TÜBİTAK BİLGEM

MA3 API

İçindekiler

1. AÇIK ANAHTAR ALTYAPISI (AAA)

1.1 Giriş

Açık anahtar altyapısı sistemleri bize kimlik doğrulama, inkar edememezlik, mesaj bütünlüğü ve gizlilik gibi hizmetleri simetrik kriptografinin kullanıldığı bir yöntemle sunarlar. Böylece anahtar dağıtımı ve elektronik imza özelliklerini asimetrik kriptografinin, gizlilik hizmetini simetrik kriptografinin sağladığı güvenli bir altyapıya kavuşabiliriz.

Açık anahtar altyapısına aşağıdaki iki sebepten dolayı ihtiyaç duyulur:

- Asimetrik kriptografi sistemlerini gerçeklemek için
- Açık ve özel anahtar yönetmek için

Bu bölümde açık anahtar altyapısına ait temel bilgiler ele alınacaktır. Açık anahtar altyapısının en önemli ürünü olan sertifikalardan bahsedilecek ve Basit Sertifikalar, Açık Anahtar Sertifikaları gibi konu başlıklarında sertifikanın tarifi verilecektir. Sertifika İptal Listeleri, Sertifikasyon Prensipleri ve Sertifikasyon Yolu gibi kavramlarda bu bölümde sırasıyla işlenecektir.

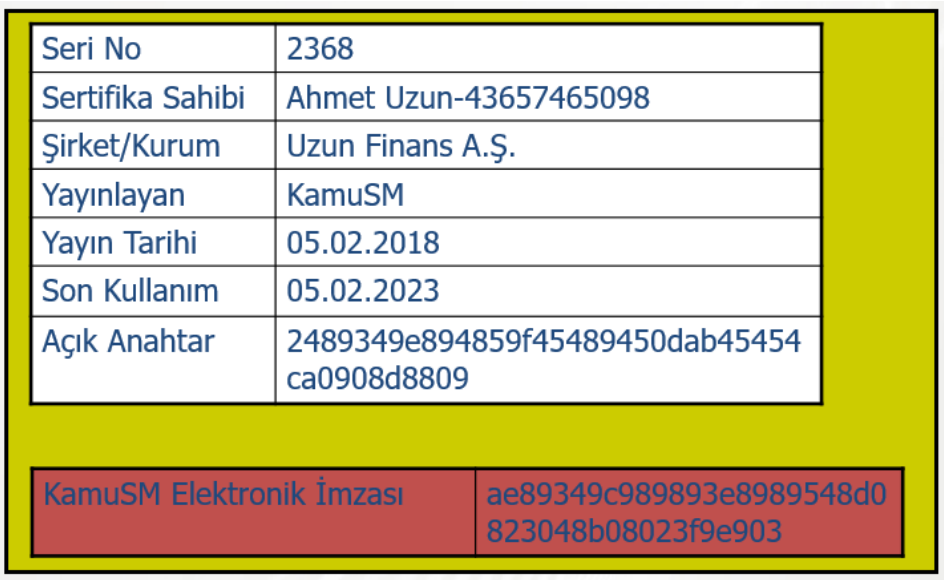
1.2 Sertifikalar

Asimetrik kriptografide bir kişi için üretilen anahtar çifti, özel ve açık anahtardan oluşur. Bu anahtarlardan açık olanı anahtarın sahibiyle haberleşmek isteyen herkes tarafından görülebilir ve kullanılabilir. Bu açık anahtarın isteyen kişilerce kullanımını kolaylaştırmak için değişik şekillerde yayınlanması ve isteyenlerin erişimine açılması mümkündür. Bu yayınlama şekline sertifika adını verilmektedir.

Açık anahtarı yayınlamak için kullanılacak ideal bir sertifika aşağıdaki özellikleri taşımalıdır:

- Elektronik ortamda (örneğin: internette) yayınlanabilmesi ve otomatik olarak işlenebilmesi için tamamen sayısal olmalıdır.
- Özel anahtarın sahibinin adını, çalıştığı şirketin/kurumun adını ve irtibat kurmak için gerekli bilgileri içermelidir.
- Sertifikanın ne zaman yayınlandığını anlamak kolay olmalıdır.
- Özel anahtarın sahibi tarafından değil güvenilir bir 3. kurum tarafından yaratılmalıdır.
- Güvenilen kurum birçok sertifika yaratacağı için (aynı kullanıcı için bile birden fazla) her bir sertifikanın diğerinden kolayca ayırt edilebilmesi gereklidir.
- Bir sertifikanın gerçek veya sahte olduğu kolayca tespit edilebilmelidir.
- Değiştirmeye karşı korunmuş olmalıdır.
- İçindeki bilgilerin güncel olup olmadığı istendiği anda tespit edilebilmelidir.
- Hangi uygulamalar için kullanılabileceği sertifikanın içinde yazmalıdır.

