Üçüncü katman, uygulama katmanının ihtiyaç duyduğu kriptografik altyapıyı ve veriyi içerir. Veri kaynağı Oracle gibi bir veritabanı, XML belgeleri kümesi yada LDAP gibi bir dizin sunucusu olabilir [15]. Orta katman birden fazla uygulamaya izin verecek şekilde genişletilebilir. Şekil 5.1 de bu genel mimari gösterilmiştir.

Politecnico di Torino'da kullanılan elektronik doküman yönetim sistemi, gerçek elektronik belgelerde elektronik imza kullanımında öncü olmuştur ve statik belgeler için kullanılmıştır. Bu belgeler durum değişimlerinde yer almaz ve bu nedenle süreçlere dahil olmazlar. Elektronik imza uygulama geliştiriminde kullanılan mimari Şekil 5.2 de görülmektedir.

5.1.1. Sayısal imzanın kriptografik altyapı gerçekleştirimi

AIDA elektronik imza uygulamasının kriptografik altyapı gerçekleştiriminde EuroPKI'ın açık anahtar altyapısı servislerinden (zaman damgası sunucusu, iptal kontrolü sunucusu vs.) faydalanılmıştır [34, 35]. EuroPKI, ticari alanda yada kamu

yönetimi, araştırma enstitüleri vs. için güvenli uygulamalar sağlayan Avrupa çapında bir AAA organizasyonudur [35]. Kriptografik hizmet sağlayıcı bağlamında PKCS#11 ve MS-CryproAPI gibi farklı arayüzleri olan kütüphaneler ve kart okuyucu ve akıllı kart gibi donanım modülleri kullanılmıştır [7, 36]. Modülün geliştirilmesinde CMS [3] kütüphanesi baz alınmış ve bu kütüphane üstüne ETSI 101733 standardını gerçekleştiren bir kütüphane oluşturulmuştur [3, 4].



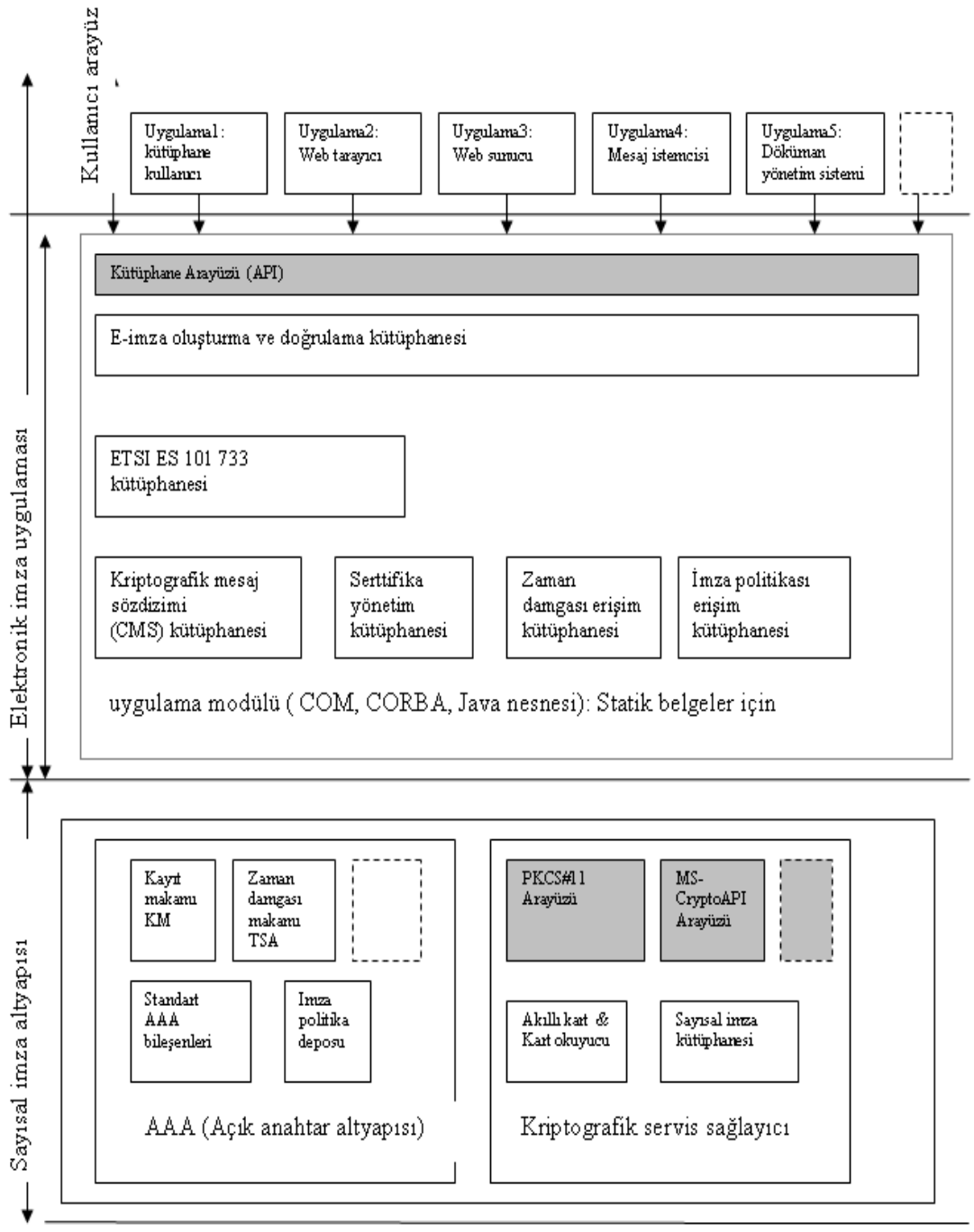
Şekil 5.1. AIDA genel mimarisi

5.1.2. Elektronik imza uygulaması gerçekleştirimi

AIDA genel mimarisinde Şekil 5.1'de belirtilen uygulama katmanı, farklı platformlarda (Linux, Win32, Solaris gibi) ve sunucu ve istemci tarafında çalışabilen, tek başına, web sunucu, mesaj sistemi yada doküman yönetim sistemi şeklinde kullanılabilecek şekilde tasarlanmıştır [34]. Bu katmandaki hedefler:

- Kütüphaneler tarafından sağlanan tüm işlevselliğin kullanıcıya sergilendiği kullanıcı arayüzleri geliştirmek.

- Avrupa standartlarında elektronik imza kullanılabilen elektronik posta sistemleri (Eudora, Netscape yada Microsoft Outlook gibi) kullanmak ve birlikte çalışabilirliği sağlamak.



Şekil 5.2. AIDA elektronik imza mimarisi

5.2. Görüntü Belgelerinin Çevrimiçi İmzalanması ve Doğrulanması

Güvenli elektronik imza kullanımına, yoğun olarak web-tabanlı kurumsal doküman yönetim sistemlerinde ihtiyaç duyulmaktadır. Elektronik imza entegre edilmesi öngörülen yazılımlar genelde merkezi bir sunucu üzerinde çalışır ve son kullanıcıya gerekli noktalarda imza atma imkanı sağlayan küçük uygulamalar applet veya activeX olarak sunulur [6, 8]. Bu uygulamaların önceden kullanıcı bilgisayarına kurulması gerekmemekte, gerek duyulduğunda kurumsal uygulamanın bir parçası olarak sunucudan yüklenmektedir.

Bu amaçla E-imza applet uygulaması, kullanıcıya web üzerinden güvenli ve ETSI standardına uygun elektronik imza atabilme imkanı sağlamak üzere geliştirilmiştir [4]. E-imza applet uygulamasında kriptografik altyapı gerçekleştirimi için Tübitak UEKAE ürünü olan İmzager API kütüphanesi kullanılmıştır [37].

5.2.1. Kullanılan teknoloji - Java Applet

Applet, Java dilinde yazılan, HTML sayfasına eklenebilen programdır [6]. Java teknoloji kullanabilen tarayıcı ile applet içeren bir sayfayı görmek istersek appletin kodu istemci bilgisayara indirilir ve tarayıcının Java Virtual Machine (JVM) ile çalıştırılır [6]. Appletler küçük boyutlu, platformdan bağımsız, ve güvenli programlar olduğu için tarayıcıdan erişilen küçük internet uygulamaları için idealdirler [6].

5.2.2. JAVA Advanced Imaging (JAI)

Sun şirketi, görüntü (image) uygulamaları geliştirmede kullanılmak üzere JAI (Java Advanced İmaging) kütüphanesini geliştirmiştir [38]. JAI kütüphanesi nesneye yönelik arayüzler sağlar [38]. Bu arayüzler, görüntü dosyalarının java uygulamalarında ve appletlerinde kolayca işlenebileceği, basit ve yüksek seviyeli bir programlama modeli sunar. JAI, tiff, jpeg, flashpix, png, bmp, gif, pnm gibi belge tiplerine destek verir [38]. E-imza applet uygulamasında görüntü dosyalarının işlenmesinde JAI kütüphanesi kullanılmıştır.

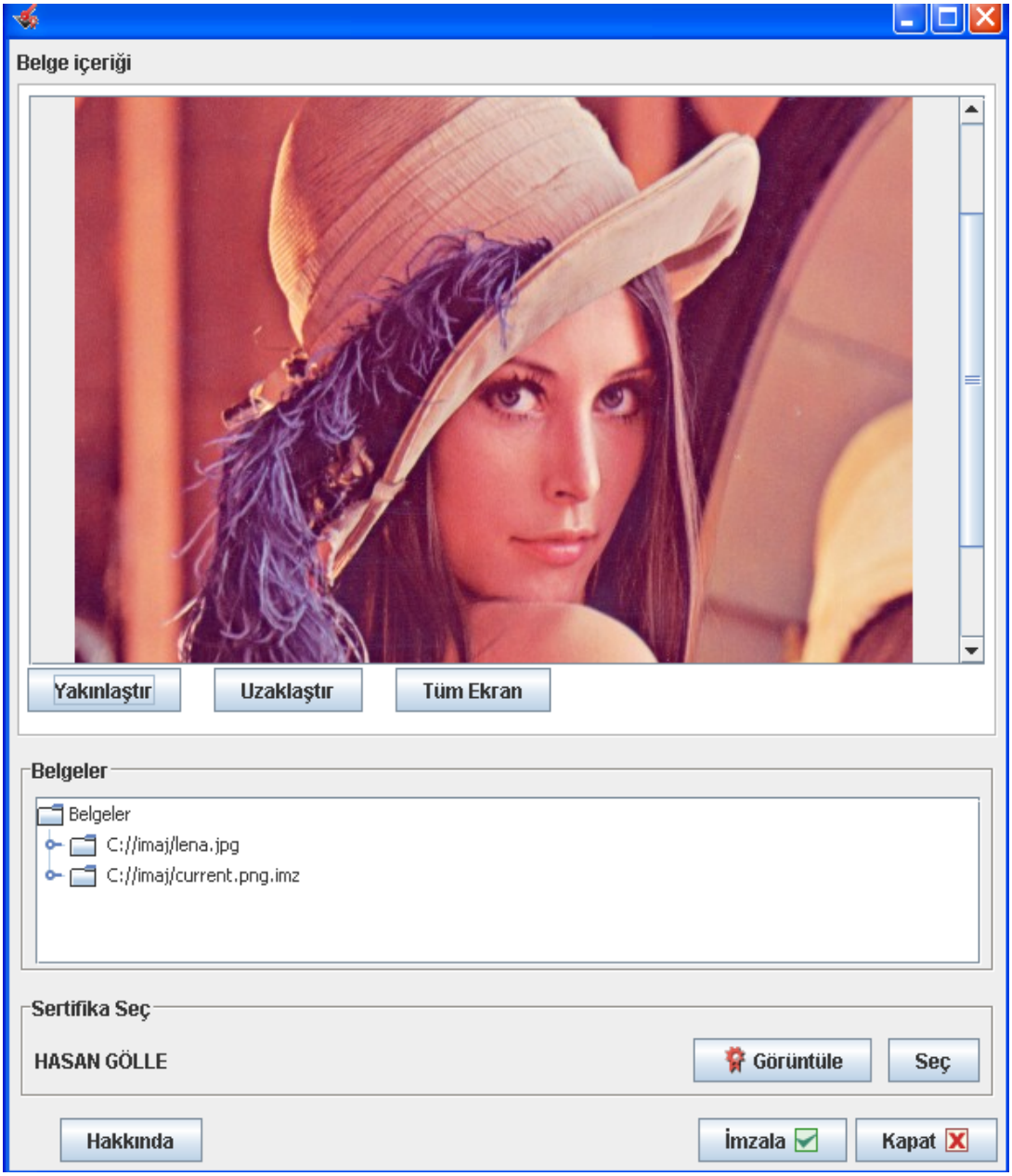
5.2.3. Kullanıcı arayüzü

Kullanıcı arayüzü gerçekleştiriminde CWA 14171 standart olarak kullanılmıştır [23]. Şekil 5.3'de E-imza applet kullanıcı arayüzü görülmektedir. Kullanıcı bilgisayarına takılı kart okuyucu ve akıllı kartlar görüntülenir. Kullanıcıya ait e-imza sertifikaları görüntülenir. Elektronik imza atmak isteyen kullanıcının birden fazla sertifikaya sahip olması durumunda, kendisine imzalama işleminde kullanacağı sertifikayı seçme imkanı sunulur. Sertifika detay bilgileri istendiğinde kullanıcı tarafından görüntülenebilir.

Kullanıcının imzalalayacağı belgeler, ekleri ve bu belgeler üzerindeki elektronik imzalar ile beraber görüntülenebilir. Elektronik imza appleti "rtf", "gif", "tif", "bmp", "png", "jpg" gibi görüntü ve text dosyalarını ve bunların imzalanmış "imz" uzantılı hallerini görüntüleyip imzalayabilir. E-imza applet, imzalanacak belgeyi ekleri ile beraber görüntüleme özelliğine sahiptir. Kullanıcı istediği belgeyi ağaç üstünden seçerek görüntüleyebilir. Sözkonusu belgeler imzalı ise imza doğrulaması yapılıp kullanıcıya belge üzerindeki elektronik imza gösterilir. Kullanıcı PIN numarasını klavyeden girer veya arayüzdeki tuşları tıklayarak seçebilir.

5.2.4. Fonksiyonlar

Elektronik imza oluşturulacak belgeler diskten okunur, belge dizisi oluşturulur, ağaç yapısında varsa belge üstündeki imzalar da görüntülenebilir. Eğer belge imzalı ise imza doğrulama yapılır. Kullanıcı, imza doğrulama sonucunu görebilir. Kullanıcı, fare kullanarak ağaç üstündeki istediği belgeyi görüntüleyebilir. Belgenin çok sayfalı bir "tif" belgesi olması durumunda kullanıcıya diğer sayfaları da görüntüleme imkanı sağlanır. Kullanıcıya istediği belgeyi ayrı bir pencerede görüntüleme imkanı da verilir. Kullanıcıya bilgisayarına bağlı akıllı kartları ve kart okuyucuları görüntüleme imkanı verilir. Elektronik imza sertifikası kullanıcı tarafından seçilebilir.



Şekil 5.3. E-imza applet kullanıcı arayüzü

Elektronik imza oluşturulacak belge, kullanıcı tarafından imzalanmak istendiğinde akıllı kart PIN kodunun girileceği bir ekran görüntülenir. Kullanıcı, akıllı kart şifresini girer, e-imza appleti şifre bilgilerini akıllı karta güvenli bir yoldan iletir ancak şifrenin doğru olması durumunda e-imza işlemini sonuçlandırarak kullanıcıyı bilgilendirir. Yanlış PIN girilirse kullanıcıya uyarı mesajı görüntülenir. Elektronik

imzalı belgeler, istenirse kayıt dizinine ve sıkıştırılarak sunucuya yüklenebilir (upload).

5.2.5. Programa giren parametreler

E-İmza applet, web tarayıcısından aşağıdaki parametreleri alarak çalışır:

Parametrelerden herhangi birisinin verilmemesi veya yanlış verilmesi durumunda, Applet hata vererek çıkacaktır.

- Dosya Adı

<param name= "fileName" value = " C://imaj/test.tif , test2.tif , http://localhost:8080/jspbook/ornek.tif, ornek2.tif ">

İmzalanacak belge ve eklerin isimleri, mutlak (absolute) adresleri ile beraber virgül ile ayrılarak string yapısında E-imza appletine girdi olarak verilir. Applet, "fileName" parametresinde belirtilen belgeleri diskten okuyarak bir ağaç yapısı içinde kullanıcıya gösterir. "test.tif" dosya adı ile diskten okunan belge elektronik imzalandıktan sonra "test2.tif.imz" dosya adı ile kayıt dizinine kaydedilir. "ornek.tif" dosya adı ile imzalanacak belge sunucudan da alınabilir, imzalandıktan sonra "ornek2.tif" dosya adı ile kayıt dizinine kaydedilir. Virgül ile ayrılan, ilki mutlak dosya adı ve ikincisi imzalandıktan sonra belgeye verilecek dosya adını ifade eden belge adları çiftler halinde E-imza applet'e girdi olarak verilebilir. İmzalanacak belgelerden e-imzalı olanlara imza doğrulaması yapıldıktan sonra yeni bir seri imza eklenir.

- Yükleme (Upload) Adresi

<param name = "uploadAdres" value =http://www.sunucuadresi">

<param name = "uploadParams" value = " param1, param2, param3 ">

E-imzalı belgelerin zip formatında sıkıştırıldıktan sonra kaydedileceği sunucu (ve ilgili dizinin) adresidir. Kullanıcı seçtiği e-imza sertifikasını kullanarak, istenilen belgeleri ekleri ile beraber imzalayıp sunucuda ilgili adrese yükler (upload). (İmzalı

belgelerin yükleneceği sunucu adresi, E-imza applet'ine Doküman Yönetim Sistemi tarafından sağlanmalıdır.)

- Zip Dosya Adı

<param name = "zipDosyaAdi" value = "C://ornek.zip">

Sunucuya yüklenecek belgeler zip formatında sıkıştırıldıktan sonra özel bir isim verilir.

- Lokal Kayıt Dizini

<param name = "kayitDizini" value = "C://kayit">

Sunucuya yüklenecek belgeler zip formatında sıkıştırıldıktan sonra istenirse kullanıcının bilgisayarında özel bir dizine de kaydedilebilir.

5.2.6. Sertifika bilgileri

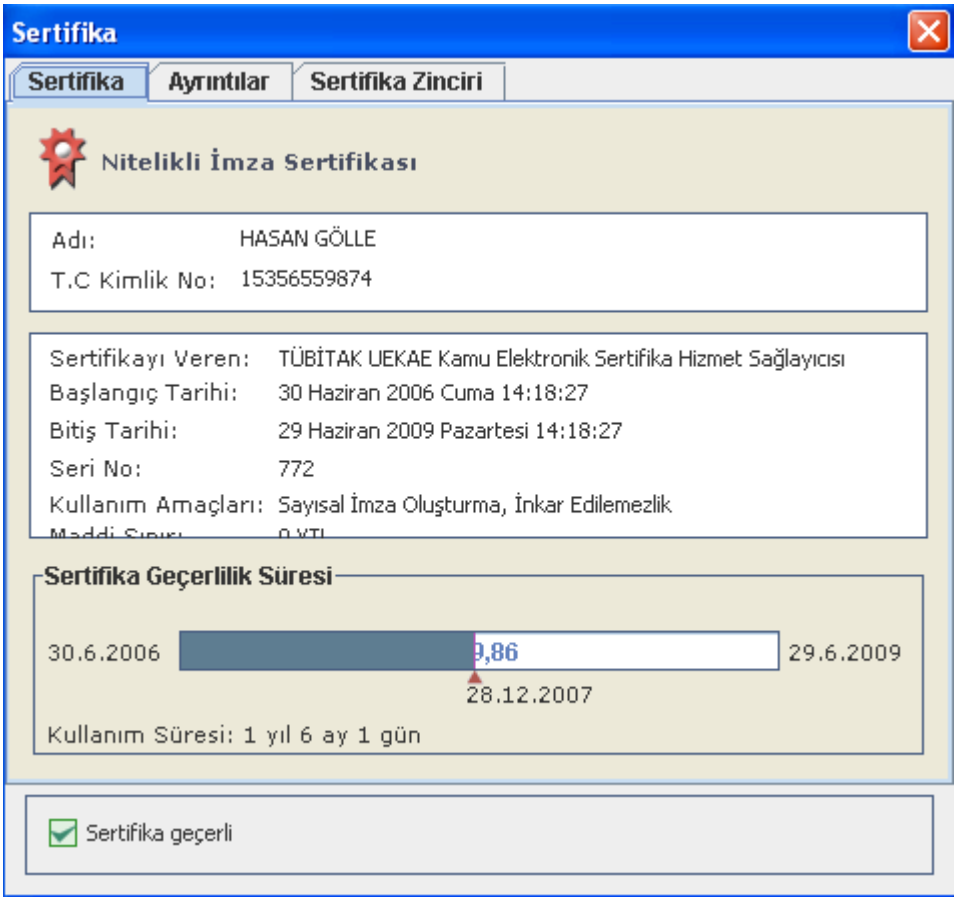
Kullanıcı sertifika bilgilerinin görüntülendiği ekran Şekil 5.4'de görülmektedir. Sertifika ayrıntıları ve sertifika zinciri bilgileri bu ekran yardımı ile görüntülenebilir.

5.2.7. Sertifika seçimi

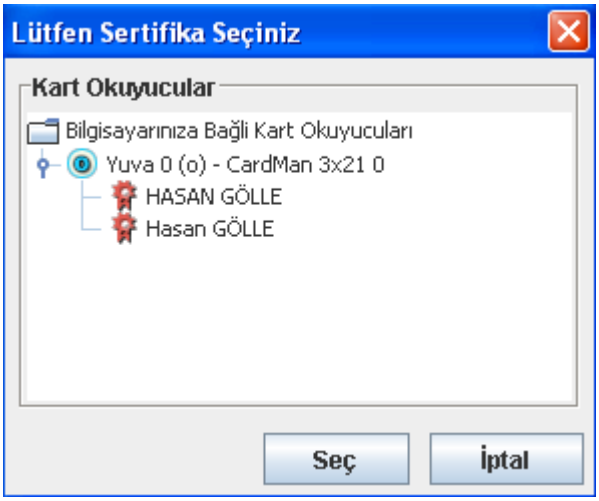
Kullanıcının elektronik imza sırasında kullanabileceği sertifikalar Şekil 5.5 deki gibi sertifika seçme ekranında listelenir ve kullanıcı'nın istediği sertifikayı seçebilmesi sağlanır.