İdeal sertifika tanımına uygun bir elektronik sertifika oluşturabilmek için uluslararası ITU kurumu X.509 standardını tanımlamıştır. Bu özellikleri taşıyan örnek bir sertifika aşağıdaki gibi olabilir:



Seri No	2368
Sertifika Sahibi	Ahmet Uzun-43657465098
Şirket/Kurum	Uzun Finans A.Ş.
Yayınlayan	KamuSM
Yayın Tarihi	05.02.2018
Son Kullanım	05.02.2023
Açık Anahtar	2489349e894859f45489450dab45454 ca0908d8809

KamuSM Elektronik İmzası	ae89349c989893e8989548d0 823048b08023f9e903

Şekil 1: Örnek Sertifika

1.2.1 Nitelikli Sertifika

Nitelikli Sertifika (Qualified Certificate), X.509 Sertifikası baz alınarak hazırlanan ve sadece gerçek kişilere verilen bir sertifika çeşididir. Bu sertifika tipi RFC 3739'da tanımlanmıştır. Nitelikli sertifikalar Türkiye'de ve bir çok Avrupa ülkesinde elle atılan ıslak imzaya eşdeğer elektronik imzalar atmak için kullanılır. Bu sertifikaları standart X.509 sertifikasından farklı kılan **en önemli** özellik üretiminde ve sahibine verilmesinde çok sıkı kimlik doğrulama kuralları uygulanması ve sertifika merkezlerinin işletiminin denetlenmesi olarak sayılabilir.

Nitelikli sertifika alanları ile ilgili önemli bilgiler aşağıda verilmiştir.

Yayınlayıcı adı(Issuer Name), aşağıdakilerin bir alt kümesidir:
domainComponent, countryName, stateOrProvinceName, organizationName, localityName, serialNumber.

Kullanıcı adı(Subject Name), aşağıdakilerin bir alt kümesidir:
countryName, commonName, surname, givenName, pseudonym, serialNumber, organizationName, organizationalUnitName, stateOrProvinceName ,localityName, postalAddress**.**

1.2.2 Sertifika Bütünlüğünün Korunması

X.509 Sertifikasının bütünlüğünün korunması için sertifika içinde yer alan tüm bilgiler sertifikayı veren makam tarafından elektronik olarak imzalanır ve oluşan elektronik imza bu sertifikanın arkasına eklenir. Böylece sertifika alanlarından herhangi birisi üzerinde sonradan değişiklik yapılıp yapılmadığı kontrol edilebilir. Bir elektronik sertifikanın üstündeki elektronik imza doğrulanarak sertifikanın geçerliliği kontrol edilir.

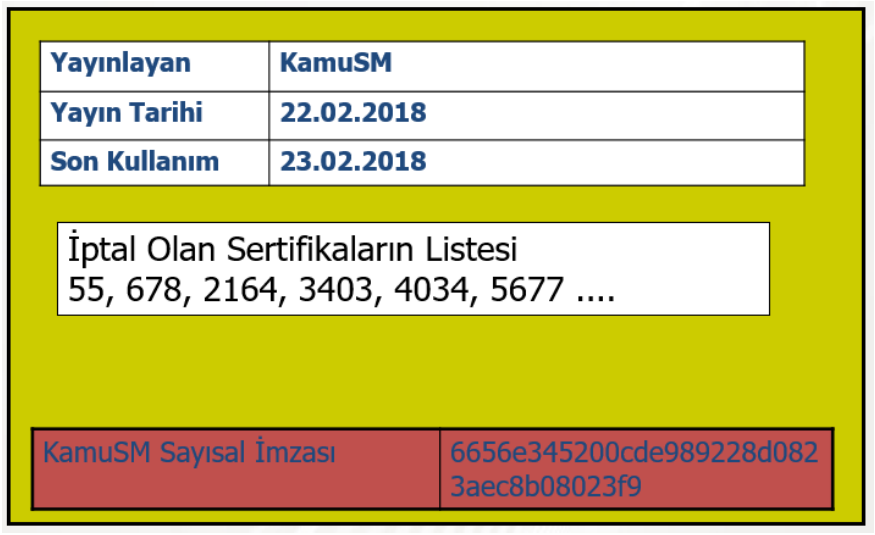
1.3 Sertifika İptal Listesi (SİL)

Sertifika sahibi sertifikasını kullanmak isteyen kişilere dağıttıktan sonra geri toplayamaz. Ayrıca sertifika sahibinin kendisi ile ilgili değişiklikleri duyurması çok zordur. Bu nedenle sertifika iptal listeleri (SİL) kullanılır.

Sertifika iptal listeleri aşağıdaki özellikleri taşır:

- Sayısaldır
- Artık güvenilemeyecek olan ve kullanım süresi dolmamış sertifikaların seri numaralarını içerir.
- Yayın tarihini ve son kullanım tarihini içerir.
- Yayınlayan kuruluşun adını ve sayısal imzasını içerir.
- Sık aralıklarla elektronik ortamda (örneğin internette) yayınlanır.

Örnek bir sertifika iptal listesi aşağıdaki gibi olabilir:



Şekil 2: Örnek SİL

AAA sisteminde, sertifika tabanlı işlem yapan (kimlik doğrulama, elektronik imza doğrulama vb.) her türlü yazılım ve donanım kullanacağı sertifikaların geçerliliğini kontrol ederken güncel bir sertifika iptal listesine bakmak zorundadır.

Eğer işlemde kullanılacak bir sertifikanın seri numarası SİL içinde bulunursa o sertifika geçersiz kabul edilir. Açık anahtar sertifikasının içeriğinin güncel olup olmadığı sorusuna cevap vermek gerekmektedir. Çünkü;

- Sertifika sahibinin erişim bilgileri değişmiş olabilir.
- Sertifika sahibi özel anahtarını kaybettiği için yeni bir açık anahtar kullanmaya başlamış olabilir.

1.4 Sertifika Makamı

Sertifika makamı(SM), açık anahtar altyapısında yer alan sertifikaları ve sertifika iptal listelerini üretmekle görevli olan merkezi birime verilen addır. Sertifika makamı, donanım ve yazılım parçalarından ve sistemi işleten kişiler ve kurallardan oluşan bir yapıdır. Bir sertifika makamını diğer sertifika makamlarından ayırt edici özellikleri; adı ve anahtar çiftidir(açık ve özel anahtardan oluşan).