5.2.8. PIN giriş ekranı

Kullanıcı, ana pencere'de seçtiği belge ve sertifikalar ile elektronik imza işlemini başlatabilir. Bu işlem kullanıcıya akıllı kartına ait PIN kodunun girilmesi için PIN Giriş ekranı'nın Şekil 5.6'deki gibi görüntülenmesi ile başlar. Kullanıcı PIN kodunu girdikten sonra "Tamam" butonuna basarak elektronik imza işlemine devam edebilir ya da "İptal" butonu yardımı ile vazgeçebilir.



Şekil 5.4. Kullanıcı sertifikası



Şekil 5.5. Sertifika seçme ekranı

Şekil 5.6. PIN girme ekranı

Kullanıcının doğru PIN girerek "Tamam" butonuna basması ile seçilen belge seçilen sertifika ile imzalanır ve e-imzalı belgeler belirtilen sunucu adresine yüklenir (upload). Kullanıcı, imzalama ve yükleme işleminin gerçekleştiğine dair bilgilendirilir. İmzalama işleminin başarısız olması durumunda kullanıcıya uyarı mesajları görüntülenir.

5.2.9. Applet jar

Java appletin sistem kaynaklarına erişebilmesi için yani belge yada dizin oluşturup diske yazabilmesi için imzalı olması gerekir [6]. Javanın sağladığı *jarsigner* aracı ile applet jarı imzalanmıştır. Ayrıca applet ile aynı dizinde bulunan JAI imageIO jarı da aynı araçla imzalanmıştır [38]. Sonuçta elde edilen imzalı dosyalar Tübitak tarafından üretilmiş olan İmzager API kullanılarak açıldığında imzaların doğrulandığı ve ETSI standardına uygun olduğu görülmüştür [4, 37]. Ayrıca oluşturulan imzalı dosyaların sıkıştırılıp (zip) parametre olarak verilen sunucu adresine başarıyla yüklendiği (upload) görülmüştür.

6. ETSI UYUMLU E-İMZA KÜTÜPHANESİ OLUŞTURMA (BESImza.dll)

5070 sayılı elektronik imza kanunu ile elektronik imzanın hukuki yapısı, elektronik sertifika hizmet sağlayıcılarının faaliyetleri ve her alanda elektronik imzanın kullanımına ilişkin işlemler kapsanmaktadır [39]. Bu kanuna göre oluşturulan elektronik imzalar, ıslak imza ile aynı geçerliliğe sahiptir. Kanun, elektronik imza hakkında teknik detaylara girmemektedir. Farklı programlar ve kişiler tarafından oluşturulan elektronik imzaların karşılıklı olarak doğrulanabilmesi ve bu esnada problem yaşanmaması amacıyla, Bilgi Teknolojileri ve İletişim Kurumu, oluşturulan elektronik imzaların ETSI 101733 standartına uygun olmasını tavsiye etmektedir.

Microsoft CryptoAPI ile CMS standardı kullanılarak elektronik imza oluşturulmaktadır [3, 7]. Türkiyede Tübitak'ın geliştirmiş olduğu imzager (java ve .net) e-imza kütüphaneleri ETSI 101733 uyumlu elektronik imza üretip doğrulamaktadırlar [4, 37]. ETSI TS 101733'de belirtilen BES (temel elektronik imza-basic electronic signature) formatının CMS formatından farklı olarak sahip olduğu ek imza nitelikleri nedeniyle MS Crypto API'nin ürettiği elektronik imzalar, Tübitak imzager elektronik imza kütüphanesi tarafından doğrulanamamaktadır [3, 4, 7, 37].

ETSI 101733, Cryptographic Message Syntax (RFC 3369) üzerine yazılmış bir standartdır [4, 31]. CMS'nin güncel RFC'si RFC 3852'dir[3]. ETSI ise RFC 3369'u kullanmıştır [31]. CMS standartında, atılan imzalara "Attribute" olarak adlandırılan çeşitli nitelikler eklemek mümkündür. ETSI 101733, niteliklerle (attribute) ilgili çeşitli kısıtlar koyarak 9 farklı imza formatı tanımlamaktadır [4]. Bunlardan en basiti BES (Basic Elektronic Signature) yani Temel Elektronik İmzadır. BES imzası içinde bulunması zorunlu olan 3 nitelik (attribute) vardır. Bunlar RFC 3369'de tanımlı Content-type, Message-digest nitelikleri ve RFC 2634'te tanımlı signing-certificate veya other-signing-certificate niteliğidir [31, 32].

MS CryptoAPI ile ETSI uyumlu elektronik imza atamamadaki problem de bu noktada ortaya çıkmaktadır. MS CryptoAPI, BES yapısı içine konması gereken 3

nitelikten (attribute) ilk ikisini imza içine koymasına rağmen, üçüncü olarak konması gereken signing-certificate veya other-signing-certificate niteliklerinin ikisini de koymamaktadır [7]. Bu nedenle MS CryptoAPI içinde bu nitelikleri direkt olarak tanımlamak mümkün değildir [7]. Fakat Microsoft, imza yapısına bir niteliği (attribute) eklemek için gerekli alt yapıyı oluşturmuştur. Kodlaması (encoding) dahil olmak üzere nitelik oluşturup MS CryptoAPI'ye verilirse istenen niteliğin eklenmesi mümkündür [7]. Bu şekilde geliştirilmiş olan BESImza.dll kütüphane modülü ile ETSI uyumlu elektronik imza oluşturup imza doğrulama yapmak mümkündür [4].

6.1. BesImza.dll Kütüphanesinin Sağladığı Temel Metodlar

BesImza.dll elektronik imza kütüphane modülü, Microsoft .Net C++ ortamında, MS CryptoAPI kullanılarak hazırlanmıştır. MS CryptoApi'nin temel kriptografik metodlarını kullanabilmek amacıyla projede "wincrypt.h" dosyası dahil edilmiştir (include).

BesImza.dll elektronik imza modülü kullanılarak ETSI 101733 uyumlu elektronik imza oluşturulabilir ve oluşturulan imzaların iptal kontrolleri yapılabilir [4]. Masaüstü ve web tabanlı kullanıcı arayüzü yazılımlarında ve uygulama yazılımlarında bu kütüphanenin dışarıdan kullanılması (import edilmesi) yeterlidir. Uygulama geliştiren kişinin, karmaşık kriptografik algoritmaları bilmesine ve kullanmasına gerek kalmadan güvenli elektronik imza uygulaması geliştirilebilir. Bu da kütüphanenin sağladığı kullanımı kolay 5 adet temel metodla mümkün olmaktadır. Metodların kullanım şekli, işlevi, gönderilen ve dönen parametreler ve kodlaması aşağıda detaylı olarak verilmektedir.

6.1.1 İmzaAt metodu

Metodun kullanımı aşağıdaki gibidir.

*extern "C" __declspec (dllexport) BYTE * imzaAt (int len, BYTE * imzalanacak , int * size);*

Bu metod, parametre olarak verilen *imzalanacak* adındaki byte dizisini elektronik imzalar ve byte dizisi şeklinde imzalı veri döndürür. Bu imzalı veri, ETSI 101733 CadES uyumludur [4]. *Len* tamsayısı *imzalanacak* byte dizisinin büyüklüğüdür. *Size* tamsayısı ise imzalama işleminden sonra dönen imzalı byte dizisinin büyüklüğüdür.

Bölüm 4.1.1 de Cades BES'de bulunması zorunlu olan SigningCertificate niteliği anlatılmış ve ASN.1 gösterimi verilmişti [4]. SigningCertificate niteliğinin oluşturulup elektronik imzaya eklenmesi gerekir. Aşağıda detayları verilen metodla SigningCertificate niteliği oluşturulmuştur. SigningCertificate niteliği oluşturulurken seçimli olan PolicyInformation kısmı eklenmemiştir. Fakat ESSCertID içindeki seçimli IssuerSerial kısmı imzaya dahil edilmiştir. Böylece imzayı atan kişinin sertifikasına ulaşmak mümkün olmuştur.

Öncelikle DER encode fonksiyonları tanımlanmış, bu temel encode fonksiyonları ile nesne belirteci 4.1.1 de verilen SigningCertificate niteliği oluşturulmuş, bu niteliğin imzaya eklenmesi anlatılmıştır. Son olarak imzaAt metodunun örnek bir kullanımı verilmiştir.

ASN.1 encode

Bu yapıyı encode edebilmek için DER kodlama (encode) fonksiyonları hazırlanır. Bu fonksiyonlar, MS CryptoAPI'de tanımlı bir veri yapısı olan CRYPT_ATTR_BLOB üzerinde işlem yapmaktadır [7]. Bu amaçla hazırlanmış olan EncodeLen, MakeTLV ve BasaTagEkle metodları EK-1 de verilmiştir. TLV, Tag, Length, Value kelimelerinin baş harflerinden oluşmaktadır. Der kodlaması böyle bir yapı kullanmaktadır.

Niteliğin oluşturulması

Elimizde kodlama için gerekli fonksiyonlar olduğuna göre, bu fonksiyonları kullanarak ASN.1 yapısı tanımlı SigningCertificate yapısı kodlanabilir. Bu esnada MS CryptoAPI veri yapısı olan PCCERT_CONTEXT tipindeki sertifikanın elimizde olduğu varsayılmaktadır [7].

```
CRYPT_ATTRIBUTE signingCertAttributeOlustur(PCCERT_CONTEXT pSignerCert)
{

	//Sertifika icinden blob olarak issuer'i ve serial'i alalim.
	CERT_NAME_BLOB issuerBlob = pSignerCert->pCertInfo->Issuer;
	CRYPT_INTEGER_BLOB serialBlob = pSignerCert->pCertInfo->SerialNumber;

	//blob halindeki issuerdan GeneralNames yapisini olusturalim.
	CRYPT_ATTR_BLOB issuerGeneralNames = issuerBlob;
	basaTagEkle(&issuerGeneralNames,0xa4,false);
	basaTagEkle(&issuerGeneralNames,0x30);


	//blob halindeki serial'dan CertificateSerialNumber=INTEGER yapisini olusturalim
	CRYPT_ATTR_BLOB serial = serialBlob;
	basaTagEkle(&serial,0x02,false);

	//IssuerSerial yapisini olusturalim
	CRYPT_ATTR_BLOB issuerSerial = makeTLV(issuerGeneralNames,serial);

	//Hash'i hesaplayalim ve Hash yapisini olusturalim.
	CRYPT_ATTR_BLOB hash = calculateDigest(pSignerCert);
	basaTagEkle(&hash,0x04);

	//Hash ve IssuerSerial'i kullanarak ESSCertID yapisini olusturalim.
	CRYPT_ATTR_BLOB eSSCertID = makeTLV(hash,issuerSerial);
	basaTagEkle(&eSSCertID,0x30);

	//SigningCertificate yapisini elde edelim.
	PCRYPT_ATTR_BLOB signingCert = new CRYPT_ATTR_BLOB;
	*signingCert = eSSCertID;
	basaTagEkle(signingCert,0x30);

	//Boylece attribute olarak ekleyecegimiz encoded yapiyi olusturmus olduk.

	//Ve bu encoded yapidan attribute'u olusturalim.
	CRYPT_ATTRIBUTE attr;
	attr.pszObjId = "1.2.840.113549.1.9.16.2.12"; //id-aa-signingCertificate RFC 2634 icinde
	//tanimli
	attr.cValue = 1;
	attr.rgValue = signingCert;

	return attr;
}
```

İmzanın MS CryptoApi ile oluşturulması

Nitelik oluşturulduktan sonra imza yapısına bu niteliğin eklendiği imzala() metodu aşağıdaki gibi olabilir:

```
int __stdcall imzala(PCCERT_CONTEXT pSignerCert,BYTE *Data,DWORD
DataLen,PCRYPT_ATTR_BLOB sonuc)
{

	CRYPT_SIGN_MESSAGE_PARA  SigParams;
	DWORD DataSize;
	const BYTE* MessageArray[] = {Data};
	DWORD MessageSizeArray[1] = {DataLen};

	if(sonuc == NULL)
		return CRYPT_SIGN_MESSAGE_ERROR;

	//Once parametreleri sifirlayalim.
	memset(&SigParams,0,sizeof(CRYPT_SIGN_MESSAGE_PARA));

	//Simdi parametreleri set edelim.
	//Structure'in uzunlugu
	SigParams.cbSize = sizeof(CRYPT_SIGN_MESSAGE_PARA);
	//Encoding'i belirleyelim
	SigParams.dwMsgEncodingType = MY_ENCODING;
	//Imzalamada kullanilacak sertifika
	SigParams.pSigningCert = pSignerCert;
	//Imzada kullanilacak hash (ozet) algoritmasi
	SigParams.HashAlgorithm.pszObjId = szOID_OIWSEC_sha1;
	SigParams.HashAlgorithm.Parameters.cbData = 0;
	//Imza yapisi icine konacak sertifikalari ekleyelim.
	//Biz sadece imzayi atanin sertifikasini koyuyoruz.
	SigParams.cMsgCert = 1;
	SigParams.rgpMsgCert = &pSignerCert;


	//Mikrosoft'un eklemedigi attribute'u olusturalim.
	CRYPT_ATTRIBUTE attr = signingCertAttributeOlustur(pSignerCert);
	//Olusturdugumuz attibute'u parametrelere ekleyelim.
	SigParams.cAuthAttr= 1;
	SigParams.rgAuthAttr = &attr;


	//DataSize icine imza sonucunun kac BYTE oldugunu alalim.
```

```
	if(!CryptSignMessage(&SigParams,false,1,MessageArray,MessageSizeArray,NULL,&D
ataSize)) {
		delete[] attr.rgValue->pbData;
		delete attr.rgValue;
		return CRYPT_SIGN_MESSAGE_ERROR;
	}

	//Sadece ne kadar yere ihtiyac oldugu ogrenilmek istenmisse imzayi atmadan donelim.
	if(sonuc->cbData <= 0)
	{
		sonuc->cbData = DataSize;
		delete[] attr.rgValue->pbData;
		delete attr.rgValue;
		return SUCCESS;
	}

	//Eger bana verilen yer yeterli degilse hata donelim.
	if(sonuc->cbData < DataSize)
	{
		sonuc->cbData = DataSize;
		delete[] attr.rgValue->pbData;
		delete attr.rgValue;
		return CRYPT_SIGN_SIZE;
	}


	//Imzayi atalim
	if(!CryptSignMessage(&SigParams,false,1,MessageArray,MessageSizeArray,sonuc-
>pbData,&DataSize)) {
		delete[] attr.rgValue->pbData;
		delete attr.rgValue;
		return CRYPT_SIGN_MESSAGE_ERROR;
	}
	sonuc->cbData = DataSize;

	//BITTI
	delete[] attr.rgValue->pbData;
	delete attr.rgValue;
	return SUCCESS;

}
```

imzaAt fonksiyonunun hafıza kullanımına dikkat çekmek gerekirse, MS CryptoAPI yaklaşımında olduğu gibi bu fonksiyonun çalışması, gerçekten imzalama işlemini yapmak ile sadece ihtiyaç duyulan hafıza miktarını dönmek arasında duruma göre değişmektedir [7]. Eğer parametre olarak verilen sonuc blobunun hafıza büyüklüğü

(sonuc->cbData) 0 ise sadece ihtiyaç duyulan hafıza büyüklüğünü sonuc->cbData içinde döndürür. Eğer hafıza büyüklüğü yeterli ise imza atma işlemi yapılır.

Örnek imzaAt () kullanımı

Yukarda verilen metodları kullanarak aşağıdaki gibi elektronik imza oluşturulabilir.

```
PCCERT_CONTEXT cert;
BYTE imzalanacak[] = "deneme";
int imzalanacakLen = 6;

CRYPT_ATTR_BLOB imza;
imza.cbData = 0;
sonuc = imzala(cert,imzalanacak,imzalanacakLen,&imza);
if(sonuc != SUCCESS)
        cout << "HATA" << endl;
imza.pbData = new BYTE[imza.cbData];
sonuc = imzala(cert,imzalanacak,imzalanacakLen,&imza);
if(sonuc != SUCCESS)
        cout << "HATA" << endl;
//imzayı kullan
delete[] imza.pbData;
CertFreeCertificateContext(cert);
```

6.1.2. İptalKontrolu metodu

Metodun kullanımı aşağıdaki şekildedir.

*extern "C" __declspec (dllexport) int iptalKontrolu (int len, BYTE * dogrulanacak);*

Bu metoda iptal kontrolü yapılacak olan *dogrulanacak* adındaki byte dizisi (imzalı mesaj) parametre olarak verilir. İptal kontrolü sonucunda sonuç geçerliyse 1 geçersizse 0 döner. *Len* tamsayısı, iptal kontrolü yapılacak byte dizisinin büyüklüğüdür.

Metod, öncelikle parametre olarak verilen imzalı veriyi çözümler ve imzacı sertifikasını alır (Bölüm 6.1.2.1). Sertifikadan SİL dağıtım noktasını alır (Bölüm 6.1.2.2). Daha sonra sertifika iptal listesini (SİL) internetten indirir (Bölüm 6.1.2.3). İndirdiği sertifika iptal listesini imzalayan kök sertifikanın parmak izini alır. Sertifikayı üreten ESHS nin internette ilan edilen parmakizi ile karşılaştırır. Örneğin;
"\x30\x30\xac\xab\xb5\x4b\x2d\x31\xff\xa7\x6\x0\xea\x74\xc0\xf1\xa6\x70\xbf\x91",
"http://www.kamusm.gov.tr/Bilgideposu/NESIL.v2.crl"
Böylece sahte sertifika iptal listesi kullanmanın önüne geçilir. (Bölüm 6.1.2.4) Son olarak sertifika iptal kontrolü yapılır. Sonuç geçerliyse 1 döner, geçersizse 0 döner.

İmzacı sertifikasını almak

SİL dağıtım noktalarının imzacı sertifikasından alınması gereklidir [14]. Bu nedenle İptalKontrolü metodu aşağıda verilen getSignerCert altmetodunu çağırır. Bu metod imzalı mesajı parametre alır ve imzacı sertifikasını döner. GetSignerCert metodu EK-2 de verilmiştir. Aşağıda sözde kod (pseudocode) gösterimi verilmektedir.

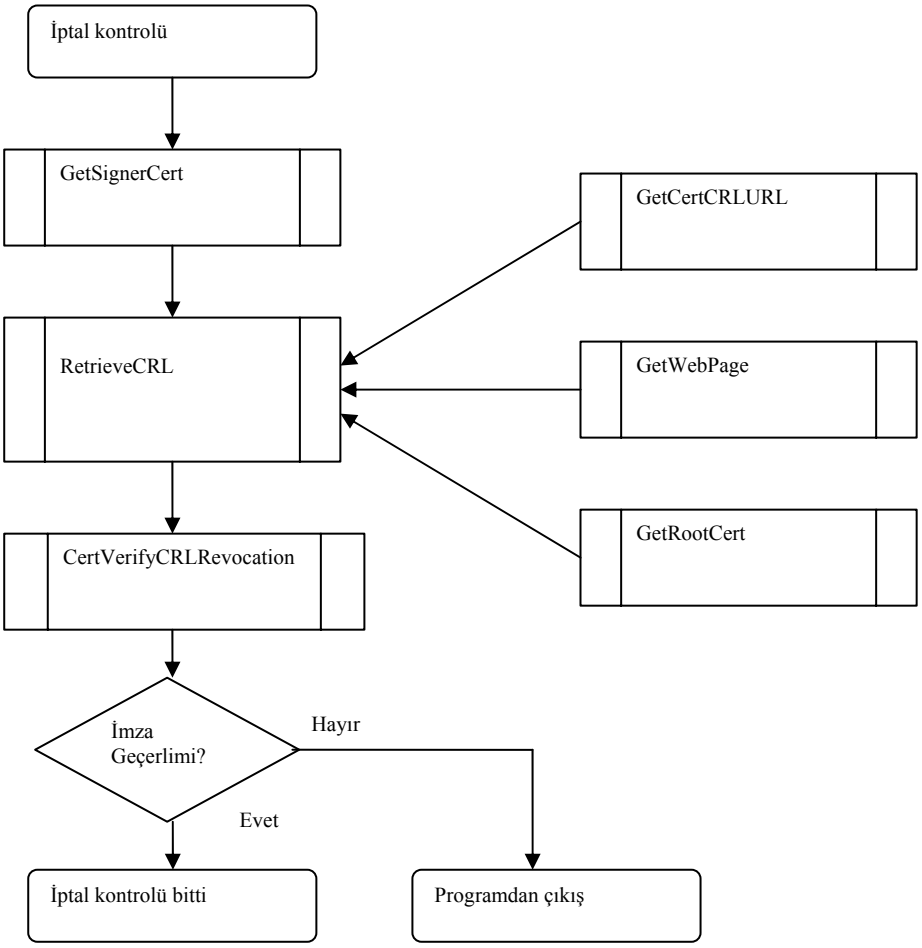
```
FUNCTION  Pccert_Context getSignerCert
Sertifika sahibinin ismini tutmak için buffer tanımla.
//char pszNameString[MAX_NAME];
Decode için gerekli değişkenlere ilk değer ata.
CryptMsgOpenToDecode ile mesajı decode et ve mesajdaki imzayı doğrula.
CryptMsgUpdate kullanarak mesajı encoded blob ile güncelle.
CryptMsgGetParam ile decode edilen mesajı tutmak için gerekli byte sayısını bul.
Malloc ile bellek ayır.
CryptMsgGetParam ile içeriği kopyala.
Mesajdan imzacı CERT_INFO bilgisini al.  {
  Sertifika için gerekli bellek büyüklüğünü al.
      Malloc ile bellek ayır.
  // Sertifika bilgisi CERT_INFO) yapısını alındı.   }
CERT_STORE_PROV_MSG kullanarak sertifika deposunu aç.
CertGetNameString ile sertifika deposunda imzacı sertifikasını bul.
```

```
Temizlik
//free(pSignerCertInfo)
pSignerCertContext dön;
END FUNCTION
```

Sertifikadan SİL dağıtım noktasını almak

Yukarda elde edilen imzacı sertifikası içinden öncelikle SİL dağıtım noktasının url bilgisi alınmalıdır. Bulunan SİL dağıtım noktası url'inden SİL indirilir. Bu işlem için kullanılan RetrieveCRL, GetCertCRLURL, GetWebPage, GetRootCert altmetodları EK-2 de verilmiştir. Şekil 6.1 de iptal kontrolü işlemi genel metod ve altmetod çağırımları verilmektedir. Bu altmetodların sözde kod (pseudocode) gösterimleri aşağıdaki gibi olabilir.

```
FUNCTION Pccrl_Context RetrieveCRL
CertOpenSystemStore ile sertifika deposunu aç.
GetCertCRLURL ile sertifikadan sil dağıtım noktasının url'ini al.
GetWebPage ile SİL indir.
Sertifikadan kök sertifikayı bul.
Kök sertifikanın parmakizi ile ilan edilen parmak izini karşılaştır.
cert->issuer değerini tabloda kontrol et.
IF tabloda geçerli bir parmak izi
        CRL yapısını oluştur
END IF
pCRL dön;
END FUNCTION
```



Şekil 6.1. İmzacı sertifikasından SİL dağıtım noktasını almak

GetCertCRLURL metodu ile sertifika iptal listesi dağıtım noktası alınır ve parmakizi kontrolü yapılır [14]. EK-2 de verilmiştir. Aşağıda sözde kod gösterimi verilmektedir.

```
FUNCTION Lpstr GetCertCRLURL
IF bLookupOnly==FALSE
   sil dağıtım noktasını sertifikadan al ve dön.
ELSE
   sertifika yayımlayıcı kök sertifikanın parmakizini al
   internette ilan edilen parmakizleri ile karşılaştır
END IF
END FUNCTION
```

BesImza.dll projesinde bulunan crlVerify.h belgesindeki rgLookupTable tablosunda (Elektronik Sertifika Hizmet Sağlayıcı) ESHS'lerin internet sayfalarında ilan edilen kök sertifikalarının parmakizleri bulunmaktadır. Yukarda bulunan parmakizi bu tablodaki parmakizleri içinde aranır. Eğer bulunamazsa programdan çıkılır.

```
static CACERT rgLookupTable[] = {
  { (BYTE
*)"\x30\x30\xac\xab\xb5\x4b\x2d\x31\xff\xa7\x6\x0\xea\x74\xc0\xf1\xa6\x70\xbf\x91",
    "http://www.kamusm.gov.tr/Bilgideposu/NESIL.v2.crl" },
  { (BYTE
*)"\x1b\x4b\x39\x61\x26\x27\x6b\x64\x91\xa2\x68\x6d\xd7\x02\x43\x21\x2d\x1f\x1d\x96",
    "http://www.kamusm.gov.tr/Bilgideposu/NESIL.v3.crl" },
  { (BYTE
*)"\x66\xe6\x28\x6a\x61\xcb\x79\xa8\x24\xb3\xfc\xca\x49\x02\xbf\x24\x6a\x39\xd8\xd1",
    "http://www.kamusm.gov.tr/tr/Bilgideposu/Bilgideposu/NESIL.v2.crl" },
  { (BYTE *)"\x74\x7b\x82\x03\x43\xf0\x00\x9e\x6b\xb3\xec\x47\xbf\x85\xa5\x93",
    "http://crl.verisign.com/RSASecureServer.crl" },
  { (BYTE *)"\xc5\x70\xc4\xa2\xed\x53\x78\x0c\xc8\x10\x53\x81\x64\xcb\xd0\x1d",
 "https://www.thawte.com/cgi/lifecycle/ getcrl.crl?skeyid=%07%15%28mps%AA"
     "%B2%8A%7C%0F%86%CE8%93%008%05%8A%B1" },
  {(BYTE *)"\x5f\x94\x4a\x73\x22\xb8\xf7\xd1\x31\xec\x59\x39\xf7\x8e\xfe\x6e",
 "https://www.trustcenter.de:443/cgi-bin/CRL.cgi/ TC_Class3.crl?Page=GetCrl"
     "&crl=4" },
  { (BYTE *)"\x71\x1f\x0e\x21\xe7\xaa\xea\x32\x3a\x66\x23\xd3\xab\x50\xd6\x69",
    "http://crl.verisign.com/Class2SoftwarePublishers.crl" },
  { 0, 0 }
};
```

Sertifika iptal listesini internetten indirmek

GetWebPage metodu ile SİL, bölüm 6.1.2.2'de bulunan url'den indirilir [14]. GetWebPage metodu EK-2 de verilmiştir. Sözde kod gösterimi aşağıdaki gibi olabilir.

```
FUNCTION Byte*  GetWebPage
HINTERNET  m_Session;
InternetOpenUrl ile Url aç ve HTTP dosyası için işleyici al.
IF hHttpFile geçerli ise
        HttpQueryInfo ile HTTP dosyasının büyüklüğünü öğren.
        HTTP dosyası içeriği için bellek ayır.
        InternetReadFile ile HTTP dosyasını oku.
        InternetCloseHandle ile bağlantıyı kapat.
END IF
İçerik dön.
END FUNCTION
```

Kullanıcı sertifikasının kök sertifikasını bulmak ve kontrol etmek

GetRootCert metodu ile yukarda getSignerCert metodu ile bulunan kullanıcı sertifikasının kök sertifikası bulunur. Daha sonra bu sertifikanın parmak izi alınıp internette ilan edilen parmakizi ile karşılaştırılabilir. GetRootCert metodu EK-2 de verilmiştir. Bu metodun sözde kod gösterimi aşağıdaki gibi olabilir.

```
FUNCTION Pccert_Context GetRootCert
Veri yapılarına ilk değer ata.
CertCreateCertificateChainEngine ile sertifika zinciri motoru (Certificate Chain Engine)
oluştur.
CertOpenSystemStore(NULL,"MY") ile "My" sistem sertifika deposunu aç.
Alınan sertifika ve CertGetCertificateChain ile sertifika zincirini oluştur.
Kök sertifikayı bulalım.
Return Issuer;
}
```

6.1.3 SertifikaGoster metodu

Metodun kullanımı aşağıdaki şekildedir.
*extern "C" __declspec (dllexport) void sertifikaGoster (int len, BYTE * dogrulanacak);*

SertifikaGoster metodu, parametre olarak verilen *dogrulanacak* adındaki byte dizisi şeklindeki imzalı veriyi çözümler ve bölüm 6.1.2.1 de anlatılan getSignerCert metodu ile bulunan kullanıcı sertifikası kullanıcıya gösterilir. *Len* tamsayısı, parametre olarak verilen byte dizisinin büyüklüğüdür.

MSDN'de bulunan örnekten kodlama esnasında faydalanılmıştır [40]. Bu amaçla aşağıdaki metod geliştirilmiştir. "cryptui.lib" referansının kullanılması gereklidir.

```
void crlVerify::display (PCCERT_CONTEXT cert){
CryptUIDlgViewContext ile sertifikayı kullanıcıya göster.
if(cert)
{
  CertFreeCertificateContext(cert); //sertifikayı bırak.
}
}
```

6.1.4 İcerikAl metodu

Metodun kullanımı aşağıda gösterilmektedir.
*extern "C" __declspec (dllexport) BYTE * icerikAl (int len, BYTE * dogrulanacak, int * size);*

İçerikAl metodu, parametre olarak verilen *dogrulanacak* adındaki byte dizisi şeklindeki imzalı veriden mesajın imzalanmadan önceki orijinal halini elde etmek için kullanılır. Bu amaçla EK-3'de verilen DecodeMessage metodu, seçenek parametresi '1' ile çağrılarak kullanılır. *Len* tamsayısı, parametre olarak verilen byte dizisinin büyüklüğüdür.

6.1.5. İmzaciCertAl metodu

Metodun kullanımı aşağıdaki şekildedir.
*extern "C" __declspec (dllexport) BYTE * imzaciCertAl (int len, BYTE * dogrulanacak, int * size);*

İmzaciCertAl metodu, parametre olarak verilen *dogrulanacak* adındaki byte dizisi şeklindeki imzalı veriden imzacı sertifikası sahibini elde etmek için kullanılır. Bu amaçla EK-3'de verilen DecodeMessage metodu, seçenek parametresi '2' ile çağrılarak kullanılır. *Len* tamsayısı, parametre olarak verilen byte dizisinin büyüklüğüdür.

6.2 Hata Mesajları

Elektronik imza oluşturma, imza doğrulama ve iptal kontrolü işlemleri esnasında alınabilecek hatalar ve olası nedenleri aşağıda açıklanmaktadır.

6.2.1. İmza oluşturma hata mesajları

ASN.1 encode metodları (EncodeLen, MakeTLV ve BasaTagEkle), signingCertAttributeOlustur ve imzala metodlarının çalışması esnasında alınabilecek hatalar aşağıda verilmektedir.

1 "len çok uzun"
İmzalanacak veri boyutu çok büyük
2 "Sertifika deposu açılamadı"
Sertifika deposu açılamadı.
3 "Sertifika seç Uİ başarısız"
Sertifika seçme ekranı başarısız oldu.
4 "imzalamada HATA"
İmzalama yapılamadı

6.2.2. İmza doğrulama hata mesajları

GetSignerCert ve DecodeMessage metodlarının çalışması esnasında alınabilecek hatalar aşağıda verilmektedir.

1 "OpenToDecode başarısız"
İmza doğrulamada decode için mesaj açılması başarısız.

2 "Decode MsgUpdate başarısız"
Encode edilen blob mesaja eklenemedi.

3 "Decode CMSG_CONTENT_PARAM başarısız."
Buffer'ın decode edilen mesajı tutabilmesi için gerekli byte sayısını bulunamadı.

4 "Decode bellek ayırma başarısız."
Bulunan byte sayısı oranında bellek ayırma başarısız.

5 "Decode CMSG_CONTENT_PARAM #2 başarısız"
İmzalanan içerik ayrıştırılamadı.

6 "Doğrulama SIGNER_CERT_INFO #1 başarısız."
Mesajdan imzacı CERT_INFO alınamadı.

7 " Doğrulama bellek ayırımı başarısız."
Bellek ayırma işlemi başarısız.

8 "Doğrulama SIGNER_CERT_INFO #2 başarısız"
Mesaj sertifika bilgisi (CERT_INFO) yapısını alınamadı

9 "Doğrulama depo açma başarısız"
Mesajdaki sertifikalarla başlatılan bellek sertifika depo açılamadı.

10 "İmzacı bulunamadı.\n"
Mesajdan imzacı alınamadı.

11 "Verify GetSubjectCert başarısız"
Genel olarak doğrulamada hata var.

12 "İmza doğrulanamadı. \n"
İmzacı sertifikasından alınan CERT_INFO ile imza doğrulanamadı.

6.2.3. İptal kontrolü hata mesajları

RetrieveCRL, GetCertCRLURL, GetWebPage ve GetRootCert metodlarının çalışması esnasında alınabilecek hatalar aşağıda verilmektedir.

1 "CryptUİDlgViewContext call failed."
Sertifika gösterme başarısız.

2 "Memory allocation failed."
Veri yapıları başlatılırken bellek ayırmada hata var.

3 "The engine creation function failed."
Sertifika zincir motoru başarısız.

4 "The MY system depo did not open."
MY sistem depo açılamadı.

5 "The chain could not be created."
Sertifiika zinciri oluşturulamadı.

6 "Depo açılamadı.\n"
Sertifika depo açılamadı.

7 "crl urlleri birbirini tutmuyor!"
SİL dağıtım sunucu urlleri uyuşmazlığı var.

8 "OpenToDecode failed"
Decode için mesaj açılamadı.

9 "Decode MsgUpdate başarısız"
Encode edilen blob mesaja eklenemedi

10 "Decode CMSG_CONTENT_PARAM başarısız."
Buffer'ın decode edilen mesajı tutabilmesi için gerekli byte sayısını bulunamadı.

11 "Decode bellek ayırma başarısız."
Bulunan byte sayısı oranında bellek ayırma başarısız.

12 "Decode CMSG_CONTENT_PARAM #2 başarısız"
İmzalanan içerik ayrıştırılamadı.

13 "Doğrulama SIGNER_CERT_INFO #1 başarısız."
Mesajdan imzacı CERT_INFO alınamadı.

14 " Doğrulama bellek ayırımı başarısız."
Bellek ayırma işlemi başarısız.
15 "Doğrulama SIGNER_CERT_INFO #2 başarısız"
Mesaj sertifika bilgisi (CERT_INFO) yapısını alınamadı.
16 "Doğrulama depo açma başarısız"
mesajdaki sertifikalarla başlatılan bellek sertifika depo açılamadı.
17 "İmzacı bulunamadı.\n"
Mesajdan imzacı alınamadı.
18 "Verify GetSubjectCert başarısız"
Genel olarak doğrulamada hata var.

6.3. Yazılım Tasarımı

Elektronik imza oluşturma, imzalı mesajın çözümlenmesi (decode) ve imza doğrulama, sertifika iptal kontrolü için kullanılan işlem adımları ve algoritmalar aşağıda verilmektedir. Ayrıca bu işlemler için gerekli MS Crypto API metodları ve bu metodların kullanımı da aşağıda verilmektedir [7].

6.3.1. Elektronik imza oluşturmada kullanılan algoritma

MS CryptoApi'nin sağladığı en temel kriptografik medodlar kullanılarak Şekil 6.2 deki işlem adımları ile ETSI 101733 uyumlu elektronik imza gerçekleştirilebilir [4, 7].

Değişkenleri tanıtıp ilk değer atadıkdan sonra kullanıcıya sertifika seçtirilir.

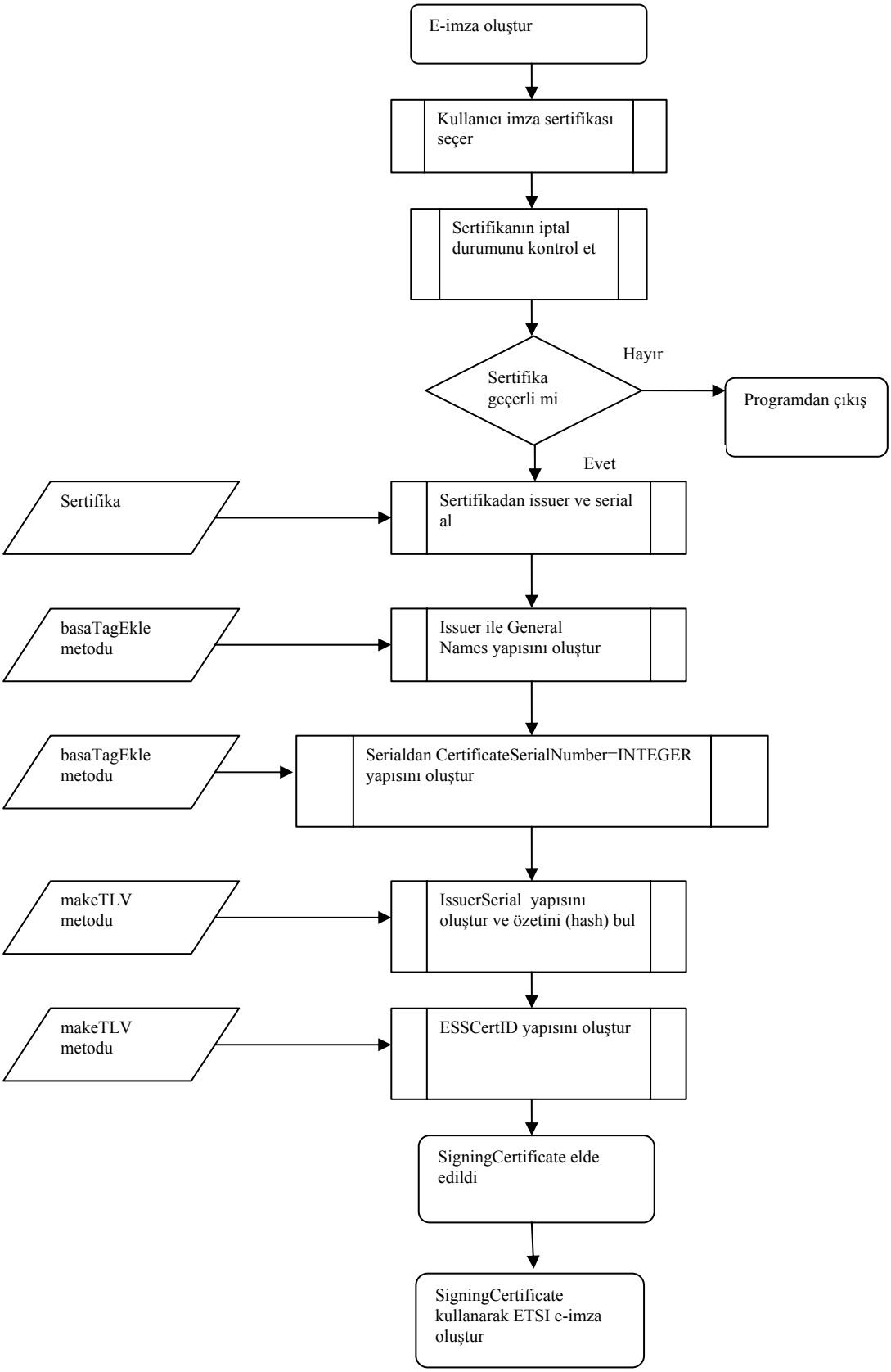
- Sertifika depo açmak için CertOpenSystemStore kullanılır.
- Depodaki sertifikaları liste halinde gösterip kullanıcının seçmesine imkan tanımak için CryptUIDlgSelectCertificateFromStore kullanılır.

Seçilen sertifika için iptal kontrolü yapılır. Sertifika geçerliyse devam edilir.
Microsoft'un eklemediği signingCertificate niteliği oluşturulur. Bu nitelik, RFC 2634'de tanımlıdır [32].

- Sertifika içinden blob olarak issuer'i ve serial'i alınır.
- Blob halindeki issuerdan GeneralNames yapısı oluşturulur.
- Blob halindeki serial'dan CertificateSerialNumber=İNTEGER yapısı oluşturulur.
- IssuerSerial yapısı oluşturulur.
- Hash'i hesaplamak ve Hash yapısını oluşturmak için CryptAcquireContext, CryptCreateHash, CryptHashData, CryptGetHashParam, CryptDestroyHash, CryptReleaseContext metodları kullanılır.
- Hash ve IssuerSerial kullanarak ESSCertID yapısı oluşturulur.
- SigningCertificate yapısı elde edilir.

İmza sonucunun kaç byte olduğunu belirlemek için CryptSignMessage kullanılır.
CryptSignMessage(&SigParams,false,1,MessageArray, MessageSizeArray,NULL, &DataSize)

İmzayı atmak için CryptSignMessage kullanılır.
CryptSignMessage(&SigParams,false,1,MessageArray,MessageSizeArray,sonuc->pbData,&DataSize))



Şekil 6.2. ETSI imzada kullanılacak SigningCertificate niteliğinin oluşturulması

6.3.2. Decode ve imza doğrulamada kullanılan algoritma

Mesajı çözümlemek (decode) ve imza doğrulamak için MSDN'de bulunan örnekten faydalanılarak, Şekil 6.3'de gösterilen işlem adımları kullanılmıştır [41]:

1 Decode etmek amacıyla mesajı açmak için CryptMsgOpenToDecode.
2 Decode edilecek mesaja encode edilmiş BLOB eklemek için CryptMsgUpdate.
3 Mesajı decode etmek için CryptMsgGetParam
4 Alınan ve decode edilen mesajı kullanarak bellekteki sertifika depoyu açmak için CertOpenStore.
5 İmzacı sertifikasını almak için CertGetSubjectCertificateFromStore.
6 Mesajdaki imzayı doğrulamak için CryptMsgControl.

6.3.3. Sertifika iptal kontrolünde kullanılan algoritma

Sertifika iptal kontrolü işlem adımları Şekil 6.4'de gösterilmiştir [42].

İmzalı mesajdan sertifika çekilir:

1 Decode etmek amacıyla mesajı açmak için CryptMsgOpenToDecode.
2 Decode edilecek mesaja encode edilmiş BLOB eklemek için CryptMsgUpdate.
3 Decode edilmiş mesaj içeriğini tutmak için gerekli olan byte sayısını bulmak için CryptMsgGetParam.
4 Mesaj içeriğini buffer'a kopyalamak için CryptMsgGetParam.
5 Cert_info bilgisi için gerekli byte sayısını bulmak için CryptMsgGetParam.
6 Cert_info bilgisini almak için CryptMsgGetParam.
7 Alınan ve decode edilen mesajı kullanarak bellekteki sertifika deposunu açmak için CertOpenStore.
8 İmzacı sertifikasını almak için CertGetSubjectCertificateFromStore.
9 Temizlik .

Sertifika kullanarak SİL [14] dağıtım noktaları (CRL distribution points) bulunur :

1 Sertifika deposunu açmak için CertOpenSystemStore.
2 SİL [14] dağıtım noktası extension bulmak için CertFindExtension.
3 SİL [14] dağıtım noktası extension decode etmek için CryptDecodeObject.

SİL [14] internetten indirilir :

1 Windows İnternet Kütüphanesi (WinInet) metodlarının kullanımını başlatmak için InternetOpen.
2 Url açmak ve SİL belgesine işleyici almak için InternetOpenUrl.
3 SİL belgesinin büyüklüğünü öğrenmek için HttpQueryInfo.
4 SİL belgesini okumak için InternetReadFile
5 Bağlantıyı kapatmak için InternetCloseHandle.