Sertifika makamının görevleri şunlardır:

- Sertifika yayınlamak
- Sertifika durum bilgilerini güncel tutmak ve sertifika iptal listeleri (SİL) hazırlamak
- Güncel sertifikaları ve SİL'leri isteyen kişilere sunmak
- Süresi dolan ya da iptal edilen sertifikaların arşivini tutmak

Açık anahtar altyapısının ürettiği sertifikaları ve anahtarları kullanan kişiler bu yapının önemli bir parçasını oluştururlar. Sertifika kullanıcıları, açık anahtar altyapısını kullanırken sertifika makamına sertifika talebinde bulunurlar.

Sertifika kullanıcıları aldıkları sertifika ile bir özel ve bir açık anahtar sahibi olurlar ve elektronik imza, simetrik anahtar değiş tokuşu, kimlik doğrulama gibi işlemleri yaparlar.

Sertifika kullanıcıları;

- Sertifikalarını yayınlayan SM'yi güvenilen taraf olarak kabul ederler.
- Sertifikaları kullanarak karşı tarafın sayısal imzasını ve kimliğini doğrulayabilirler.
- İletişim kurmak istedikleri kişinin sertifikasını ve SM'nin yayınladığı SİL'lere ulaşabilirler.

1.5 Çevrimiçi Sertifika Durum Protokolü (ÇİSDUP)

SİL'lerin bir periyot boyunca geçerli oması, örneğin 24 saatte bir yenilenmesi, bu periyot boyunca iptal edilen sertifikalardan haberdar olmayı geciktirmektedir. Kullanıcılar iptal olan bir sertifikayı bir sonraki periyotta yayınlanan SİL içinde görebilmektedirler. Finansal işlemler gibi anlık doğrulamaya ihtiyaç duyulan uygulamalarda SİL yöntemi yeterince güvenlik sağlamamaktadır. Bu nedenle çözüm olarak olarak Çevrimiçi Sertifika Durum Protokolü(ÇİSDUP) kullanılmaktadır (OCSP - Online Certificate Status Protocol).

Bu protokol şu şekilde çalışır:

Kullanıcı isteği:
10 numaralı sertifikanın durumu nedir?

ÇİSDUP sunucusu yanıtı(Aşağıdakilerden biri olabilir):
Bu sertifikanın durumu;

- İyi (İptal edilmemiş)
- Kötü (İptal edilmiş)
    - İptal nedeni
    - İptal zamanı
- Bilinmiyor

Her SM'ye ait bir veya birden fazla ÇİSDUP sunucusu olabilir. ÇİSDUP sunucusu bağlı olduğu SM'nin yayınladığı sertifikaların iptal edilip edilmediği bilgisine ulaşır ve kendisine gelen kullanıcı isteklerini cevaplar. ÇİSDUP cevap mesajları elektronik imza ile imzalanarak güvenlik sağlanır.

1.6 Akıllı kartlar

X.509 Sertifikalarını ve bunlara bağlı olan anahtarları taşımak için kullanılan en yaygın ve güvenli cihazlar akıllı kartlardır. Akıllı kartlar, programlanabilir alanları olan, dayanıklı, taşınabilir bilgisayarlardır. Akıllı kartlar veri güvenliği, kimlik gizliliği ve mobil kullanıcı ihtiyaçlarına sahip sistemlerde faydalıdır. Bu kartların kriptolojik işlemci taşıyan modelleri aşağıdaki özellikleri taşır:

- Kart üzerinde şifreleme ve şifre çözme
- Kart üzerinde imzalama ve imza onaylama
- Kart üzerinde özel ve açık anahtarların tutulması
- Kart içine bilgi yazabilme
- Kartın şifre ile korunması

Şekil 3: Sim kart boyutunda bir akıllı kart

Akıllı kartların özel ve genel alanları vardır. Özel alanında anahtar üretimi, imzalama, şifre çözme gibi işlemler yapılır, bu alana erişim yasaklanamıştır.Genel alana genel bilgiler yazılır. Akıllı kart program yardımıyla buradaki bilgiler görülebilir.

Bir AAA sistemi tarafından kullanılan akıllı kartta olması gerekenler şu şekilde sıralanabilir:

- İmzalama özel anahtarı
- Şifreleme özel anahtarı
- O an geçerli olan imzalama sertifikası
- O an geçerli olan şifreleme sertifikası
- Daha önce geçerli olan şifreleme özel anahtarları ve karşılığı olan sertifikalar

2. ELEKTRONİK İMZA

5070 sayılı Elektronik İmza Yasası´nda güvenli elektronik imza, "*münhasıran imza sahibine bağlı olan, sadece imza sahibinin tasarrufunda bulunan güvenli elektronik imza oluşturma aracı ile oluşturulan, nitelikli elektronik sertifikaya dayanarak imza sahibinin kimliğinin tespitini sağlayan, imzalanmış elektronik veride sonradan herhangi bir değişiklik yapılıp yapılmadığının tespitini sağlayan imzadır*" şeklinde tanımlanmaktadır.

Elektronik imza kimlik doğrulama ve onaylama, veri bütünlüğü ve inkar edilememezlik olmak üzere üç temel özelliği sağlamaktadır;

- **Kimlik doğrulama ve onaylama,** imzalanan dökümanın mesaj sahibine ait olduğunu ve geçerliliğini sağlar.
- **Veri bütünlüğü**,imzalanan verinin değiştirilmesini, silinmesini veya veriye ekleme-çıkarma yapılmasını önler.
- **İnkar edilemezlik**,kişi veya kurumların elektronik ortamda gerçekleştirdikleri işlemleri inkar etmelerini önler.