Sertifikanın kök (root) sertifikası bulunur:

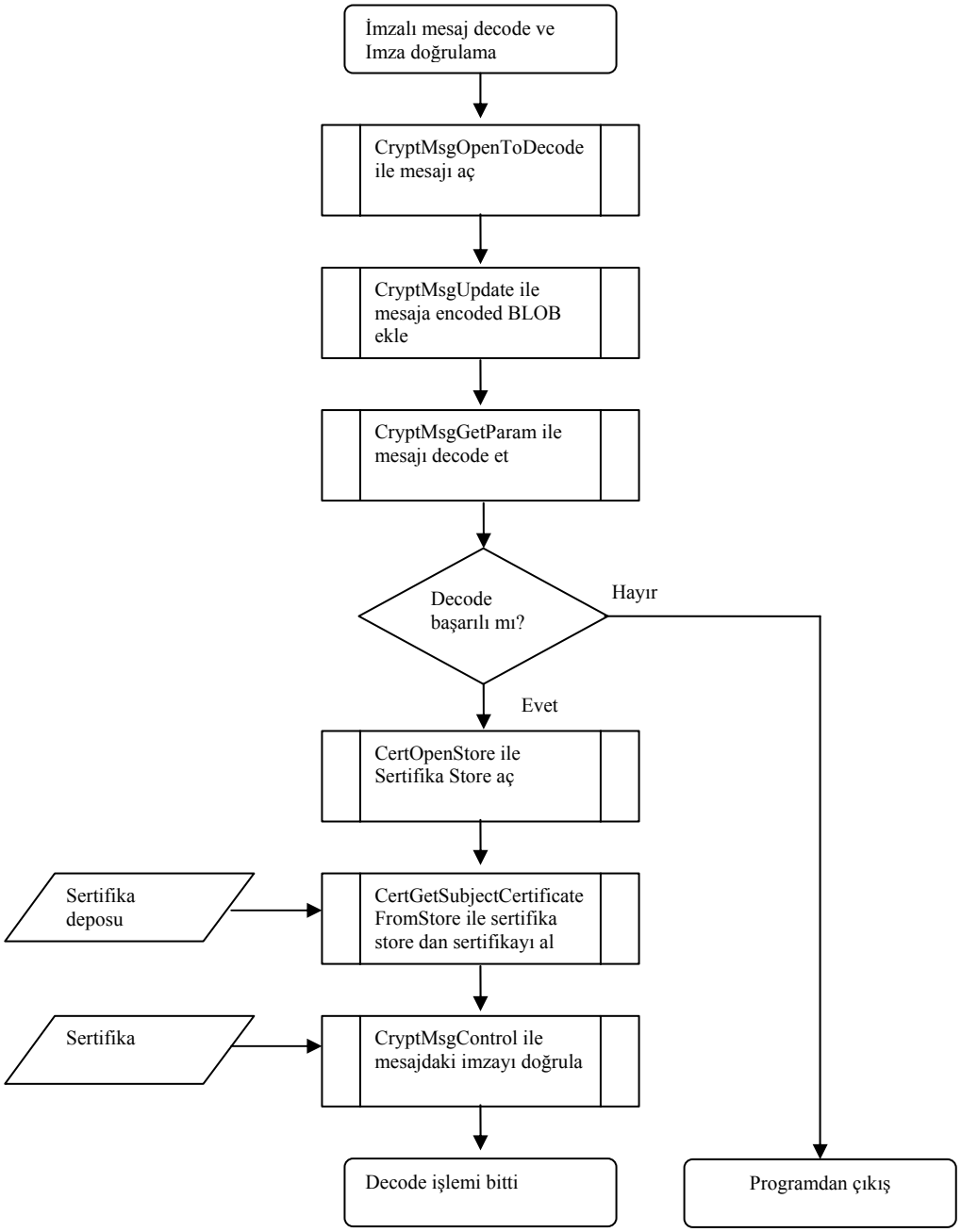
1 Sertifika zincirini oluşturmak için CertCreateCertificateChainEngine.
2 "My" sistem sertifika deposu açmak için CertOpenSystemStore.
3 Zinciri oluşturup kök sertifikayı almak için CertGetCertificateChain.

Kök sertifikanın parmakizi ile ilan edilen ve tabloda bulunan parmak izi karşılaştırılır :

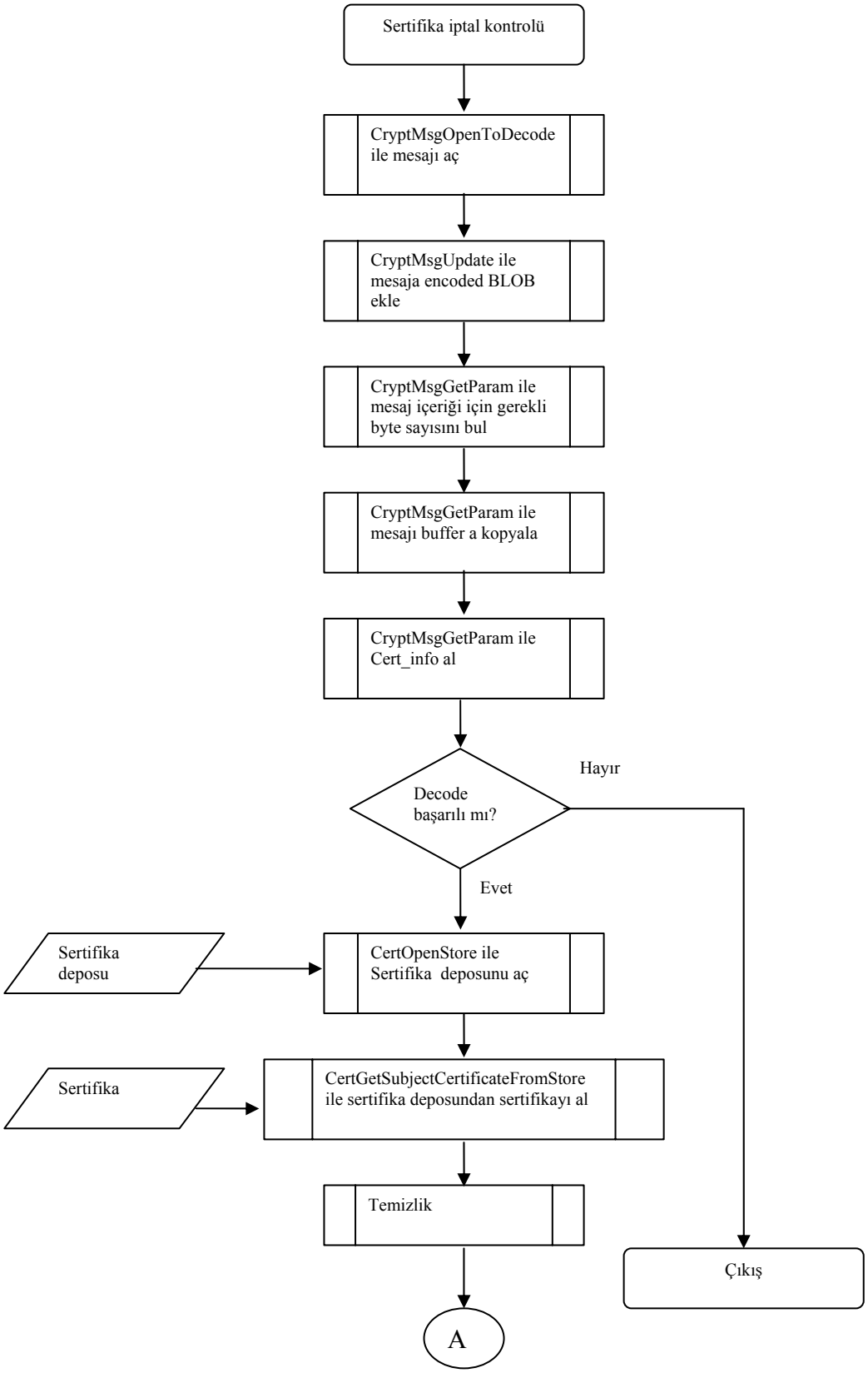
1 Kök sertifikanın parmakizini almak için CryptHashCertificate.
2 İlan edilen parmakizi ile karşılaştırmak için memcmp.

Sertifika iptal kontrolü yapılır:

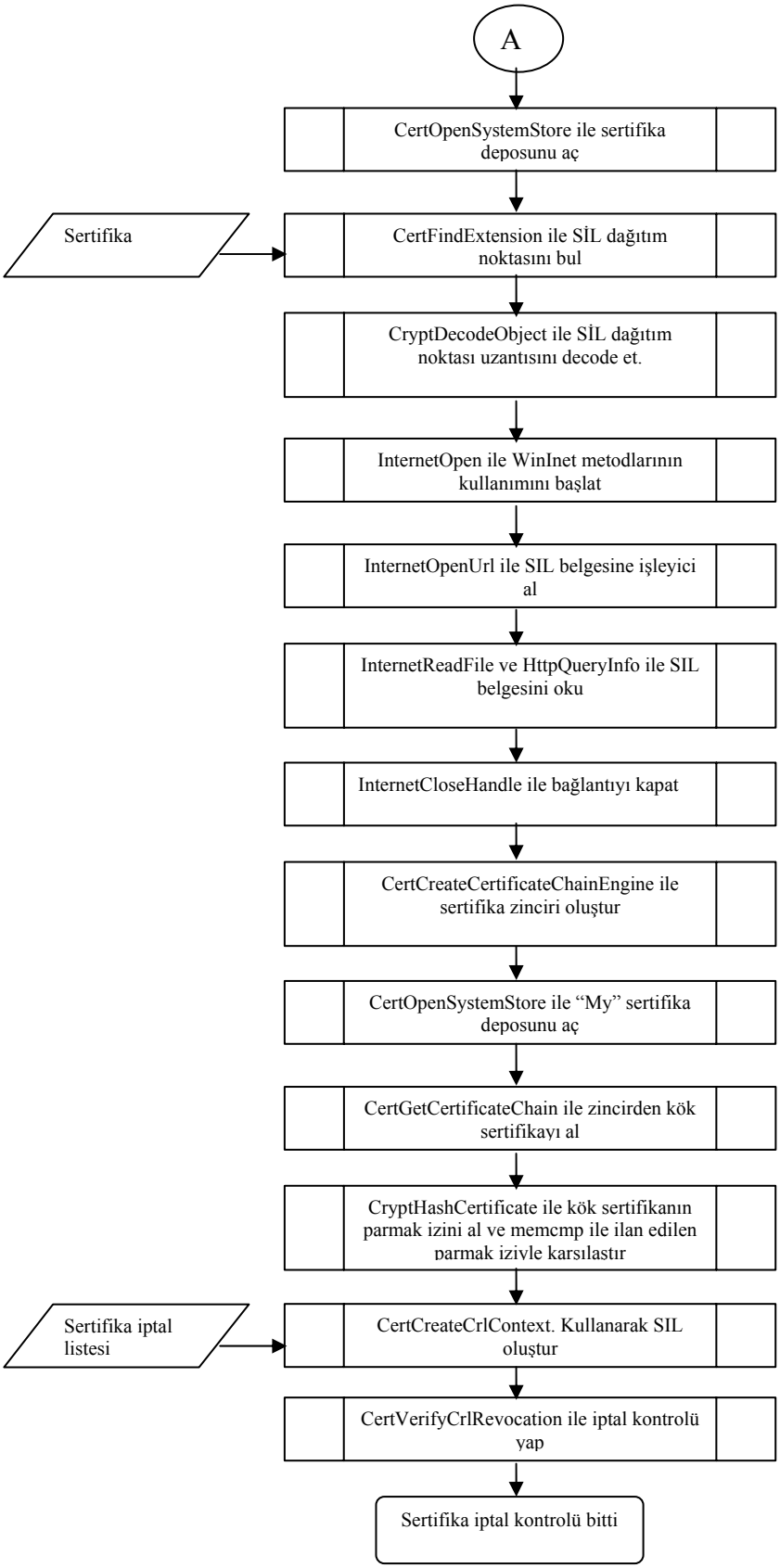
1 İnternetten indirilen SİL dosyası ile SİL yapısını oluşturmak için CertCreateCrlContext [14].
2 Sertifika iptal kontrolü için CertVerifyCrlRevocation.
3 Sonuç 1 ise sertifika geçerli, sonuç 0 ise sertifika geçersizdir.



Şekil 6.3. Decode ve imza doğrulamada kullanılan algoritma



Şekil 6.4. Sertifika iptal kontrolü algoritması



Şekil 6.4. (devam) Sertifika iptal kontrolü algoritması

6.3.4. Kullanılan MS CryptoApi metodları

Kullanılan bu metodları özetlemek gerekirse:

1 *CryptMsgOpenToDecode* metodu decode edilecek mesajı açar ve açılan mesajın işleyicisini (handle) döner. CryptMsgClose çağrılana kadar mesaj açık kalır.

```
HCRYPTMSG WINAPI CryptMsgOpenToDecode(
  DWORD dwMsgEncodingType,
  DWORD dwFlags,
  DWORD dwMsgType,
  HCRYPTPROV hCryptProv,
  PCERT_INFO pRecipientInfo,
  PCMSG_STREAM_INFO pStreamInfo
);
```

2 *CryptMsgUpdate* metodu kriptografik mesaja içerik ekler. Bu metod sayesinde mesaj parça parça yapılandırılabilir. Eklenen içerik açılan mesaja bağlı olarak encoded yada decoded olabilir.

```
BOOL WINAPI CryptMsgUpdate(
  __in  HCRYPTMSG hCryptMsg,
  __in  const BYTE *pbData,
  __in  DWORD cbData,
  __in  BOOL fFinal
);
```

3 *CryptMsgGetParam* metodu kriptografik mesaj decode edildikten sonra mesajdan bir parametre almak için kullanılır.

```
BOOL WINAPI CryptMsgGetParam(
  __in     HCRYPTMSG hCryptMsg,
```

__in DWORD dwParamType,
__in DWORD dwIndex,
*__out void *pvData,*
*__inout DWORD *pcbData*
);

4 *CertGetSubjectCertificateFromStore* metodu sertifika deposundan, yayımlayıcısı ve seri numarası belli olan sertifikayı çekmek için kullanılır.

PCCERT_CONTEXT WINAPI CertGetSubjectCertificateFromStore(
__in HCERTSTORE hCertStore,
__in DWORD dwCertEncodingType,
__in PCERT_INFO pCertId
);

5 *CryptMsgControl* metodu ile çözümlenen (decode) mesaj kontrol edilir. Kontrol işlemi kod açma, imza ve özet doğrulama, sertifika ekleme ve silme, SİL ve imzalayan doğrulanması işlemlerinden ibarettir.

BOOL WINAPI CryptMsgControl(
__in HCRYPTMSG hCryptMsg,
__in DWORD dwFlags,
__in DWORD dwCtrlType,
*__in const void *pvCtrlPara*
);

6 *CertOpenSystemStore* metodu sistem sertifika depolarını açar. Daha karmaşık gereksinimlerle örneğin belge tabanlı yada bellek tabanlı depo için CertOpenStore kullanılır.

HCERTSTORE WINAPI CertOpenSystemStore(
__in HCRYPTPROV_LEGACY hprov,

```
  __in  LPTCSTR szSubsystemProtocol );
```

7 *CertFindExtension* metodu CERT_EXTENSION dizisinde nesne belirteci (OID) verilen ilk uzantıyı bulur. Decode edilmiş sertifikada CERT_INFO yapısı elde edilir.

```
PCERT_EXTENSION WINAPI CertFindExtension(
  __in  LPCSTR pszObjId,
  __in  DWORD cExtensions,
  __in  CERT_EXTENSION rgExtensions[]
);
```

8 *CryptDecodeObject* metodu lpszStructType parametresi ile gösterilen yapıyı çözümler (decode).

```
BOOL WINAPI CryptDecodeObject(
  __in     DWORD dwCertEncodingType,
  __in     LPCSTR lpszStructType,
  __in     const BYTE *pbEncoded,
  __in     DWORD cbEncoded,
  __in     DWORD dwFlags,
  __out    void *pvStructInfo,
  __inout  DWORD *pcbStructInfo
);
```

9 *InternetOpen* metodu WinInet kütüphanesi metodlarının kullanımını başlatır.

```
HINTERNET InternetOpen(
  __in  LPCTSTR lpszAgent,
  __in  DWORD dwAccessType,
  __in  LPCTSTR lpszProxyName,
  __in  LPCTSTR lpszProxyBypass,
```

```
  __in  DWORD dwFlags
);
```

10 *InternetOpenUrl* metodu FTP, Gopher yada HTTP URL ile belirtilen kaynağı açar.

```
HINTERNET InternetOpenUrl(
  __in  HINTERNET hInternet,
  __in  LPCTSTR lpszUrl,
  __in  LPCTSTR lpszHeaders,
  __in  DWORD dwHeadersLength,
  __in  DWORD dwFlags,
  __in  DWORD_PTR dwContext
);
```

11 *HttpQueryInfo* HTTP request ile ilgili başlık (header) bilgisini alır.

```
BOOL HttpQueryInfo(
  __in     HINTERNET hRequest,
  __in     DWORD dwInfoLevel,
  __inout  LPVOID lpvBuffer,
  __inout  LPDWORD lpdwBufferLength,
  __inout  LPDWORD lpdwIndex
);
```

12 *InternetReadFile* metodu InternetOpenUrl, FtpOpenFile, GopherOpenFile, yada HttpOpenRequest ile açılan kaynaktan veri okur.

```
BOOL InternetReadFile(
  __in   HINTERNET hFile,
  __out  LPVOID lpBuffer,
  __in   DWORD dwNumberOfBytesToRead,
```

```
  __out  LPDWORD lpdwNumberOfBytesRead
);
```

13 *CertCreateCertificateChainEngine* metodu uygulama için yeni bir sertifika zinciri oluşturur. Bu zincir de doğrulama için gerekli kök sertifikalar, sertifika güven listesindeki sertifikalar ve sertifika kontrolleri arasındaki süreler sınırlıdır.

```
BOOL WINAPI CertCreateCertificateChainEngine(
  __in   PCERT_CHAIN_ENGINE_CONFIG pConfig,
  __out  HCERTCHAINENGINE *phChainEngine
);
```

14 *CertGetCertificateChain* metodu son sertifikadan güvenilir kök sertifikaya kadar sertifika zinciri yapısını oluşturur.

```
BOOL WINAPI CertGetCertificateChain(
  __in_opt  HCERTCHAINENGINE hChainEngine,
  __in      PCCERT_CONTEXT pCertContext,
  __in_opt  LPFILETIME pTime,
  __in      HCERTSTORE hAdditionalStore,
  __in      PCERT_CHAIN_PARA pChainPara,
  __in      DWORD dwFlags,
  __in      LPVOID pvReserved,
  __out     PCCERT_CHAIN_CONTEXT *ppChainContext
);
```

15 *CryptHashCertificate* metodu ile imzasıyla birlikte encode edilmiş tüm sertifika yapısının özeti (hash) alınır.

```
BOOL WINAPI CryptHashCertificate(
```

```
  __in     HCRYPTPROV_LEGACY hCryptProv,
  __in     ALG_ID Algid,
  __in     DWORD dwFlags,
  __in     const BYTE *pbEncoded,
  __in     DWORD cbEncoded,
  __out    BYTE *pbComputedHash,
  __inout  DWORD *pcbComputedHash
);
```

16 *Memcmp* metodu iki bufferdaki karakterleri karşılaştırır.

```
int memcmp(
   const void *buf1,
   const void *buf2,
   size_t count
);
```

17 *CertCreateCRLContext* metodu SİL yapısını oluşturur.

```
PCCRL_CONTEXT WINAPI CertCreateCRLContext(
  __in  DWORD dwCertEncodingType,
  __in  const BYTE *pbCrlEncoded,
  __in  DWORD cbCrlEncoded
);
```

18 *CertVerifyCRLRevocation* metodu SİL'de sertifika iptal kontrolü yapar.

```
BOOL WINAPI CertVerifyCRLRevocation(
  __in  DWORD dwCertEncodingType,
  __in  PCERT_INFO pCertId,
  __in  DWORD cCrlInfo,
  __in  PCRL_INFO rgpCrlInfo[] );
```

7. ETSI UYUMLU KÜTÜPHANE İLE ACTIVEX GELİŞTİRME

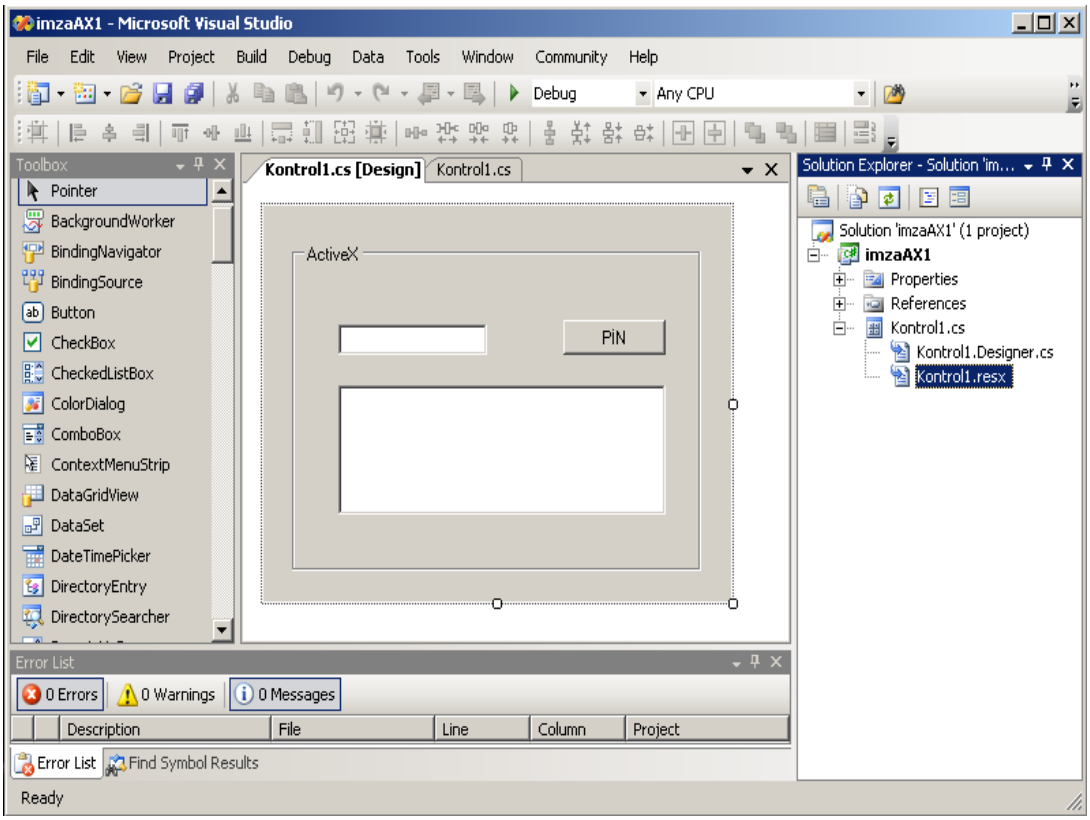
Günümüzde geliştirilen elektronik imza yazılımlarının doküman yönetim sistemleri gibi web tabanlı sistemlerde uyumlu olarak çalışabilmeleri bir zorunluluktur. Elektronik imza akıllı kartlarında (smart card) bulunan gizli anahtarların dışarıya verilmesi mümkün olmadığından elektronik imza işlemlerinin istemci (client) bilgisayarlarda yapılması gerekir. Bu nedenle istemci bilgisayarlara indirilen java applet, activeX, adobe flex gibi uygulamalar kullanılmalıdır [6, 8]. ETSI 101733 uyumlu olarak geliştirilen BesImza.dll elektronik imza modülü kullanarak activeX tabanlı bir program oluşturulmuştur [4, 8]. ActiveX, Microsoft'un windows platformları için geliştirdiği bir nesne bileşeni modelidir (COM) [8]. Yazılım tabanlı olan activeX teknolojisi internet explorer eklentisi ve web sayfalarına iliştirilmiş activeX tabanlı uygulama olarak çalışır [8]. Elektronik imza activeX'in istemci bilgisayarda çalışabilmesi için istemci bilgisayarda .Net framework'ün kurulu olması gerekir [43].