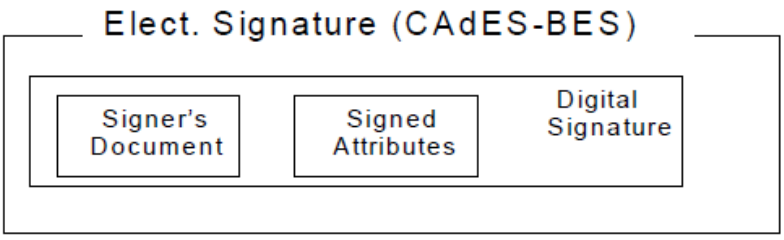
Elektronik imza oluşturmak için sertifika ve özel anahtara sahip olmak gerekmektedir. Bu özel anahtar yukarıda bahsedilen güvenlik elektronik imza oluşturma aracında genel olarak akıllı kart içinde saklanır.

2.1 İmza Tipleri

Bu bölümde, basitten karmaşığa doğru yapılarına göre e-imza tipleri açıklanmaktadır. İmza tiplerinin detayı için ETSI TS 101733 dokümanına bakılabilir. Açıklamalarda ETSI TS 101733 dokümanında yer alan şekillerden faydalanılmıştır.

2.1.1 CAdES-BES

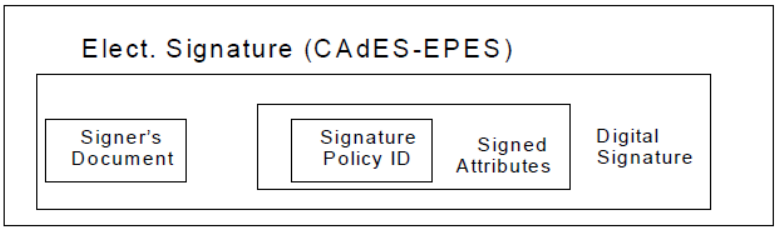
BES imza tipi en basit şekli ile oluşturulan elektronik imzadır. Şekil 4'de de görüldüğü gibi imza dosyası içeriğinde imzalanan belge ve imzaya eklenen diğer imza özellikleri(signed attributes) ile birlikte imzanın kendisi tutulur. BES imzada imza zamanı ile ilgili herhangi bir bilgi yoktur. Bu yüzden uzun süre saklanması gereken e-imzalı belgeler için güvenli kabul edilen bir yöntem değildir. İmzada kullanılan sertifikanın kalan geçerlilik süresinden daha uzun süre saklanması gereken belgeler BES imza ile imzalanmamalıdır. Aksi durumda imzada kullanılan sertifikanın süresi dolduktan sonra geçmişte oluşturulan imzalar güvenilir bir şekilde doğrulanamayacaktır.



Şekil 4: CAdES-BES İmza Yapısı

2.1.2 CAdES-EPES

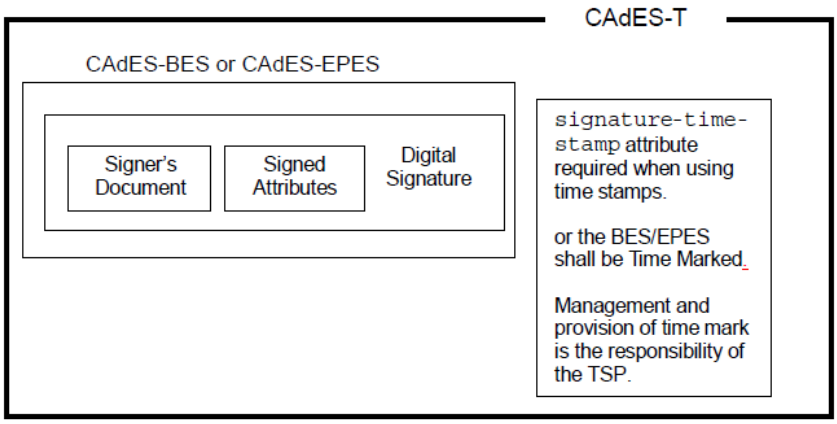
BES imza tipine çok benzer yapıdadır. BES'den tek farkı Şekil 5'de de görüldüğü gibi imza dosyası içeriğine imza politikasını belirten bir imza özelliği(signed attributes) eklenmesidir. EPES imza tipinin kullanılabilmesi için imzanın daha önceden tanımlanmış bir politikaya uygun bir biçimde oluşturulması gerekmektedir. İmzanın belirtilen politikaya uygunluğu imza dosyası içeriğine konulan imza politika numarası(Signature Policy ID) ile belirlenir.



Şekil 5: CAdES-EPES İmza Yapısı

2.1.3 CAdES-T (Zaman Damgası Eklenmiş Elektronik İmza)

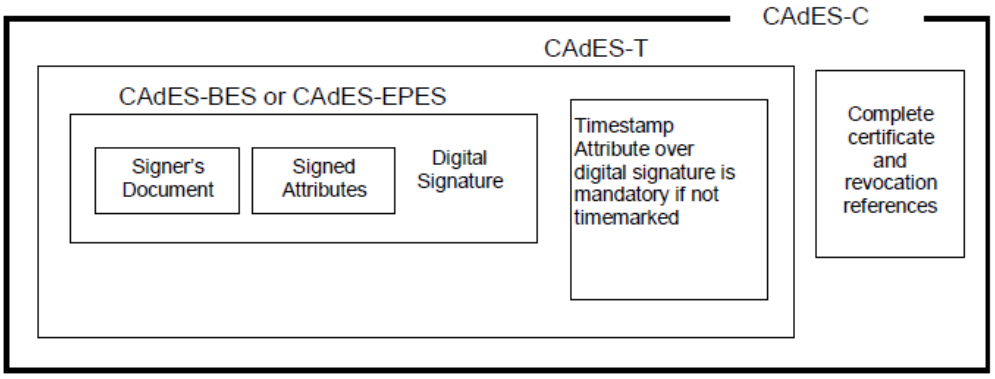
BES veya EPES üzerine kurulan bir imza tipidir. Şekil 6'da de görüldüğü gibi BES veya EPES imzaya zaman damgası alınarak oluşturulur. Zaman damgası imzanın oluşturulduğu tarihi belirlemek için kullanılır. Zaman damgasının Türkiye'de yetkilendirilmiş bir ESHS'den alınması zorunludur. İmza doğrulanırken zaman damgası üzerinde bulunan zaman bilgisi referans alınarak doğrulama işlemi gerçekleştirilir. İmzanın zaman damgası üzerindeki zaman bilgisinden önceki bir tarihte oluşturulmuş olduğu kesin bir şekilde tespit edilebilir. Uzun dönem saklanması gereken e-imzalı belgelerin mutlaka CAdES-T imza tipinde oluşturulması gerekmektedir. İmzalama işlemi yapıldıktan hemen sonra imzaya zaman damgası alınması önerilmektedir.



Şekil 6: CAdES-T İmza Yapısı

2.1.4 CAdES-C (Tüm Doğrulama Verilerinin Referanslarının Eklendiği İmza)

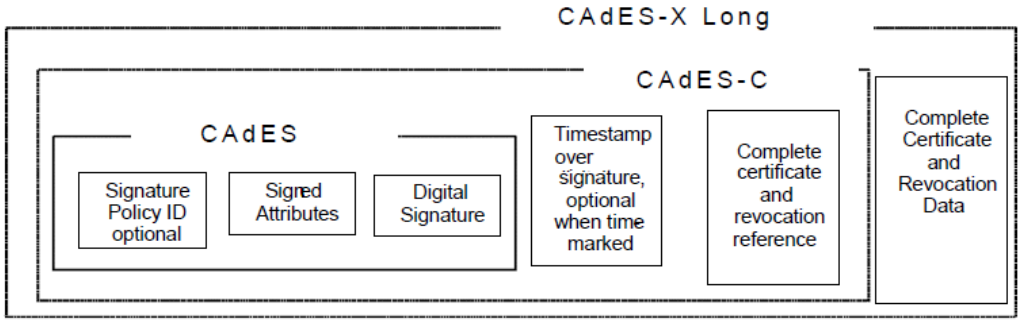
Şekil 7'de görüldüğü gibi ES-T imzası üzerine kurulan bir imza tipidir. İmza doğrulamada kullanılan ESHS'ye ait kök ve alt kök sertifikaları ile sertifikaların iptal kontrollerinin yapıldığı SİL (Sertifika İptal Listesi) veya OSCP cevaplarının referans değerleri imza dosyasına eklenir. Sertifika ve iptal bilgileri imza dosyasına eklenmez, sadece bu bilgilere erişim için tanımlanmış eşsiz referans değerleri imza dosyasına eklenir. İmza doğrulama yapılırken eklenen referans değerleri ile ilgili sertifikalar, SİL ve OCSP cevaplarının dışarıdaki bir sistemden temin edilmesi ve imza doğrulamanın bu veriler kullanılarak yapılması gerekir. İmza dosyasına eklenen referans değerleri ile ilgili sertifikalar, SİL ve OCSP cevapları imza doğrulama sistemi içinde ortak bir veri tabanı veya dizin yapısı içinde tutulabilir. ES-C imza tipinin kullanılması önerilmemekle beraber kullanımının uygun olduğu durumlar da olabilir.



Şekil 7: CAdES-C İmza Yapısı

2.1.5 CAdES-X-LONG (Genişletilmiş Uzun Elektronik İmza)

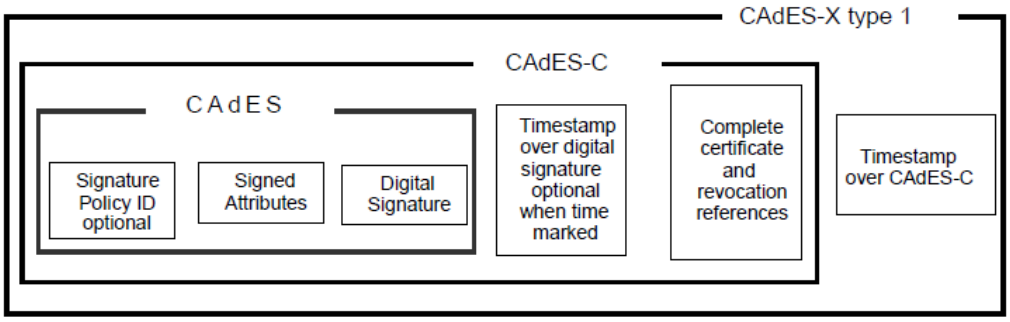
Şekil 8'de görüldüğü gibi ES-C imzası üzerine kurulan bir imza tipidir. İmza doğrulamada kullanılan ESHS'ye ait kök ve alt kök sertifikaları ile sertifikaların iptal kontrollerinin yapıldığı SİL veya OSCP cevapları imza dosyasına eklenir. İmza doğrulama yapılırken imza dosyasına eklenmiş bu veriler kullanılır. İmza doğrulama için dışarıdaki herhangi bir sisteme bağlanıp doğrulama verilerinin edinilmesine gerek kalmaz; doğrulama için gereken tüm veriler imza dosyasının içeriğinden edinilir. İmza doğrulaması yapacak tarafın doğrulama verilerini bulmasını gerektirmediğinden kullanılması en çok tavsiye edilen imza formatıdır. Özellikle oluşturulan imzalı belgelerin kurum dışına gönderileceği durumlarda kullanılması önerilmektedir. İmzalama işlemi yapıldığı anda, imza dosyası ES-X-LONG tipinde oluşturulup saklanabilir.