7.1 .Net Ortamında ActiveX Oluşturmak

.Net ortamında activeX oluşturmak için öncelikle Visual Studio .Net'de windows kontrol projesi oluşturulmadır ve COM dünyasına uygun bir arayüz oluşturulmalıdır [8, 43, 44]. Maddeler halinde sıralarsak:

- Kullanıcı kontrolü (user control) assembly oluşturulur.(class library project)
- Kontrol için arayüz oluşturulur.
- Web sayfasına kullanıcı kontrolü eklenir.
- Web formundan kontrole veri aktarılır ve veri kontrolde gösterilir.

Öncelikle sınıf kütüphanesi (class library) tipinde yeni bir proje oluşturulur. Proje oluşturulunca Class1.cs silinir. Projeye çözüm arayıcıdan (solution explorer) sağ klikle kullanıcı kontrolü (user control) eklenir. Bu kontrol üstünde gerekli kullanıcı arayüzü, alet kutusu (toolbox) kullanarak Şekil 7.1'deki gibi oluşturulur.



Şekil 7.1. Visual Studio .Net'de activeX oluşturmak

Kullanıcı arayüzünden sonra "Interface" anahtar elementi eklenmelidir. Bu interface, COM/COM+ nesnelerinin erişimini sağlar [44]. Aşağıdaki *mStr_UserText* dizgisi (string) web formdan kontrole aktarılacak veriyi tutar. Bu dizgi kontrolün içindedir. Daha sonra genel özellik (public property) oluşturulur. Web sayfası bu property ile kontrole parametre geçebilir.

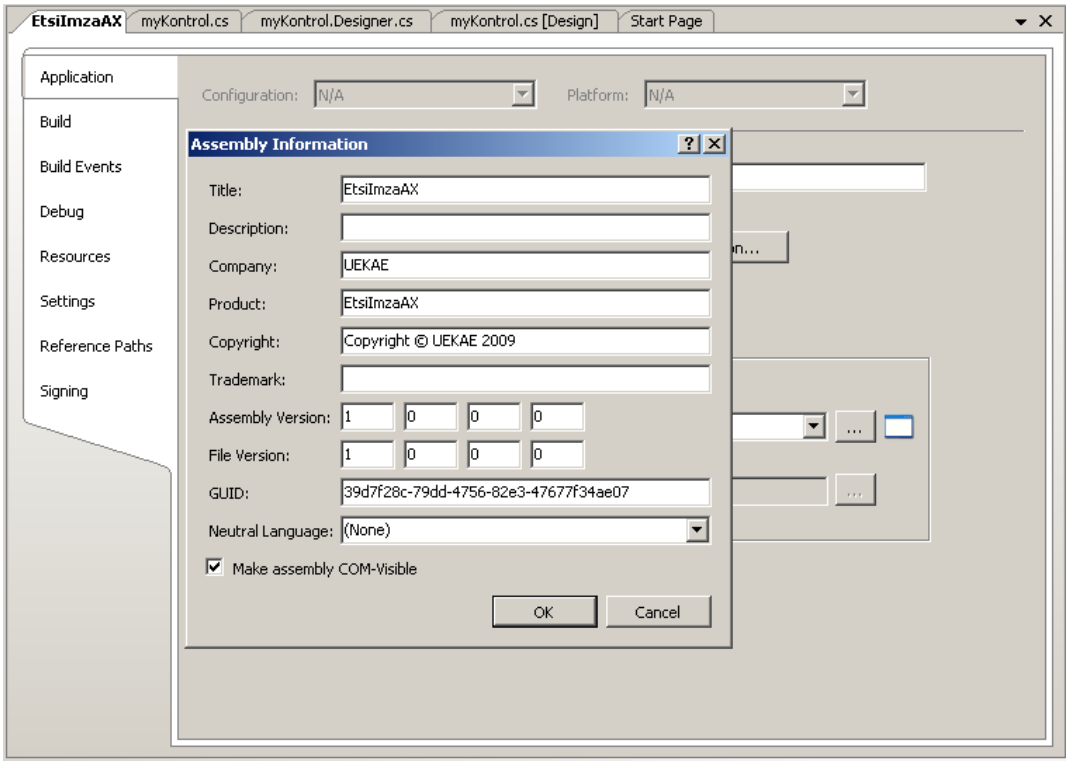
```
private String mStr_UserText;

public String UserText
{
get { return mStr_UserText; }
set
{
mStr_UserText = value;
  txtUserText.Text = value;
}
}
```

Bu örnekte web formundan geçilen *mStr_Usertext* parametresi kontrolde bulunan *txtUsertext* alanını güncelliyor.

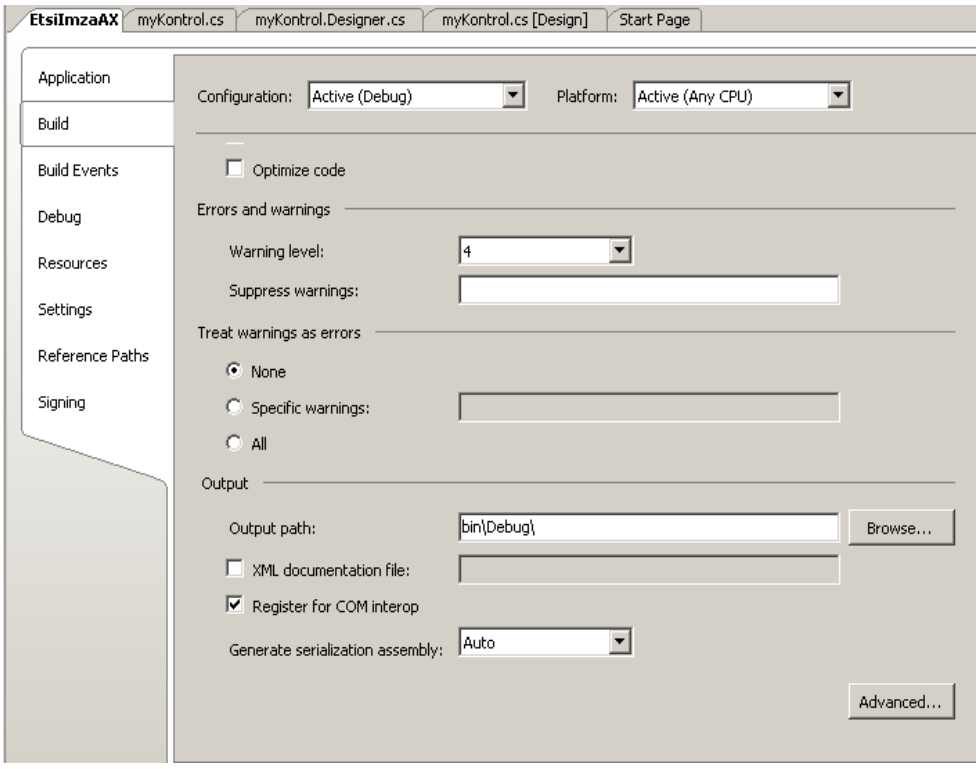
.Net assembly'lerin kullanabileceği public property'yi COM dünyasında erişilebilir hale getirmek için AxMyControl arayüzü oluşturup kontrolü bu arayüzün mirasını kullandırmamız gerekir [43, 44]. Bu sayede COM nesneleri kontrole erişebilir ve .Net assembly activeX kontrol gibi davranabiliyor [8, 44]. Kod aşağıdaki şekilde olabilir.

```
namespace imzaAX1
{
public interface AxMyControl
{
String UserText { set; get ; }
}
public class myControl : System.Windows.Forms.UserControl, AxMyControl
{
private String mStr_UserText;
public String UserText
{
get { return mStr_UserText; }
set
{
mStr_UserText = value; .
txtUserText.Text = value;
}
}
```

Projede properties-> application sekmesinde "Assembly information"->"Make COM visible" seçili olmalıdır (Şekil 7.2) ve properties->Build sekmesinde "Register for COM interop" seçili olmalıdır (Şekil 7.3).



Şekil 7.2 .“Make assembly COM-visible” seçili olmalı



Şekil 7.3. “Register for COM Interop” seçili olmalıdır

Kontrolü web sayfasından aşağıdaki gibi çağırabiliriz.

```
<html>
 <body color=white>
  <hr>
     <font face=arial size=1>
      <OBJECT id="Kontrol1" name="Kontrol1"
classid="imzaAX1.dll#imzaAX1.Kontrol1" width=288 height=72>
      </OBJECT>
    </font>
    <form name="frm" id="frm">
      <input type="text" name="txt" value="enter text here"><input type=button
value="Click me" onClick="doScript();">
     </form>
  <hr>
 </body>
<script language="javascript">
     function doScript()
      {
       Kontrol1.UserText = frm.txt.value;
      }
</script>
</html>
```

HTML sayfası web sunucu vasıtasıyla yüklendiğinde kontrol sayfaya yüklenir ve kullanıcı arayüzü görülür. İnternet explorer seçeneklerinde güvenilir siteler arasına web sunucunun çalıştığı domainin eklenmesi gereklidir.

7.2. BESImza.dll Metodlarının C# ActiveX İçinden Çağrılması

BESImza.dll metodlarının C# activeX içinden çağırmak için öncelikle activeX [8] kodu içinde aşağıdaki satır ile BESImza.dll modülünün gösterilmesi gerekir.

```
[DllImport("BESImza.dll")]
```

BESImza modülünün metodları, aşağıdaki gibi activeX içinden (yada herhangi bir C# uygulaması) çağrılabilir.

imzaAt metodu:

```
extern static unsafe byte* imzaAt(int len, byte[] imzalanacak, ref int size);
```

iptalKontrolu metodu:

extern static int iptalKontrolu(int len, byte[] dogrulanacak);

sertifikaGoster metodu

extern static void sertifikaGoster(int len, byte[] dogrulanacak);

icerikAl metodu

extern static unsafe byte icerikAl(int len, byte[] dogrulanacak, ref int size);*

imzaciCertAl metodu

extern static unsafe byte imzaciCertAl(int len, byte[] dogrulanacak, ref int size);*

7.3. C# ActiveX Bileşenlerinin CAB ile Dağıtılması

C++, COM ve ATL ile oluşturulan istemci tarafı bileşenlerinin oluşturulması ve dağıtılması kolay ve çok bilinen bir şekilde yapılmaktadır. ActiveX bileşeninin oluşturulması ve INF dosyası ile CAB şeklinde paketlenmesi yeterli olmaktadır [8]. C# ile oluşturulan activeX bileşenlerinde ise prosedür çok farklı olmalıdır [8]. Çünkü ilk durumdaki INF dosyasında bulunan RegisterServer=YES satırı ile çağrılan regsvr32, C# bileşenini kayıt (register) edemez. Burada temel olarak CAB dosyasının oluşturulması ve sunucuda dağıtılması anlatılacaktır. Aşağıda basit bir C# bileşeninin COM arayüzü ile nasıl çağrıldığı görülmektedir.

```
namespace imzaAX1
{
   [Guid("BC9E9444-8DBC-4870-B369-E281DDEFF073")]

   public interface AxMyControl
   {
      String UserText { set; get; }
   }

   [ClassInterface(ClassInterfaceType.AutoDual)]
   [Guid("3F98CA64-CB64-4cee-8A3D-3ACA7DB1DDC0")]

   public partial class Kontrol1 : UserControl, AxMyControl
   {
      string veri = "";
      public Kontrol1()
```

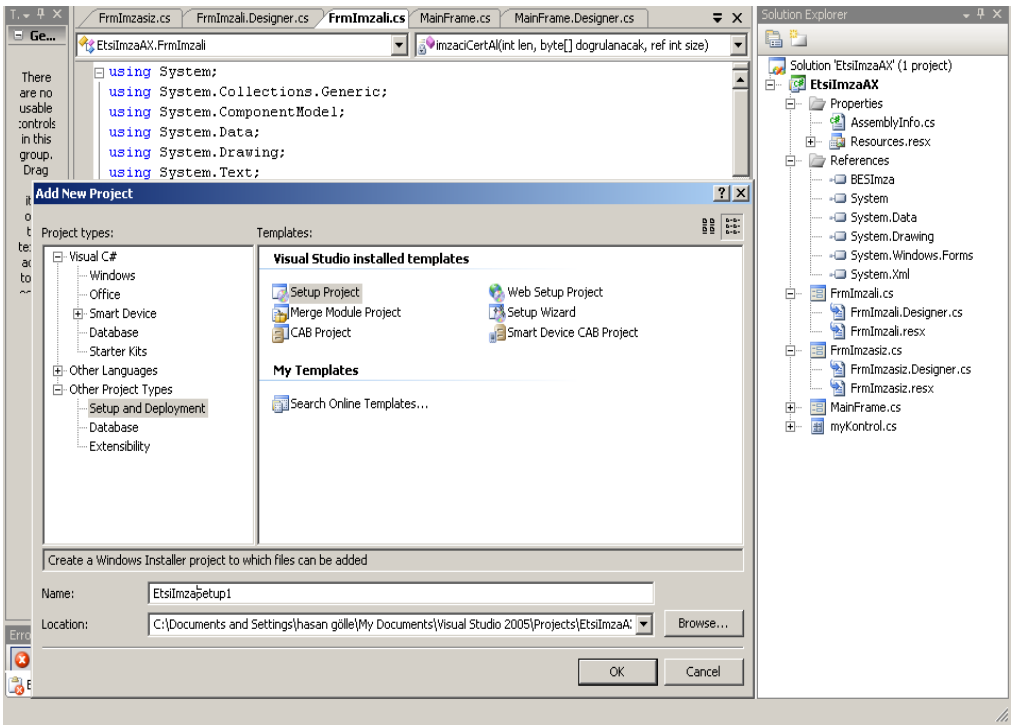
```
        {
            InitializeComponent();
        }
        private String mStr_UserText;
        public String UserText
        {
            get { return mStr_UserText; }
            set
            {
                mStr_UserText = value;
                tbPin1.Text = value;
            }
        }
         //Gerekli kodlamalar burada.
    }
}
```

7.3.1. MSI setup projesi ile installer oluşturma

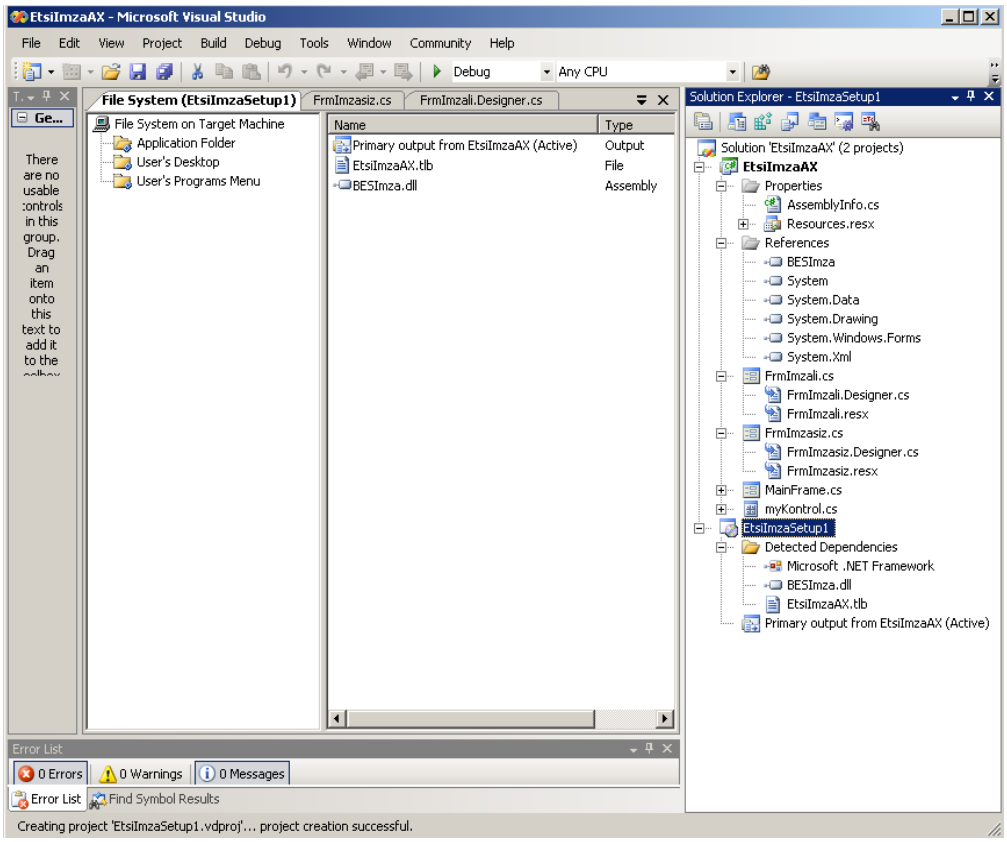
Bu bölümde "Writing an ActiveX Control in .NET" makalesinden faydalanılmıştır [45]. Bir önceki bölümde oluşturulan BESImza.dll modülü projeye referans olarak eklenmelidir. Projede kullanılan MainFrame, FrmImzasiz ve FrmImzali formları oluşturulup projeye eklenir. Daha sonra "solution exlorer->add->new Project->setup and deployment" yapılarak projeye installer modülü Şekil 7.4'deki gibi eklenir.

Oluşturulan EtsiImzaSetup1 installer üstünde "add->Project output-> primary output" yapılarak activeX dll'i Şekil 7.5'deki gibi yükleyiciye (installer) eklenir. Build'den sonra projede EtsiImzaSetup1\Debug dizini altında EtsiImzaSetup1.msi dosyası oluşmuştur. Bu installer dosyasının istemci bilgisayarda kurulabilmesi için bir INF dosyası ile birlikte paketlenmesi gerekir. CL1.inf dosyasının içeriği aşağıdaki şekildedir:

```
[version]
signature="$CHICAGO$"
AdvancedINF=2.0

[Setup Hooks]
hook1=hook1
[hook1]
run=msiexec.exe /i "%EXTRACT_DIR%\EtsiImzaSetup1.msi" /qn
```



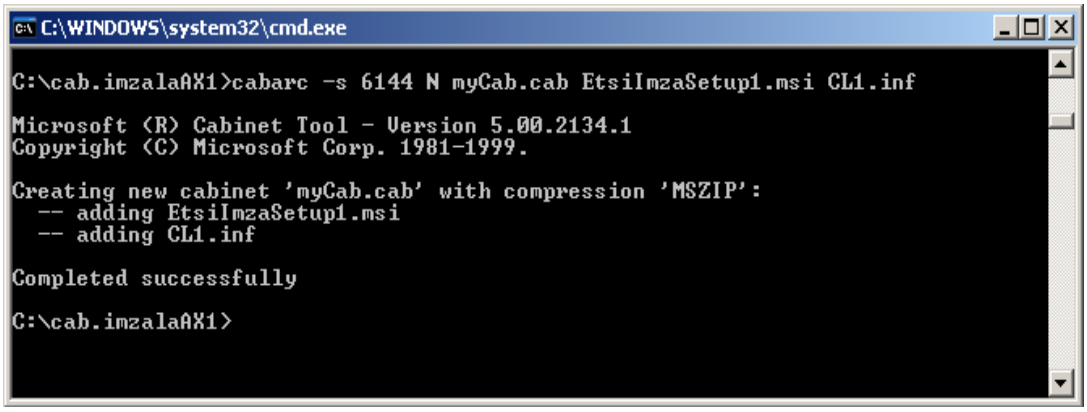
Şekil 7.4. MSI Setup projesi ile installer oluşturma



Şekil 7.5. MSI Setup projesi ile installer oluşturma

7.3.2. Cab oluşturma

Microsoft'un bir aracı olan "cabarc" ile EtsiImzaSetup1.msi ve CL1.inf dosyalarını içeren bir cab dosyası Şekil 7.6'daki gibi oluşturabiliriz. Oluşturulan cab dosyasının istemci bilgisayarda sistem kaynaklarına erişip dosya okuyup yazabilmesi için imzalı olması gereklidir. Bunun için bir kod imzalama sertifikası edinip yine Microsoft'un bir aracı olan "signcode" yada "signtool signwizard" kullanılabilir. Bu bölümde "Downloading C# ActiveX Components through CAB File" makalesinden faydalanılmıştır [46].