Şekil 8: CAdES-X-LONG İmza Yapısı

2.1.6 CAdES-X-Type 1 (Genişletilmiş Elektronik İmza Tip 1 Zamanlı)

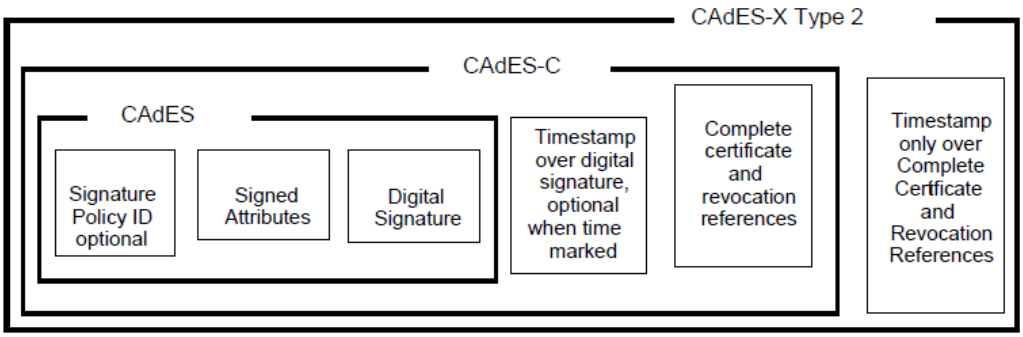
Şekil 9'da görüldüğü gibi ES-C imzası üzerine kurulan bir imza tipidir. ES-C imzasının tamamına zaman damgası alınması ve alınan zaman damgasının imza dosyasına eklenmesi ile oluşturulur. Zaman damgası imza doğrulamada kullanılan sertifika ve iptal verileri referans değerlerinin ve diğer verilerin koruma altına alınması, imza dosyasına hangi tarihten önce eklendiğinin belirlenmesi amacıyla kullanılır. İmza dosyasına fazladan bir zaman damgası alınmasını gerektirdiğinden çok tercih edilen ve kullanılan bir imza tipi değildir. Ancak kullanımının uygun olduğu durumlar da olabilir.



Şekil 9: CAdES-X-Type 1 İmza Yapısı

2.1.7 CAdES-X-Type 2 (Genişletilmiş Elektronik İmza Tip 2 Zamanlı)

CAdES-X-Type 1 ile çok benzer yapıdadır. Aralarındaki tek fark zaman damgasının, ES-C'nin tamamına değil sadece içeriğindeki referans değerlerine alınmış olmasıdır.

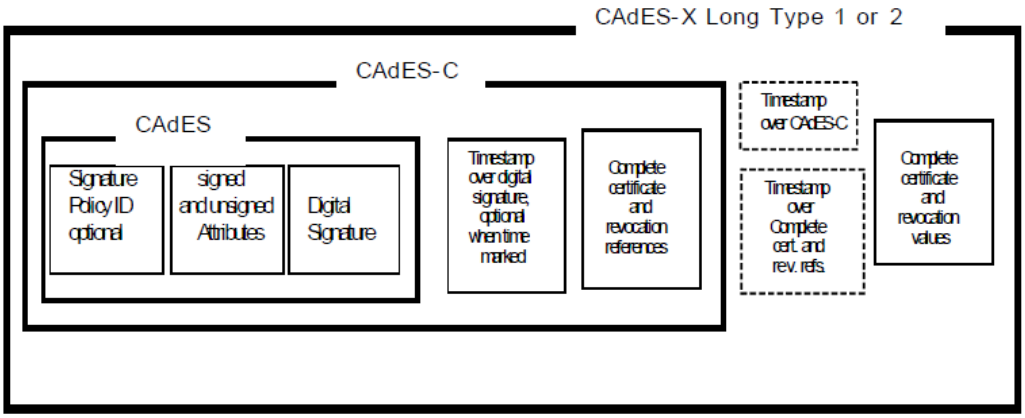


Şekil 10: CAdES-X-Type 2 İmza Yapısı

2.1.8 CAdES-X-Long-Type 1 or Type 2 (Genişletilmiş Uzun Elektronik İmza Tip 1 ve Tip 2 Zamanlı)

CAdES-X-Type 1 ve Type 2 ile çok benzer yapıdadır. Aralarındaki tek fark CAdES-X-Long-Tip 1 veya Tip 2'de imza doğrulamada kullanılan sertifikalar, SİL ve OCSP cevaplarının da imza dosyasına eklenmesidir.