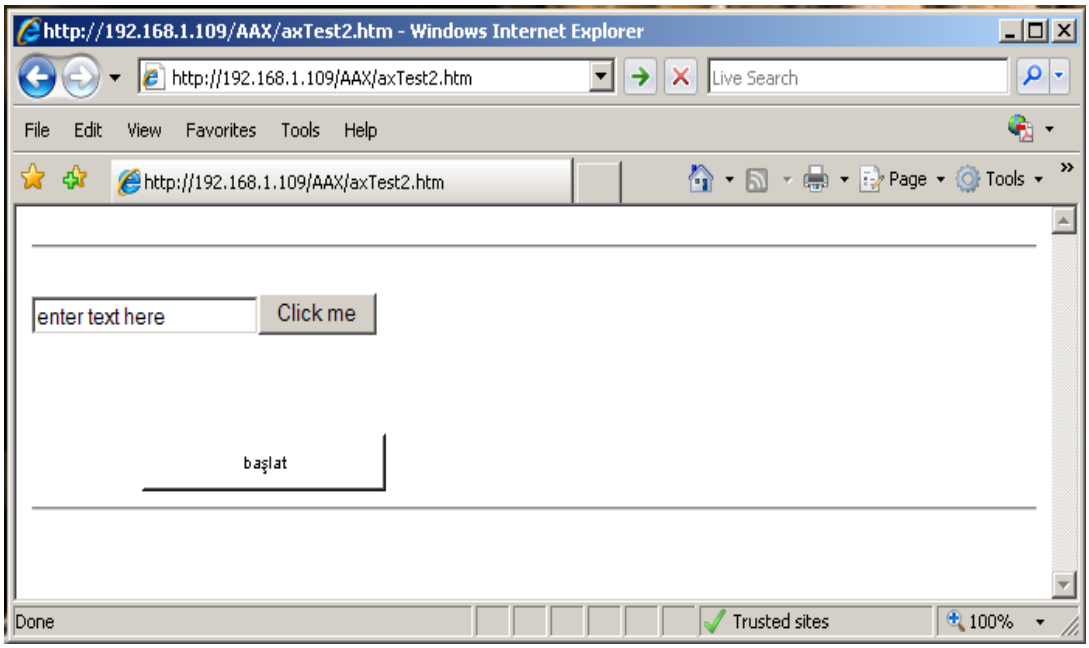
Şekil 7.6. Cab oluşturma

7.3.3. ActiveX cab'in dağıtılması (deploy)

Aşağıdaki gibi html sayfasından activeX cab'i çağrılabilir.

```
<html>
 <body color=white>
  <hr>
       <form name="frm" id="frm">
        <input type="text" name="txt" value="enter text here"><input type=button
value="Click me" onClick="doScript();">
       </form>

       <font face=arial size=1>
        <OBJECT id="myKontrol" name="myKontrol" classid="clsid:A657786C-
0E4A-4173-A92F-C9EFCD97FE32"
       codebase="myCab.cab#8:1.0.0" width=288 height=72>
       </OBJECT>
       </font>
```

```
 <hr>
 </body>
<script language="javascript">
        function doScript()
        {
         myKontrol.UserText = frm.txt.value;
        }
</script>

</html>
```

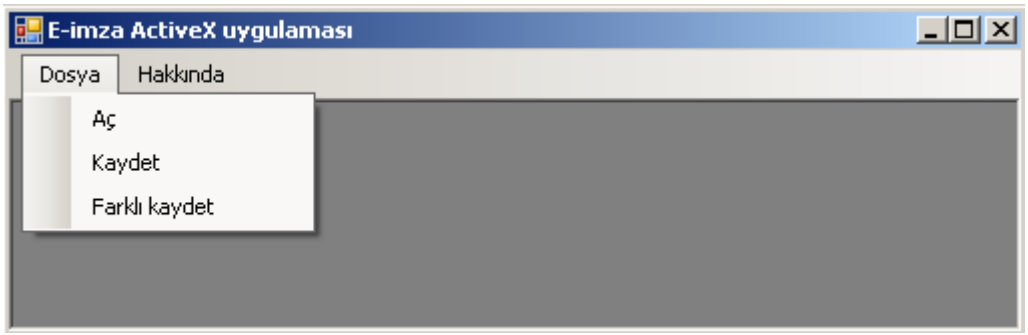


Şekil 7.7. Tarayıcıdan activeX çağırma

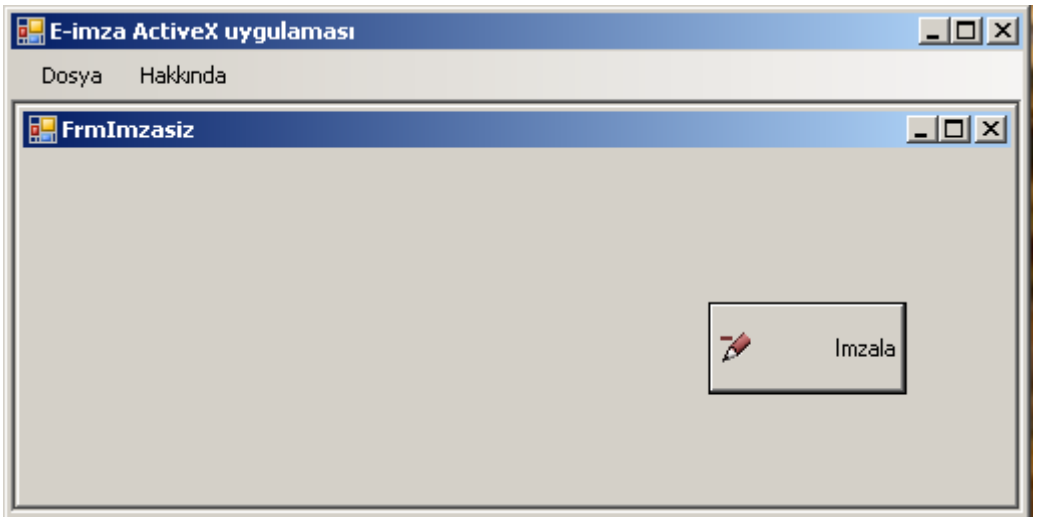
İstemci bilgisayarda internet explorer tarayıcısını Şekil 7.7'deki gibi kullandığımızda activeX uygulaması bu bilgisayara inecek ve kendiliğinden kurulacaktır. Başlat butonu tıklandığında aşağıdaki gibi elektronik imza programı çalışmaya başlayacaktır.

7.4 E-imza Programının Arayüzleri

Şekil 7.8'de görülen pencere ile kullanıcı elektronik imzalamak amacıyla diskten bir dosya seçebilir. Seçilen dosya imzasız bir dosya ise aşağıdaki pencere görülür. (Şekil 7.9)



Şekil 7.8. ActiveX ile imzalanacak dosya seçme

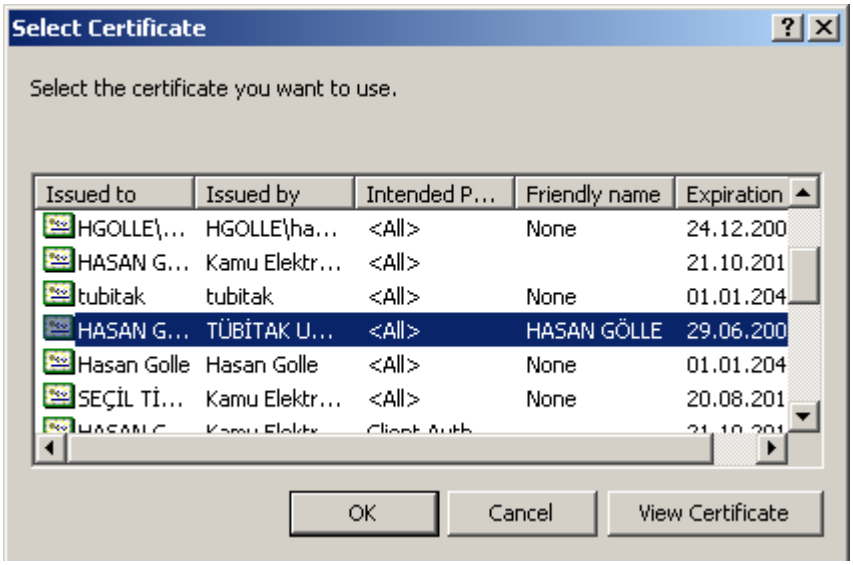


Şekil 7.9. Seçilen dosya imzasız

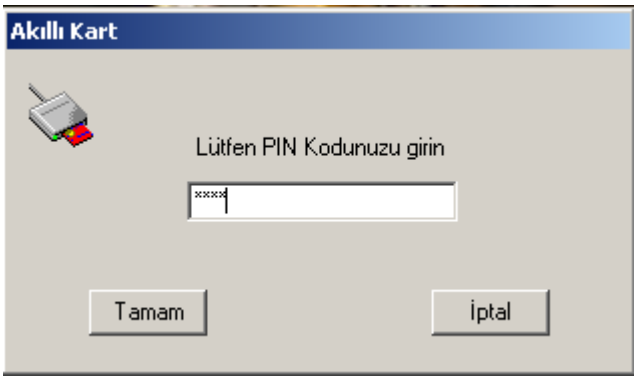
Kullanıcı imzala butonunu tıklarsa elektronik imzalamada kullanacağı sertifikasını Şekil 7.10'daki gibi seçebilir .

Daha sonra program arka planda sertifika doğrulama işlemi yapar. Eğer geçerli bir sertifika ise Şekil 7.11'deki gibi PIN girme ekranı gelir.

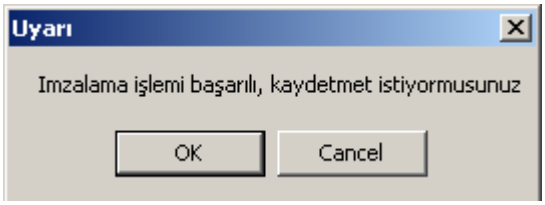
Kullanıcı akıllı kartın PINini doğru girerse elektronik imza oluşturulur ve kullanıcı Şekil 7.12'deki gibi uyarılır.



Şekil 7.10. Kullanıcı sertifikaları seçme ekranı



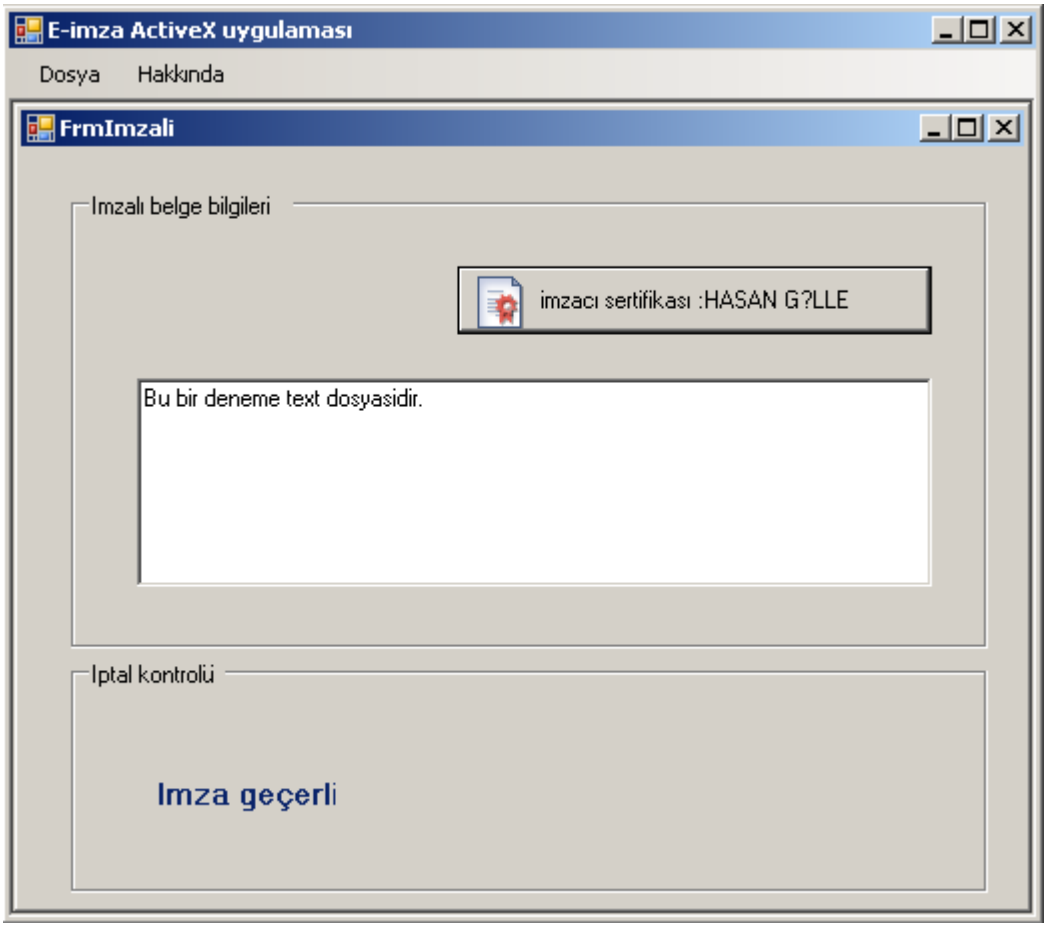
Şekil 7.11. PIN girme ekranı



Şekil 7.12. imzala işlemi başarılı

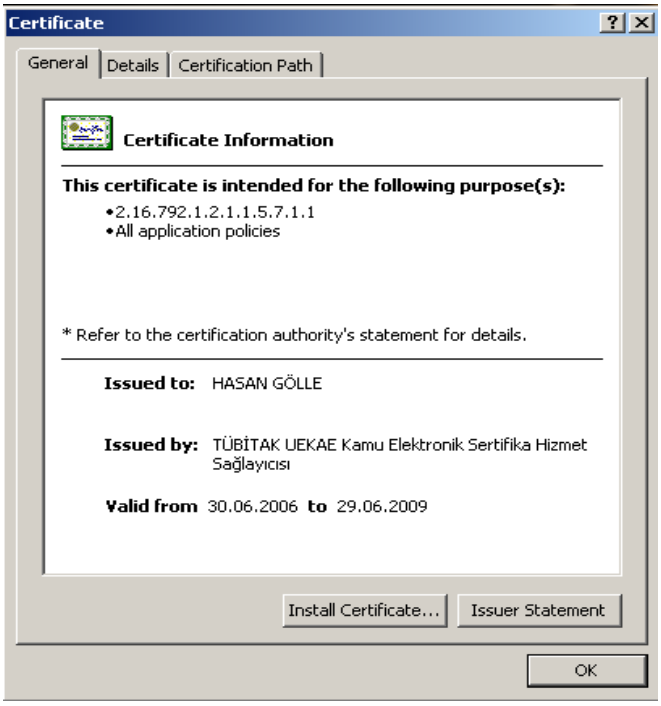
"Ok" butonu tıklanırsa elektronik imzalı .imz uzantılı yeni bir belge oluşturulur ve orijinal belgeyle aynı dizine konur. Yeni belge "Farklı kaydet" seçeneği ile farklı bir dizine de kaydedilebilir.

İlk pencerede "dosya aç" ile açılan belge eğer imzalı bir belge ise program imzalı belge üstünde bulunan elektronik imzanın ve imzacı sertifikasının geçerlilik kontrolünü yapar. Orijinal imzasız belgeyi ve sertifikayı ayrıştırır ve Şekil 7.13'deki gibi kullanıcıya gösterilir.

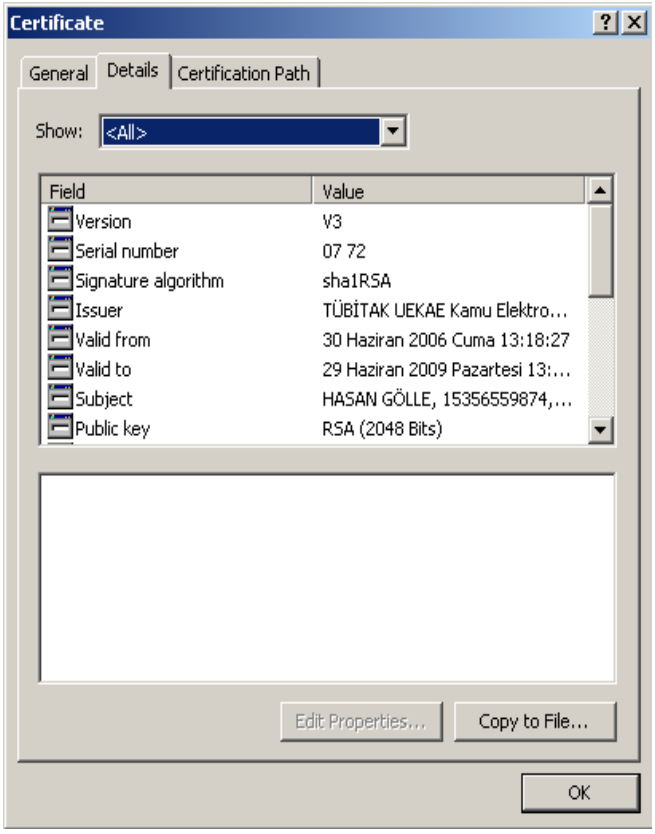


Şekil 7.13. Seçilen dosya imzalı

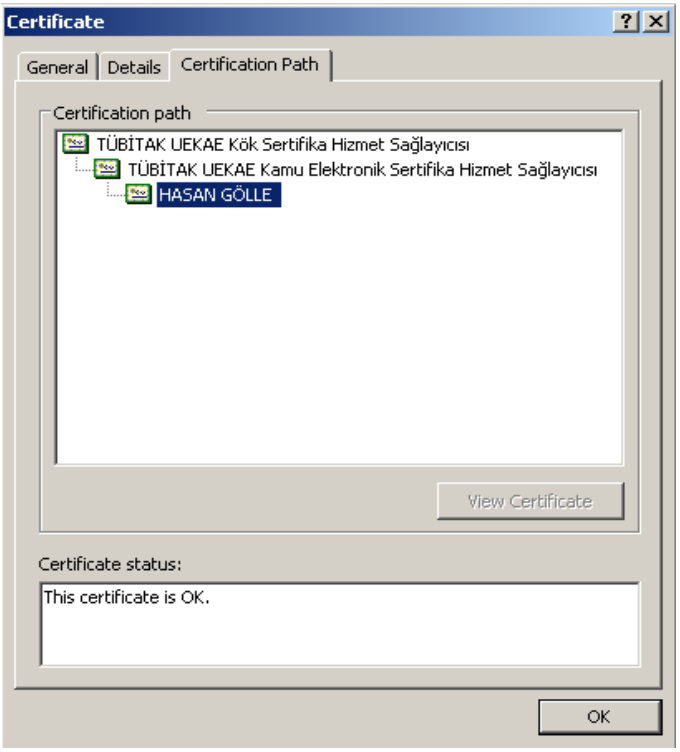
İptal kontrolü sonucu bu ekranda "imza geçerli" yada "imza geçersiz" şeklinde kullanıcıya bildirilir. İmzacı sertifikası butonunda imzacı sertifikasının sahibinin adı görülür. Kullanıcı, bu butonu tıklarsa Şekil 7.13, Şekil 7.14 ve Şekil 7.15'deki gibi sertifikasının ayrıntılarını görebilir.



Şekil 7.14. Kullanıcı sertifikası



Şekil 7.15. Kullanıcı sertifikası



Şekil 7.16. Kullanıcı sertifikası

7.5. Genel Değerlendirme ve Test

Sonuçların değerlendirilmesi amacıyla, elde edilen elektronik imzalı belgeler Tübitak tarafından geliştirilen İmzager API ile Şekil 7.17'deki gibi açıldığında bu imzaların ETSI 101733 uyumlu olduğu ve geçerli olduğu görülmektedir [4, 37].

ActiveX uygulaması ile elde edilen elektronik imzalı belgeler ayrıca Bölüm 5.2'de bahsedilen e-imza applet programı ile de açılmıştır. Şekil 7.18'de bu programın ekran görüntüsü verilmiştir. Yine oluşturulan elektronik imzalı belgelerin ETSI 101733 uyumlu olduğu ve başarılı bir şekilde doğrulanabildiği görülmektedir [4]. Her iki uygulama (BESImza.dll ve EtsiImzaAX), daha geniş kapsamlı projelerin (doküman yönetim sistemi vs.) bir parçası olarak kullanılabilir.

Tezin gerçekleştiriminde yaşanan sıkıntılardan biri, BesImza.dll elektronik imza modülünün ürettiği ve byte dizisi şeklinde döndürdüğü imzalı dosyayı activeX yada

herhangi bir c# uygulamasında alıp işlemek olmuştur. Bu amaçla BesImza.dll içinde imzaAt metodu tanıtılırken aşağıdaki gibi BYTE* terimi kullanıldı.

```
extern "C" __declspec(dllexport) BYTE * imzaAt(int len, BYTE * imzalanacak , int * size);
```

C# ile yazılan activeX içindeyse imzaAt metodu aşağıdaki gibi tanımlanmıştır.

```
extern static unsafe byte* imzaAt(int len, byte[] imzalanacak, ref int size);
```

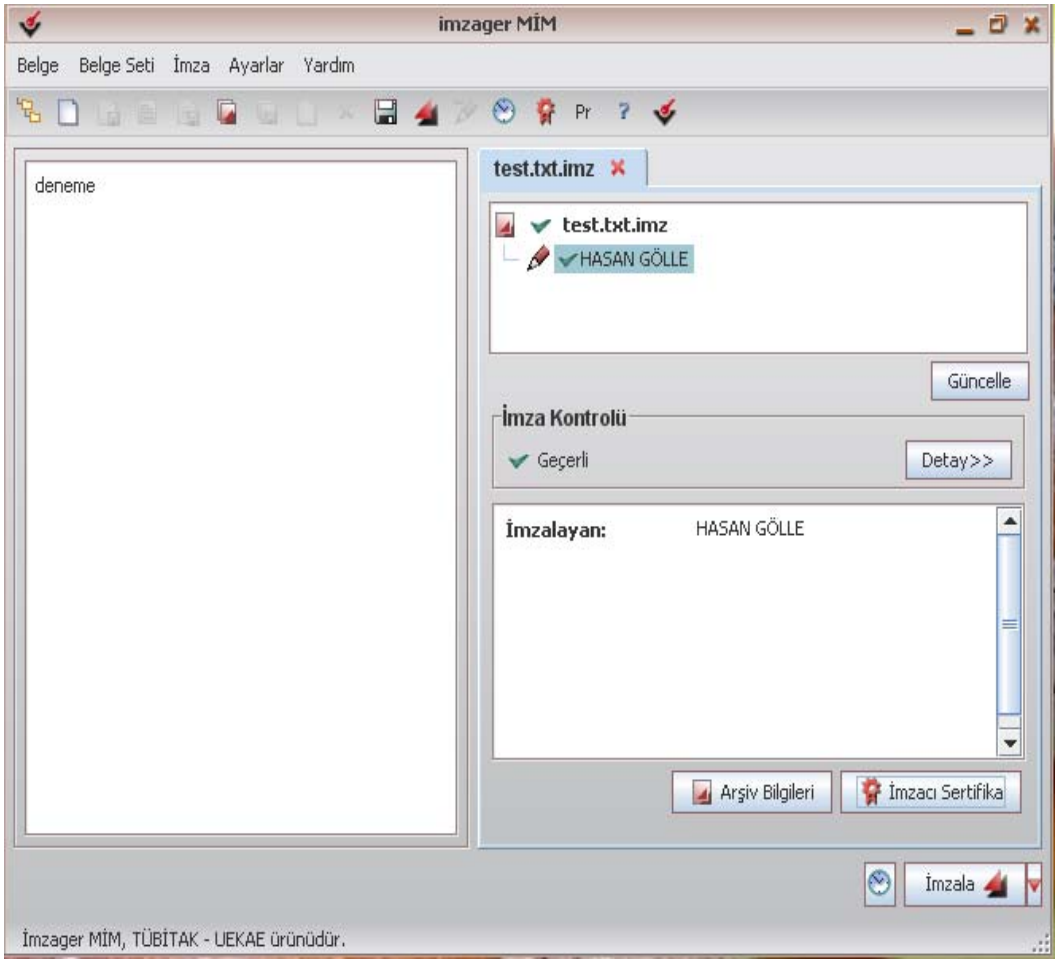
Ayrıca, activeX içinde aşağıdaki satırla elektronik imza modülü tanıtılmalıdır.

```
[DllImport("C:\\BESImza.dll", CallingConvention = CallingConvention.Cdecl)]
```

C# ile yazılmış activeX kodu içinde işaretçi (pointer) kullanıldığı için imzalı dönen veri activeX içinde aşağıdaki gibi unsafe kullanılarak kontrol edilir.

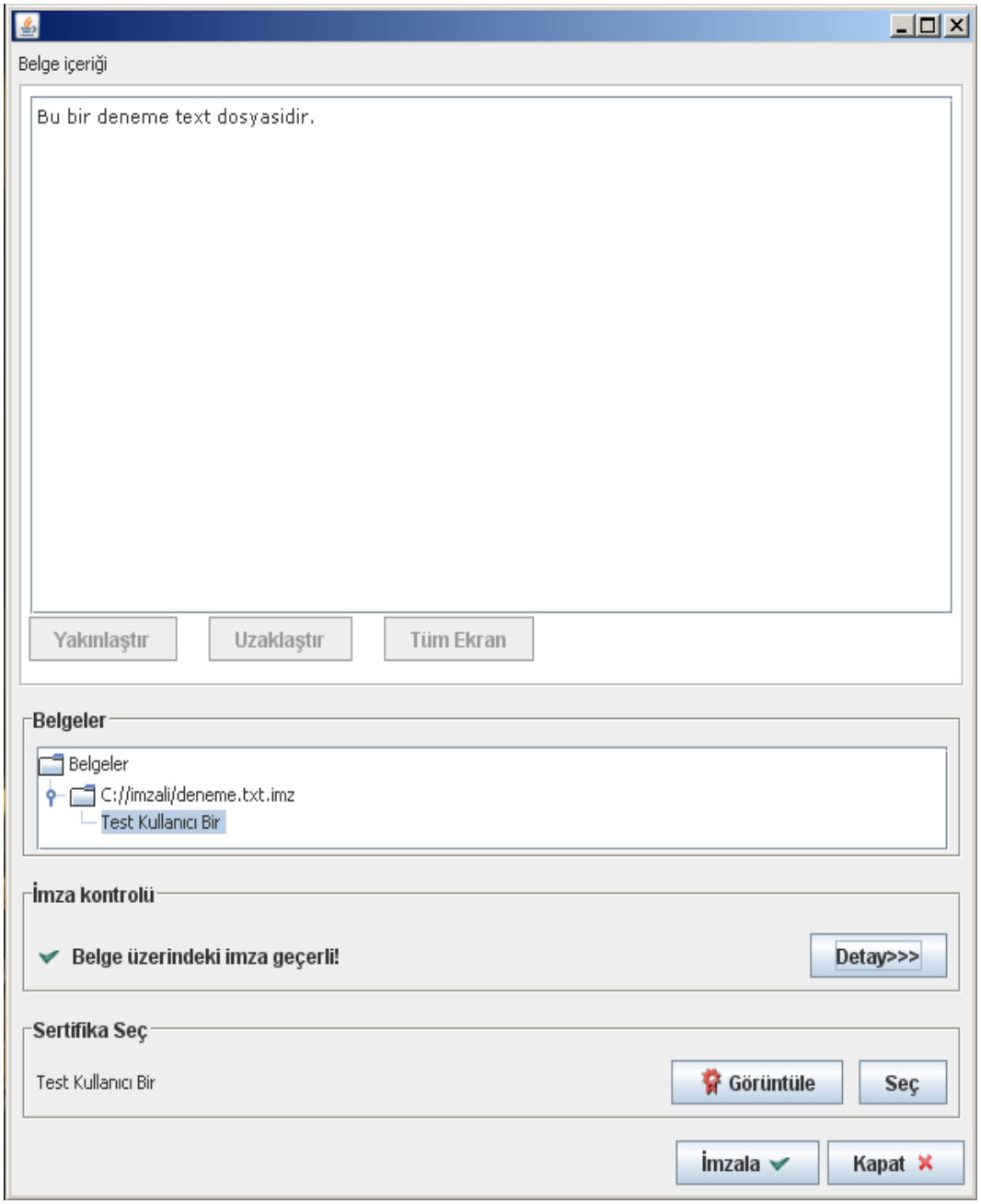
```
unsafe
{
        byte* imzalanacak = null;
        byte* imzali = null;
        int size = 0;
        imzali = imzaAt(len, okunan, ref size);
}
```

.Net C# kullanarak web tabanlı activeX uygulaması geliştirmek de çok bilinen ve kullanılan bir yöntem değildir. C++, COM ve ATL ile activeX gibi istemci tarafı bileşenlerinin oluşturulması ve dağıtılması çok daha kolay bir şekilde yapılmaktadır. Bu nedenle .Net C# kullanarak activeX uygulaması geliştirmek ve oluşturulan activeX'in dağıtılması da çözülmesi gerekli ciddi bir sıkıntı oluşturmuştur. Bu sıkıntıyı aşmak için "Writing an ActiveX Control in .NET" makalesinden ve "Downloading C# ActiveX Components through CAB File" makalesinden faydalanılmıştır [45, 46].



Şekil 7.17. Oluşturulan elektronik imzalı belgelerin İmzager API ile açılması

İstemci bilgisayarlar, sertifika iptal kontrolü yapmak amacıyla ESHS'nin sertifika iptal listesi sunucularına bağlanmak istediğinde, bu sunucunun activeX'in indirildiği sunucu bilgisayarla farklı olması nedeniyle bölge çakışması (cross domain) problemi çıkmaktadır. Bu problemin çözümü, istemci bilgisayarlarda "system32/drivers/etc/hosts" dosyasında www.kamusm.gov.tr adresi ile activeX sunucu bilgisayarı eşleştirmesi yapmak olabilir. Daha kalıcı bir çözüm için "cross domain" probleminin çözülmesi gerekmektedir.



Şekil 7.18. Elektronik imzalı belgelerin e-imza applet ile açılması (Şekil 5.3)

8. SONUÇ

Ülkemizde halen elektronik imza uygulamaları web sistemlerine erişim (login) amaçlı olarak kullanıldığı gibi, doküman yönetim sistemlerinde dolaşan belgelere nitelikli elektronik imza eklemek amacıyla da kullanılabilmektedir. Adalet Bakanlığında 45000 kullanıcı için satın alınan elektronik imza sertifikalarının toplam maliyetini bir yıl içinde kağıt kullanılmaması, havale, posta masraflarının azalması, zamandan kazanım gibi girdiler karşılamıştır. Elektronik imza kullanımının ülkemiz için ne kadar gerekli ve zorunlu olduğu gerçeğini bu örnek ortaya koymaktadır.

Bu tezin gerçekleştirimi iki ana bölümden oluşmaktadır. İlk bölümde Microsoft CryptoApi'nin sağladığı en temel kriptografik metodları kullanarak ETSI 101733 uyumlu elektronik imza üreten ve elektronik imzaları doğrulayan, elektronik sertifika iptal kontrolü yapan, imzalı mesajdan orijinal imzasız mesajı ve kullanıcı sertifikasını bulup kullanıcıya gösteren 5 adet kullanımı kolay metodu bulunan bir elektronik imza kütüphanesi oluşturulmuştur (BESImza.dll). Bu kütüphaneyi kullanan birinin ETSI 101733 uyumlu elektronik imza uygulaması geliştirmesi için çok karmaşık kriptografik metodları kullanmasına gerek yoktur. Kulanımı kolay 5 metodu kullanarak masaüstü yada web tabanlı bir uygulamayı rahatlıkla geliştirebilir.

İkinci bölümdeyse BESImza.dll kütüphanesini kullanarak istemci bilgisayarda çalışan ETSI 101733 uyumlu elektronik imza üreten ve doğrulayan web tabanlı bir activeX uygulaması geliştirilmiştir. ActiveX ile oluşturulan elektronik imzalı belgeler ETSI 101733 standardı üstüne geliştirilen farklı programlarla açılmış ve bu imzaların geçerli olduğu görülmüştür. (Bölüm 7.5)

BESImza.dll modülünü oluştururken ASN.1 yapısını encode edebilmek için gerekli RFC'leri okuyup anlamak ve bu yapıları gerçeklemek zor bir süreçti. .Net teknolojisi ile elektronik imza activeX'i geliştirmek de uzun araştırmalar gerektirmiştir. Bu çalışmaları ve çözümlerini detaylarına kadar tezde vermiş olmam nedeniyle bu

konuda ilerde yapılacak çalışmalar için tezin faydalı bir kaynak olacağını umuyorum.

Bu tez ile gerçekleştirilen CaDES BES elektronik imza yapısı en temel kimlik doğrulama ve bütünlük denetimi sağlayan, ETSI 101733 standardında tanımı yapılmış ilk elektronik imza formatıdır. Uzun dönemli geçerlilik için CaDES BES formatı yeterli değildir. Uzun dönemli geçerliliğin ilk koşulu olan zaman damgasının da elde edilip elektronik imza formatına eklenmesi gerekmektedir. Bu nedenle bir zaman damgası sunucusuna bağlanıp zaman damgası alan ve bunu ETSI 101733 de tanımı verilen ES-T (Zaman damgalı elektronik imza) formatına uygun olarak imza yapısına ekleyen bir zaman damgası istemcisinin de gerçekleştiriminin yapılması gerekmektedir.

KAYNAKLAR

1. İnternet: Official Journal of the European Communities, "Directive 1999/93/EC of the European Parliament and of the Council on a common framework for electronic signatures", http://www.ict.etsi.org/Working_Groups/EESSI/Documents/e-sign-directive.pdf (19.1.2000).

2. Kaliski, B., "Request for Comments: 2315, PKCS#7: Cryptographic Message Syntax Version 1.5", ***IETF, RFC 2315***, 8-15, (1998).

3. Housley, R., "Request for Comments: 3852, Cryptographic Message Syntax (CMS)", ***IETF, RFC 3852***, 5-18, (2004).

4. ETSI TS 101733 V1.5.1 "Electronic Signatures and Infrastructures (ESI); Electronic Signature Formats", (12.7.2003).

5. Sağıroğlu, Ş., Alkan, M., "Her Yönüyle Elektronik İmza (E-İmza)", ***Grafiker Yayınları***, Ankara, 61, (2005).

6. İnternet: SUN Developer Network, "Java applet", http://java.sun.com/applets/, (3.5.2009).

7. İnternet: Microsoft Cryptography API, "The Cryptography API, or How to Keep a Secret" , http://msdn.microsoft.com/en-us/library/ms867086.aspx# (19.8.1996).

8. İnternet: MSDN-Microsoft Developer Network, "Introduction to ActiveX Controls", http://msdn.microsoft.com/en-us/library/aa751972.aspx#, (5.5.2009).

9. Eastlake, D., Reagle, J., Solo, D., "Request for Comments: 3275, (Extensible Markup Language) XML-Signature Syntax and Processing", ***IETF, RFC 3275***, 64-65, (2002).

10. Ramsdell, B., "Request for Comments: 2633, S/MIME Version 3 Message Specification", **RFC 2633**, 8-18, (1999).

11. İnternet: RSA Laboratories, "PKCS #12: Personal Information Exchange Syntax Standard", http://www.rsa.com/rsalabs/node.asp?id=2138, (5.5.2009).

12. İnternet: IETF-Pkix web sitesi, "Public-Key Infrastructure (X.509) (pkix)", http://www.ietf.org/html.charters/pkix-charter.html, (21.8.2008).

13. İnternet: X.509 CCITT. Recommendation X.509: The Directory Authentication Framework, https://ecs.itu.ch/itudoc/itu-t/rec/x/x500up/x509_27505.html (1988).

14. Housley, R., Ford, W., Polk, W., Solo, D., "Request for Comments: 2459, Internet X. 509 Public Key Infrastructure Certificate and CRL Profile", ***IETF, RFC 2459***, 5-17, (1999).

15. İnternet: W3C web sitesi, "Extensible Markup Language (XML) 1.0, W3C Recommendation", http://www.w3.org/TR/1998/REC-xml-19980210 (10.2.1998).

16. İnternet: W3C web sitesi, "XML Schema Part 1: Structures" http://www.w3.org/TR/xmlschema-1/, (28.10.2004).

17. İnternet: W3C web sitesi, "XSL Transformations (XSLT) Version 1.0", http://www.w3.org/TR/1999/PR-xslt-19991008, (8.10.1999).

18. İnternet: W3C web sitesi, "Extensible Stylesheet Language (XSL) Version 1.0", http://www.w3.org/TR/xsl/, (5.12.2006).

19. Eastlake, D., Reagle, J., Solo, D., "Request for Comments: 3075, XML-Signature Syntax and Processing", ***RFC 3075,*** 14-26, (2001).

20. Berners-Lee, T., Fielding, R., Masinter, L., "Request for Comments: 2396, Uniform Resource Identifiers (URI): Generic Syntax", ***IETF, RFC 2396***, 5-22, (1998).

21. Adobe Systems, Adobe Creative Team, "Adobe Acrobat 4.0 Classroom in a book", **Adobe**; 2. edition, 175, (2000).

22. Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C., "Request for Comments: 2560, X509 Internet Public Key Infrastructure Online Certificate Status Protocol –OCSP", ***IETF, RFC 2560***, 4-13, (1999).

23. İnternet, CWA 14171, CEN. Workshop Agreements: " General guidelines for electronic signature verification", ftp://ftp.cenorm.be/PUBLIC/CWAs/e-Europe/eSign/cwa14171-00-2004-May.pdf, (2004).

24. Sağıroğlu, Ş., Alkan, M., "Her Yönüyle Elektronik İmza (E-İmza)", ***Grafiker Yayınları***, Ankara, 75-84, (2005).

25. İnternet: X.200 CCITT. Recommendation X.200: Reference Model of Open Systems Interconnection for CCITT Applications, http://webstore.iec.ch/preview/info_isoiec10745%7Bed1.0%7Den.pdf (1984).

26. İnternet: X.208 CCITT. Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1), http://www.itu.int/rec/T-REC-X.680-200207-S/en (1988).

27. İnternet: X.209 CCITT. Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), http://www.itu.int/rec/T-REC-X.690/en (1988).

28. İnternet: An RSA Laboratories Technical Note, "A Layman's Guide to a Subset of ASN.1, BER, and DER", http://luca.ntop.org/Teaching/Appunti/asn1.html, (1.11.1993).

29. Yeong, W., Howes, T., Kille, S., "Request for Comments: 1487, X.500 Lightweight Directory Access Protocol", ***IETF, RFC 1487***, 5-11, (1993).

30. İnternet: X.520 CCITT. Recommendation X.520: The Directory Selected Attribute Types, http://www.iso.org/iso/catalogue_detail.htm?csnumber=35920 (1988).

31. Housley, R., "Request for Comments: 3369, Cryptographic Message Syntax (CMS)", ***IETF, RFC 3369***, 35-40, (2002).

32. Hoffman, P., " Request for Comments: 2634, Enhanced Security Services for S/MIME", ***IETF, RFC 2634***, 46, (1999).

33. Schaad, J., "Request for Comments: 5035, Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility", ***IETF, RFC 5035***, 6, (2007).

34. Berbecaru D., Lioy, A., Marian, M., "A Framework for Secure Digital Administration", **Proceedings of the EuroWeb**, 22-33, (2001).

35. İnternet: EuroPKI web sitesi, "EuroPKI", http://www.europki.org, (4.5.2009).

36. İnternet: RSA Laboratories, "PKCS #11: Cryptographic Token Interface Standard", http://www.rsa.com/rsalabs/node.asp?id=2133, (1.4.2009).

37. İnternet: Kamu Sertifikasyon Merkezi, "İmzager API", http://www.kamusm.gov.tr/tr/Urunler/Eimza/, (1.3.2009).

38. İnternet: SUN Developer Network, "Java Advanced Imaging (JAI)", http://java.sun.com/javase/technologies/desktop/media/jai/, (3.3.2009).

39. İnternet: TBMM, "Elektronik imza kanunu", http://www.tbmm.gov.tr/kanunlar/k5070.html, (15.1.2004).

40. İnternet: MSDN-Microsoft Developer Network, "Example C Program: Listing the Certificates in a Store", http://msdn.microsoft.com/en-us/library/aa382363(VS.85).aspx#, (26.6.2009).

41. İnternet: MSDN-Microsoft Developer Network, "Example C Program: Signing, Encoding, Decoding, and Verifying a Message", http://msdn.microsoft.com/en-us/library/aa382373(VS.85).aspx, (26.6.2009).

42. Messier, M., Viega, J., "Secure programming cookbook with C & C++" **O'Reilly** 556-562, 530-535, (2003).

43. İnternet: Microsoft Download Center, "Microsoft .NET Framework Version 2.0 Redistributable Package (x86)", http://www.microsoft.com/downloads/details.aspx?familyid=0856eacb-4362-4b0d-8edd-aab15c5e04f5&displaylang=en, (5.5.2009).

44. İnternet: Microsoft web sitesi, "COM: Component Object Model Technologies", http://www.microsoft.com/com/default.mspx, (4.5.2009).

45. İnternet: C# Corner Web sitesi, "Writing an ActiveX Control in .NET", http://www.c-sharpcorner.com/UploadFile/dsandor/ActiveXInNet11102005040748AM/ActiveXInNet.aspx, (12.3.2003).

46. İnternet: CodeProject Web sitesi, "Downloading C# ActiveX Components through CAB File", http://www.codeproject.com/KB/cs/C__Deployment.aspx, (24.7.2007).

EKLER

EK-1. ASN.1 Encode Metodları

EncodeLen, makeTLV ve basaTagEkle metodları aşağıda verilmiştir.

```
BYTE* encodeLen(int len,int &returnLen)
{
	BYTE *r;
	if(len < 0x80)
	{
		r = new BYTE[1];
		r[0] = len;
		returnLen = 1;
	}
	else if (len < 0x00010000)
	{
		if(len < 0x0100)
		{
			r = new BYTE[2];
			r[0] = (BYTE)0x81;
			r[1] = (BYTE)(len & 0x0000ff) ;
			returnLen = 2;
		}
		else {
			r = new BYTE[3];
			r[0] = (BYTE)0x82;
			r[1] = (BYTE)((len & 0x00ff00) >> 8);
			r[2] = (BYTE)(len & 0x0000ff) ;
			returnLen = 3;
		}
	}
	else throw "len cok uzun";
	return r;
}

CRYPT_ATTR_BLOB encodeLen(int len)
{
	CRYPT_ATTR_BLOB r;
	int x;
	BYTE *p = encodeLen(len,x);
	r.cbData = x;
	r.pbData = p;
	return r;
}

void sil(CRYPT_ATTR_BLOB blob)
{
	delete[] blob.pbData;
}
```

EK-1. (Devam) ASN.1 Encode Metodları

```
CRYPT_ATTR_BLOB makeTLV(CRYPT_ATTR_BLOB data1,CRYPT_ATTR_BLOB data2)
{
	CRYPT_ATTR_BLOB r,len;
	len = encodeLen(data1.cbData + data2.cbData);
	r.cbData = 1 + len.cbData + data1.cbData + data2.cbData;
	r.pbData = new BYTE[r.cbData];
	r.pbData[0] = 0x30; //Tag
	memcpy(r.pbData+1,len.pbData,len.cbData); //Length
	memcpy(r.pbData+1+len.cbData,data1.pbData,data1.cbData); //Value
	memcpy(r.pbData+1+len.cbData+data1.cbData,data2.pbData,data2.cbData); //Value
	sil(len);
	sil(data1);
	sil(data2);
	return r;
}

void basaTagEkle (CRYPT_ATTR_BLOB *blob,BYTE tag,bool silinecek = true)
{
	CRYPT_ATTR_BLOB r,len;
	len = encodeLen(blob->cbData);
	r.cbData = 1+len.cbData+blob->cbData;
	r.pbData = new BYTE[r.cbData];
	r.pbData[0] = tag; //Tag
	memcpy(r.pbData+1,len.pbData,len.cbData); //Length
	memcpy(r.pbData+1+len.cbData,blob->pbData,blob->cbData); //Value
	sil(len);
	if(silinecek)
		sil(*blob);
	*blob = r;
}
```

EK-2. İptal Kontrolü Metodları

İptal kontrolünde kullanılan GetSignerCert, RetrieveCRL, GetCertCRLURL, GetWebPage, GetRootCert metodları aşağıda verilmiştir.

```
PCCERT_CONTEXT crlVerify::getSignerCert (PCRYPT_DATA_BLOB pEncodedBlob){
      ////////////////////////

  //--------------------------------------------------------------
  //    Sertifika sahibinin ismini tutmak için buffer.

  char pszNameString[MAX_NAME];

  //--------------------------------------------------------------
  // Bu değişkenler yalnızca decode aşamasında kullanılır.

  HCRYPTMSG hMsg;
  HCERTSTORE hStoreHandle;          // Sertifika deposu İşleyicisi
  DWORD cbData = sizeof(DWORD);
  DWORD cbDecoded;
  BYTE *pbDecoded;
  DWORD cbSignerCertInfo;
  PCERT_INFO pSignerCertInfo;
  PCCERT_CONTEXT pSignerCertContext;

  /*--------------------------------------------------------------
     Aşağıdaki kod parçası mesajı decode eder ve mesajdaki imzayı
                 doğrular.
  --------------------------------------------------------------*/

  //--------------------------------------------------------------
  //  Decode için mesajı aç.

  if(hMsg = CryptMsgOpenToDecode(
     MY_ENCODING,      // encode tipi
     0,                // bayraklar
     0,                // varsayılan mesaj tipini kullan

     NULL,             // kriptografik sağlayıcı

     NULL,             // alıcı bilgisi
     NULL))            // akış (stream) bilgisi
  {
     printf("Çözümlenecek mesaj açıldı. \n");
  }
  else
  {
```