Doğrulama verilerinin tamamını içerdiğinden CAdES-X-Long ile de benzerlik göstermektedir. Ancak CAdES-X-Long'dan farklı olarak fazladan bir zaman damgası alınmasını gerektirmektedir. Kullanımı çok fazla önerilmemektedir. Kullanımının uygun olduğu durumlar oluşabilir ancak genel olarak CAdES-X-Long tipinin kullanılması daha uygundur.



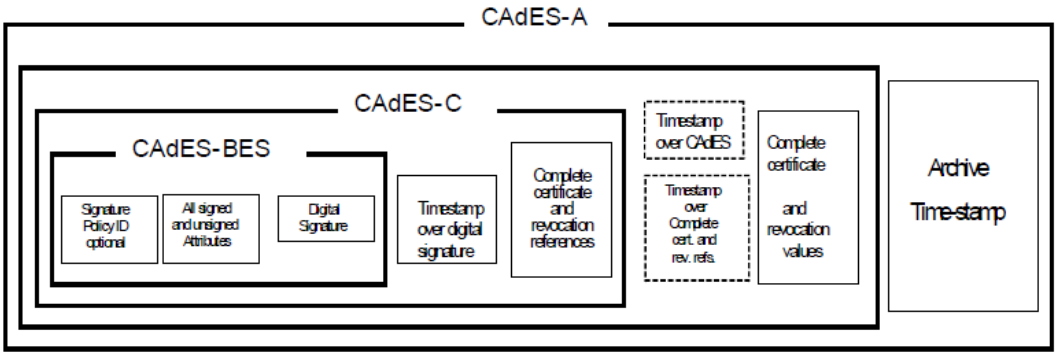
Şekil 11: CAdES-X-Long-Type 1 or Type 2 İmza Yapısı

2.1.9 CAdES-A (Arşiv Elektronik İmza)

E-imzalı belgelerin ESHS'ye ait kök/alt kök ve zaman damgası sertifikalarının geçerlilik süresinden daha uzun bir süre saklanması gerektiği durumlarda kullanılması gereken bir imza formatıdır. ES-A imza formatı tüm doğrulama verileri eklenmiş imza formatları üzerine oluşturulur. ES-A tipinde imza dosyası aşağıdaki imza dosyalarından birisinin üzerine arşiv zaman damgası alınması yoluyla yapılır.

- CAdES-X-Long
- CAdES-X-Long-Type 1
- CAdES-X-Long-Type 2

Arşivleme ESHS'ye ait sertifikaların geçerlilik süresinin sonuna yaklaşılması, sertifikaların iptal olması veya kullanılan algoritmaların kırıldığının duyurulması durumlarında yapılır. Arşivlemenin yukarıdaki durumlar oluşmadan önce yapılmasında bir sakınca yoktur. Arşivleme, yukarıda belirtilen imza dosyalarına alınan arşiv zaman damgası sertifikasının geçerliğinin sona ermesi üzerine gerektikçe tekrarlanmalıdır. Uygulamada bununla ilgili altyapı sağlanmış olmalıdır.



Şekil 12: CAdES-A İmza Yapısı

2.2 İmza Profilleri

Bu bölümde, Bilgi Teknolojileri ve İletişim Kurumu(BTK) tarafından yayınlanmış, Türkiye'de geçerli imza tipleri ve imza içerisinde yer alması gereken özellikler ifade edilmektedir.

E-imza yaratma ve doğrulama süreçleri oluşturulurken veya bir uygulamaya e-imza kabiliyeti eklenirkenkullanılacak e-imzanın kullanım ömrü belirlenmelidir. İmzanın kullanım ömrü, ileride tekrar doğrulanabilme ihtiyacına göre farklı imza profillerinde imza atılabilir.

Profil	İmza Ömrü	Zaman Damgası	İptal Bilgisi [1]	Kesinleşme Süresi	İmza Formatı	İmza Dosya Boyutu
P1	Anlık	Yok	SİL/ ÇiSDuP	Uygulanmaz	BES	Düşük
P2	Kısa	Var	SİL	Uygulanır	ES-T	Orta
P3	Uzun Süreli	Var	SİL	Uygulanır	ES-XL	Çok yüksek
P4		Var	ÇiSDuP	Uygulanmaz	ES-XL	Yüksek

2.2.1 P1: Anlık - İmza Profili

Kullanım ömrü bir sonraki iptal bilgisinin yayınlanmasından daha kısa süren uygulamalarda kullanılır. Örneğin 4 saatte bir SİL yayınlanan senaryoda anlık imzaların ömrü bir kaç dakikadan başlayıp 4 saate kadar uzayabilir. Güvenlik ihtiyacı düşük seviyede olan uygulamalarda kullanılır. İmza doğrulayıcının eline geçtiği an, imza zamanı sayılır. Gelecekte imzayı tekrar doğrulama ihtiyacı olmayacak senaryolarda tercih edilmelidir.Bu imza profili tanımlanmış bir imza politikasına referans vermemektedir.

2.2.2 P2: Kısa Süreli - İmza Profili

İmza ömrünün imza sertifikasının kalan ömründen kısa olduğu uygulamalarda tercih edilmelidir. Kısa ömürlü imzalar sertifika ömrü kadar ömre sahip olabilirler; örneğin 3 yıla kadar. Kısa süreli kullanım ömrü olan imzalarda, ÇiSDuP erişimi bulunmayan ortamlarda tercih edilmelidir. ÇisDuP erişimi olan ortamlarda P4 profili kullanılmalıdır.