EK-2. (Devam) İptal Kontrolü Metodları

```
    cout<<"Çözümlenecek mesaj açılamadı.";
}
//--------------------------------------------------------------
//  Mesajı encoded Blob ile güncelle

if(CryptMsgUpdate(
   hMsg,                // mesajı ele al
   pEncodedBlob->pbData, // encoded Bloba işaretçi (pointer)
   pEncodedBlob->cbData, // encoded BLOB büyüklüğü
   TRUE))               // son çağrı
{
   printf("Encoded blob mesaja eklendi. \n");
}
else
{
   cout<<"Encoded blob mesaja eklenemedi.";
}

//--------------------------------------------------------------
//  Decode edilen mesajı tutmak için gerekli byte
//  sayısını bul.

if(CryptMsgGetParam(
   hMsg,                // mesajı ele al
   CMSG_CONTENT_PARAM,    // parametre tipi
   0,                   // index
   NULL,
   &cbDecoded))          // dönen bilginin büyüklüğü
{
   printf("Mesaj parametresi (CMSG_CONTENT_PARAM) alındı. \n");
}
else
{
    cout<<"Mesaj parametresi (CMSG_CONTENT_PARAM) alınamadı.";
}
//--------------------------------------------------------------
// Bellek ayır.

if(!(pbDecoded = (BYTE *) malloc(cbDecoded)))
{
    cout<<"Bellek ayırma başarısız.";
}

//--------------------------------------------------------------
// İçeriği kopyala.
```

EK-2. (Devam) İptal Kontrolü Metodları

```
if(CryptMsgGetParam(
   hMsg,               // mesajı ele al
   CMSG_CONTENT_PARAM,   // parametre tipi
   0,                  // index
   pbDecoded,          // dönen bilginin adresi
   &cbDecoded))        // dönen bilginin büyüklüğü
{
   printf("Çözülen mesaj =>\n%s\n\n",
      (LPSTR)pbDecoded);
}
else
{
            cout<<"Mesaj parametresi (CMSG_CONTENT_PARAM # 2) alınamadı";
}

//--------------------------------------------------------------
// Imza doğrulama ;
    // Öncelikle mesajdan imzacı CERT_INFO bilgisini al.

//--------------------------------------------------------------
// Sertifika için gerekli bellek büyüklüğünü al.

if(CryptMsgGetParam(
   hMsg,                    // mesajı ele al
   CMSG_SIGNER_CERT_INFO_PARAM,  // parametre tipi
   0,                       // index
   NULL,
   &cbSignerCertInfo))         // dönen bilginin büyüklüğü

{
            printf("buffer için gerekli byte sayısı : %d .\n",
      cbSignerCertInfo);
}
else
{
    cout<<"SIGNER_CERT_INFO #1 doğrulama başarısız.";
}

//--------------------------------------------------------------
// Bellek ayır.

if(!(pSignerCertInfo = (PCERT_INFO) malloc(cbSignerCertInfo)))
{
   cout<<"Doğrulama belleği ayırma başarısız.";
}
```

EK-2. (Devam) İptal Kontrolü Metodları

```
//-------------------------------------------------------------
// Sertifika bilgisi CERT_INFO) yapısını al.

if(!(CryptMsgGetParam(
   hMsg,                    // mesajı ele al
   CMSG_SIGNER_CERT_INFO_PARAM,  // parametre tipi
   0,                       // index
   pSignerCertInfo,         // dönen bilginin addresi
   &cbSignerCertInfo)))     // dönen bilginin büyüklüğü
{
    cout<<"Doğrulama SIGNER_CERT_INFO #2 başarısız.";
}

//-------------------------------------------------------------
    // CERT_STORE_PROV_MSG kullanarak sertifika deposunu aç.

if(hStoreHandle = CertOpenStore(
   CERT_STORE_PROV_MSG,     // depo sağlayıcısı tipi
   MY_ENCODING,             // encode tipi
   NULL,                    // kriptografik sağlayıcı
                    //varsayılan bayrak için NULL kullan
   0,
   hMsg))                   //
{
   printf("Sertifika deposu açma başarılı.");
}
else
{
    cout<<"Sertifika deposu açma başarısız.";
}

//-------------------------------------------------------------
// Sertifika deposunda imzacı sertifikasını bul.

if(pSignerCertContext = CertGetSubjectCertificateFromStore(
   hStoreHandle,    // depoyu ele al
   MY_ENCODING,     // encode tipi
   pSignerCertInfo))   // alınan CERT_CONTEXT yapısına işaretçi (pointer)
{
   if(CertGetNameString(
      pSignerCertContext,
      CERT_NAME_SIMPLE_DISPLAY_TYPE,
      0,
      NULL,
      pszNameString,
```

EK-2. (Devam) İptal Kontrolü Metodları

```
        MAX_NAME) > 1)
       {
                        printf("Mesajı imzalayan:  %s \n",pszNameString);
       }
       else
       {
           cout<<"Mesajı imzalayan bulunamadı.\n";
       }
   }
   else
   {
       cout<<"Imzacı sertifikası alınamadı.";
   }

  //--------------------------------------------------------------
   // Temizlik

   if(pSignerCertInfo)
   {
      free(pSignerCertInfo);
   }
   if(hMsg)
   {
      CryptMsgClose(hMsg);
   }

       return pSignerCertContext;
}


PCCRL_CONTEXT RetrieveCRL(PCCERT_CONTEXT cert) {
  BYTE         *pbData;
  DWORD         cbData;
  LPSTR        lpszURL, lpszSecondURL;
  PCCRL_CONTEXT pCRL;
  PCCERT_CONTEXT issuer;
  TCHAR * pszStoreName = TEXT("MY");
  HCERTSTORE hCertStore = NULL;

//--------------------------------------------------------------
//   Sertifika deposunu aç
if ( hCertStore = CertOpenSystemStore(
    NULL,
    pszStoreName))
{
```

EK-2. (Devam) İptal Kontrolü Metodları

```
    fprintf(stderr,"Sertifika deposu açıldı.\n");
}
else
{
    printf("Sertifika deposu açılamadı.\n");
    exit(1);
}
//---------------------------------------------------------------

  if (!(lpszURL = GetCertCRLURL( cert, FALSE)))
          return 0;  //sertifikadan sil dağıtım noktasının url'ini al.
        if (pbData =GetWebPage( lpszURL, &cbData )   ) { //sil'i indir.

        issuer=getRootCert(cert);  //sertifikadan kök sertifikayı bul
//kök sertifikanın parmakizi ile ilan edilen parmak izini //karşılaştır.
        lpszSecondURL = GetCertCRLURL( issuer,  TRUE); // cert->issuer //tabloda kontrol et
    if (!lpszSecondURL ) {
      LocalFree(lpszURL);
      return 0;
    }
  }

  if (strcmp(lpszURL,lpszSecondURL)) {
    pCRL = CertCreateCRLContext(X509_ASN_ENCODING, pbData, cbData);
  }
  else
  {
          cout<<"sil url'leri birbirini tutmuyor!";
          return NULL;
  }
  LocalFree(lpszURL);
  return pCRL;
}


LPSTR GetCertCRLURL(PCCERT_CONTEXT cert, BOOL bLookupOnly) {
// bLookupOnly parametresi false ise sil dağıtım noktasını döner
// bLookupOnly parametresi true ise sertifika yayınlayıcı kök // // //sertifikanın
// parmakizini alır ve internette ilan edilen parmakizleri ile karşılaştırır.
  LPSTR lpszURL;
  CACERT *pCACert;

  if (!bLookupOnly) {
    if ( (lpszURL = GetDistributionPoint(cert)) != 0)
      return lpszURL;  }
```

EK-2. (Devam) İptal Kontrolü Metodları

```
 /* Kök sertifikanın parmakizini bul ve tabloda kontrol et */
 if (cert) {
   if (!(pCACert = LookupCACert(cert))) return 0;
   if (pCACert->lpszCRLURL) {
     lpszURL = (LPSTR)LocalAlloc(LMEM_FIXED, lstrlenA(pCACert->lpszCRLURL) + 1);
     if (!lpszURL) return 0;
     lstrcpy(lpszURL, pCACert->lpszCRLURL);
     return lpszURL;
   }
 }

 return 0;
}


BYTE * GetWebPage( char* Url, DWORD *Length )
{
   HINTERNET hHttpFile;
   char szSizeBuffer[32];
   DWORD dwLengthSizeBuffer = sizeof(szSizeBuffer);
   DWORD dwFileSize;
   DWORD dwBytesRead;
   BOOL bSuccessful;
   BYTE * szContents;

       HINTERNET  m_Session;

    m_Session = InternetOpen(TEXT("Süper program"),
            INTERNET_OPEN_TYPE_PRECONFIG, NULL, NULL, 0);
    if (!m_Session) return 0;

  // Opening the Url and getting a Handle for HTTP file
        hHttpFile = InternetOpenUrl(m_Session, ( char *) Url, NULL, 0, 0, 0);

  if (hHttpFile)
  {
     // Getting the size of HTTP Files
                BOOL bQuery =
                HttpQueryInfo(hHttpFile,HTTP_QUERY_CONTENT_LENGTH, szSizeBuffer,
                &dwLengthSizeBuffer, NULL) ;

     if(bQuery==TRUE)
     {
        // Allocating the memory space for HTTP file contents
        dwFileSize=atol(szSizeBuffer);
```

EK-2. (Devam) İptal Kontrolü Metodları

```
        szContents = new BYTE[dwFileSize];
                        *Length=dwFileSize;

        // Read the HTTP file
                BOOL bRead = InternetReadFile( hHttpFile, szContents, dwFileSize,
                &dwBytesRead);

        if (bRead)
           bSuccessful = TRUE;

        InternetCloseHandle(hHttpFile); // Close the connection.
      }

   }
   else
   {
      // Connection failed.
      bSuccessful = FALSE;
   }
   return szContents;
}


PCCERT_CONTEXT crlVerify::getRootCert(PCCERT_CONTEXT child){

HCERTCHAINENGINE        hChainEngine;
CERT_CHAIN_ENGINE_CONFIG ChainConfig;
PCCERT_CHAIN_CONTEXT    pChainContext;
PCCERT_CHAIN_CONTEXT    pDupContext;
HCERTSTORE              hCertStore;
CERT_ENHKEY_USAGE       EnhkeyUsage;
CERT_USAGE_MATCH        CertUsage;
CERT_CHAIN_PARA         ChainPara;
DWORD                   dwFlags=0;
BYTE                   *pszNameString;
char NameString[256];

//--------------------------------------------------------
// Initialize data structures.

if(!(pszNameString=(BYTE *)malloc(256)))
   cout<<"Memory allocation failed.";
EnhkeyUsage.cUsageIdentifier = 0;
EnhkeyUsage.rgpszUsageIdentifier=NULL;
CertUsage.dwType = USAGE_MATCH_TYPE_AND;
```

EK-2. (Devam) İptal Kontrolü Metodları

```
CertUsage.Usage  = EnhkeyUsage;
ChainPara.cbSize = sizeof(CERT_CHAIN_PARA);
ChainPara.RequestedUsage=CertUsage;

ChainConfig.cbSize = sizeof(CERT_CHAIN_ENGINE_CONFIG);
ChainConfig.hRestrictedRoot= NULL ;
ChainConfig.hRestrictedTrust= NULL ;
ChainConfig.hRestrictedOther= NULL ;
ChainConfig.cAdditionalStore=0 ;
ChainConfig.rghAdditionalStore = NULL ;
ChainConfig.dwFlags = CERT_CHAIN_CACHE_END_CERT;
ChainConfig.dwUrlRetrievalTimeout= 0 ;
ChainConfig.MaximumCachedCertificates=0 ;
ChainConfig.CycleDetectionModulus = 0;

//--------------------------------------------------------
// Create the nondefault certificate chain engine.

if(CertCreateCertificateChainEngine(
        &ChainConfig,
        &hChainEngine))
{
   printf("A chain engine has been created.\n");
}
else
{
   cout<<"The engine creation function failed.";
   return NULL;
}
// Open the My system store.

if(hCertStore=CertOpenSystemStore(NULL,"MY"))
{
      printf("The MY Store is open.\n");
}
else
{
      cout<<"The MY system store did not open.";
          return NULL;
}

//-------------------------------------------------------------
// Build a chain using CertGetCertificateChain
// and the certificate retrieved.
```

EK-2. (Devam) İptal Kontrolü Metodları

```
if(CertGetCertificateChain(
   NULL,                // use the default chain engine
   child,          // pointer to the end certificate
   NULL,                // use the default time
   NULL,                // search no additional stores
   &ChainPara,            // use AND logic and enhanced key usage
                        //  as indicated in the ChainPara
                        //  data structure
   dwFlags,
   NULL,                // currently reserved
   &pChainContext))       // return a pointer to the chain created
{
   printf("The chain has been created. \n");
}
else
{
   cout<<"The chain could not be created.";
       return NULL;
}

// kok bulalım
PCERT_SIMPLE_CHAIN pChainIssuer= pChainContext->rgpChain[0];
PCCERT_CONTEXT issuer=  pChainIssuer->rgpElement[pChainContext->rgpChain[0]-
>cElement-1]->pCertContext;

//display(issuer);
return issuer;
}
```

EK-3. DecodeMessage Metodu

```
BYTE * DecodeMessage(PCRYPT_DATA_BLOB pEncodedBlob, int * size, int secenek)
{
   //-------------------------------------------------------------
   //    Sertifika subject için buffer.
   char pszNameString[MAX_NAME];

   //-------------------------------------------------------------
   //  Aşağıdaki değişkenler decode aşamasında kullanılır.

   HCRYPTMSG hMsg;
   HCERTSTORE hStoreHandle;          // certificate store işleyici
   DWORD cbData = sizeof(DWORD);
   DWORD cbDecoded;
   BYTE *pbDecoded;
       BYTE *mesaj;
   DWORD cbSignerCertInfo;
   PCERT_INFO pSignerCertInfo;
   PCCERT_CONTEXT pSignerCertContext;


   //-------------------------------------------------------------
   //  Decode için mesajı aç.

   if(hMsg = CryptMsgOpenToDecode(
      MY_ENCODING,      // encoding tipi
      0,                 // bayraklar
      0,                 // default mesaj tipi seç
                        // mesaj tipi mesaj headerda
                        // belirtilmiştir.
      NULL,              // cryptographic sağlayıcı
                        // default sağlayıcı için NULL kullan
      NULL,              // alıcı bilgisi
      NULL))             // stream bilgisi
   {
      printf("Decode için mesaj açıldı. \n");
   }
   else
   {
      cout<<"OpenToDecode başarısız";
   }
   //-------------------------------------------------------------
   //  Encode edilen BLOB ile mesajı güncelle.

   if(CryptMsgUpdate(
      hMsg,              // mesajı ele al
```

EK-3. (Devam) DecodeMessage Metodu

```
    pEncodedBlob->pbData, // encoded BLOB'a pointer
    pEncodedBlob->cbData, // encoded BLOB büyüklüğü
    TRUE))              // son çağrı
{
    printf("encoded BLOB mesaja eklendi. \n");
}
else
{
    cout<<"Decode MsgUpdate başarısız";
}

//--------------------------------------------------------------
// Buffer'ın decode edilen mesajı tutabilmesi
    //için gerekli byte sayısını bul

if(CryptMsgGetParam(
    hMsg,               // mesajı ele al
    CMSG_CONTENT_PARAM,    // parameter tipi
    0,                  // index
    NULL,
    &cbDecoded))          // dönen bilginin büyüklüğü
{
    printf("Mesaj parametresi alındı. \n");
                if(secenek==1)
                        *size=cbDecoded;
}
else
{
    cout<<"Decode CMSG_CONTENT_PARAM başarısız.";
}
//--------------------------------------------------------------
// bellek ayır.

if(!(pbDecoded = (BYTE *) malloc(cbDecoded)))
{
    cout<<"Decode bellek ayırma başarısız.";
     } else if (secenek==1) {
                mesaj = (BYTE *) malloc(cbDecoded);
     }

//--------------------------------------------------------------
// Içeriği buffer'a kopyala.

if(CryptMsgGetParam(
    hMsg,               // mesajı ele al
```

EK-3. (Devam) DecodeMessage Metodu

```
    CMSG_CONTENT_PARAM,   // parameter tipi
    0,                    // index
    pbDecoded,            // dönen bilginin adresi
    &cbDecoded))          // dönen bilginin büyüklüğü
{
            printf("decode edilen mesaj : =>\n%s\n\n",
      (LPSTR)pbDecoded);
            if(secenek==1){
                    mesaj=pbDecoded;
                    if(hMsg)
                    {
                            CryptMsgClose(hMsg);
                    }
                    return mesaj;
            }
}
else
{
   cout<<"Decode CMSG_CONTENT_PARAM #2 başarısız";
}

//--------------------------------------------------------------
// Imza doğrula.
// Öncelikle imzacı CERT_INFO'yu mesajdan al.

//--------------------------------------------------------------
// Sertifika için gerekli bellek al.

if(CryptMsgGetParam(
   hMsg,                      // mesaj ele al
   CMSG_SIGNER_CERT_INFO_PARAM,  // parameter tipi
   0,                         // index
   NULL,
   &cbSignerCertInfo))        // dönen bilginin büyüklüğü

{
   printf("%d buffer için gerekli byte sayısı.\n",
      cbSignerCertInfo);

}
else
{
   cout<<"Doğrulama SIGNER_CERT_INFO #1 başarısız.";
}
```

EK-3. (Devam) DecodeMessage Metodu

```
//-------------------------------------------------------------
// Bellek ayır.

if(!(pSignerCertInfo = (PCERT_INFO) malloc(cbSignerCertInfo)))
{
   cout<<" Doğrulama bellek ayırımı başarısız.";
}

//-------------------------------------------------------------
// Mesaj sertifika bilgisi (CERT_INFO) yapısını al

if(!(CryptMsgGetParam(
   hMsg,                    // mesajı ele al
   CMSG_SIGNER_CERT_INFO_PARAM,  // parameter tipi
   0,                       // index
   pSignerCertInfo,         // dönen bilginin adresi

   &cbSignerCertInfo)))     // dönen bilginin büyüklüğü
{
   cout<<"Doğrulama SIGNER_CERT_INFO #2 başarısız";
}

//-------------------------------------------------------------
// CERT_STORE_PROV_MSG kullanarak mesajdaki sertifikalarla başlatılan
    // bellek sertifika store aç

if(hStoreHandle = CertOpenStore(
   CERT_STORE_PROV_MSG,     // store sağlayıcı tipi
   MY_ENCODING,             // encoding tipi
   NULL,                    // cryptographic sağlayıcı (default NULL)

   0,                       // bayraklar
   hMsg))                   // mesajı ele al
{
   printf("Mesaj için sertifika store " \
      "doğrulama açıldı.\n");
}
else
{
   cout<<"Doğrulama store açma başarısız";
}

//-------------------------------------------------------------
// Store'da imzacı sertifikasını bul.
```

EK-3. (Devam) DecodeMessage Metodu

```
if(pSignerCertContext = CertGetSubjectCertificateFromStore(
   hStoreHandle,       // store ele al
   MY_ENCODING,   // encoding tipi
   pSignerCertInfo))   // alınan CERT_CONTEXT'e pointer
{
   if(CertGetNameString(
      pSignerCertContext,
      CERT_NAME_SIMPLE_DISPLAY_TYPE,
      0,
      NULL,
      pszNameString,
      MAX_NAME) > 1)
   {
                        printf("Mesajı imzalayan :  %s \n",pszNameString);
                        if(secenek==2){
                                mesaj=(BYTE *)pszNameString;
                                for(int i=0; i<MAX_NAME; i++){
                                if(pszNameString[i]=='\0'){
                                        break;
                                }
                                *size=i+1;
                        }

                        }
   }
   else
   {
      cout<<"Imzacı bulunamadı.\n";
   }
}
else
{
   cout<<"Verify GetSubjectCert başarısız";
}

//--------------------------------------------------------------
// Imzacı sertifikasından alınan CERT_INFO ile imza doğrulanır

if(CryptMsgControl(
   hMsg,
   0,
   CMSG_CTRL_VERIFY_SIGNATURE,
   pSignerCertContext->pCertInfo))
{
   printf("Imza doğrulama başarılı. \n");
```

EK-3. (Devam) DecodeMessage Metodu

```
    }
    else
    {
        printf("Imza doğrulanamadı. \n");
    }

    if(pSignerCertInfo)
    {
        free(pSignerCertInfo);
    }
    if(pSignerCertContext)
    {
        CertFreeCertificateContext(pSignerCertContext);
    }
    if(hStoreHandle)
    {
        CertCloseStore(hStoreHandle, CERT_CLOSE_STORE_FORCE_FLAG);
    }
    if(hMsg)
    {
        CryptMsgClose(hMsg);
    }
        return mesaj;
}
```

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı	: GÖLLE, Hasan
Uyrugu	: T.C.
Dogum tarihi ve yeri	: 01.04.1971 Ereğli / Konya
Medeni hali	: Evli
Telefon	: 0 (312) 4688486 – 1137
Faks	: 0 (312) 4688466
e-posta	: hasan.golle@tubitak.gov.tr

Eğitim

Derece	Eğitim Birimi	Mezuniyet tarihi
Lisans	Hacettepe Üniv. / Bilgisayar Mühendisligi	2006
Lisans	ODTÜ / Elektrik Elektronik Mühendisligi	1994
Lise	Ereğli Lisesi	1989

İş Deneyimi

Yıl	Yer	Görev
2006-	TÜBİTAK-UEKAE / ANKARA	Uzman Araştırmacı
2000-2006	Türksat-Eurasiasat / ANKARA	Uydu Operasyonları Mühendisi
1998-2000	Türksat-Eurasiasat / ANKARA	Uydu Payload Mühendisi
1996-1998	Türksat-Eurasiasat / ANKARA	RF & Baseband Mühendisi

Yabancı Dil

İngilizce

Yayınlar

Gölle, H., Sağıroğlu, Ş. 2008, Developing ETSI compatible electronic signature module by using CMS based library, ISCTurkey International Conference, Ankara

Hobiler

Futbol, Seyahat, Sinema