2.2.3 P3: Uzun Süreli - İmza Profili

İlk iki kategoriye girmeyen, imza sertifikasının ve imza sertifikasını imzalayan ESHS sertifikasının geçerlilik süresi dolduktan sonra da doğrulanabilecek uygulamalarda kullanılmalıdır.ÇiSDuP erişimi olmayan ortamlarda kullanılabilir. Aksi durumda P3 profilinin, P4 profiline tercih edilebilecek herhangi bir avantajı yoktur, SİL boyutundan ve imzadan sonra SİL yayınlanmasını bekme gerekliliği gibi sebeplerle mecbur kalmadıkça tercih edilmemelidir.

2.2.4 P4: Uzun Süreli- İmza Profili

İlk iki kategoriye girmeyen, imza sertifikasının ve imza sertifikasını imzalayan ESHS sertifikasının geçerlilik süresi dolduktan sonra da doğrulanabilecek imza tipidir.En güvenilir, uzun ömürlü ve sorunsuz imza profilidir. Validasyon verisi imza içinde yer alır.

3. SERTİFİKA DOĞRULAMA

3.1 Giriş

Bu dökümanda, MA3 Sertifika Doğrulama API'sinin temel çalışma şekli ve kullanımı anlatılmaktadır. Konunun daha iyi anlaşılabilmesi amacıyla sertifika ve sertifika doğrulama konuları kısaca özetlenmektedir.

3.2 Gerekler

MA3 API Sertifika Doğrulama Kütüphanesinin kullanımı için sertifika deposu dosyasına, sertifika doğrulamanın nasıl yapılacağını tanımlayan politika dosyasına ve MA3 API sertifika doğrulama kütüphanelerine ihtiyacınız vardır.

İndirdiğiniz paketle birlikte kullanmanız gereken sertifika deposu dosyası ve örnek bir politika dosyası paketin içinde gelecektir. Örnek politika dosyası OCSP'ye öncelik vererek sertifika doğrulamayı gerçekleştirmektedir. Eğer özel olarak yapmak istediğiniz bir kontrol yoksa verilen örnek politika dosyasını kullanabilirsiniz.

MA3 API Sertifika Doğrulama kütüphanesinin kullanımı için SERTİFİKA DOĞRULAMA KÜTÜPHANESİ KULLANIMI bölümünden faydalanabilirsiniz. Sertifika deposunun nasıl kullanılacağı için Yerel Sertifika Deposu bölümüne ve politika dosyası için Sertifika Doğrulama Politikası bölümüne bakabilirsiniz.

3.3 Sertifika Doğrulama

3.3.1 Sertifikalar

Sertifikalar kısaca bir PKI anahtar çifti ile herhangi bir bilgiyi(kimlik bilgisi , yetki, rol vb.), kriptografik olarak ispatlanabilir ve inkar edilemez bir şekilde birbirine bağlayan veri yapılarıdır. Bu anahtar çifti ile veri imzalama, şifreleme, kimlik doğrulama gibi temel PKI işlemleri gerçekleştirilebilmektedir. Bir sertifikanın güvenilirliği seritifikanın güvenilir bir makam(yayıncı kuruluş) tarafından taklit edilemez bir şekilde imzalanmış olmasından kaynaklanmaktadır. Bu imza sertifikanın üzerinde yer almaktadır ve açık anahtar altyapısının doğası gereği istenildiğinde rahatlıkla doğrulanabilmektedir. Yayıncı kuruluşlar doğrudan güvenin kaynağı olabileceği gibi kendileri de bir başka güvenilir kaynaktan sertifika yayınlama sertifikası almış olabilirler. Birinci durumdaki kuruluşlara **Kök Sertifika Makamı(KSM)** , ikinci durumdaki ara makamlara ise **Sertifika Makamı(SM)** denilmektedir. Kök sertifika makamları güveni kendisinden var olan ve kendi sertifikasını kendisi imzalayan son derece güvenilir olduğu düşünülen kuruluşlardır. Sertifika imzalama fonksiyonu olmayan diğer amaçlarla kullanılan sertifikalara **kullanıcı sertifikası** denilir.

Bir kullanıcı sertifikası için o sertifikadan başlayan sertifika imzalama(sertifika yayınlama) ilişkisi ile oluşan ve kök sertifika makamına kadar giden sertifika listesine **sertifika zinciri** denilir. Bir kullanıcı sertifikasına güvenilebilmesi için herşeyden once bu sertifikanın güvenilen bir SM ya da KSM sertifikasında sonlanan bir sertifika zincirine sahip olması gerekir. Bir sertifikanın güvenilir olup olmadığının ve dolayısıyla kullanılmakta olduğu iş(imzalama, şifreleme, kimlik doğrulama vb.) için uygun olup olmadığının bütün yönleriyle kontrol edildiği işlemler bütünü **sertifika doğrulama** işlemi olarak isimlendirilmektedir.

Bütün dünyada farklı SM'lerin farklı amaçlarla yayınladığı sertifikaların ortak bir yapısı olması amacıyla bir takım standartlar tanımlanmıştır. X.509 bu anlamda dünyaca kabül gören bir açık anahtar altyapısı(PKI) standardıdır. Bu standartta açık anahtar altyapısının en temel nesneleri olan , sertifikalar ve bu sertifikaların iptal durumlarının raporlandığı SİL'ler yapısal detaylarıyla tanımlanmıştır. X.509 standardına uyumlu sertifika ve SİL'lerin yapısı ve doğrulama adımları RFC 5280'de yer almaktadır.

Sertifika yayınlayan kuruluşlar sertifika oluşturmanın yanı sıra çeşitli sebeplerden ötürü yayınlamış olduğu sertifikaların geçerliliklerini iptal edebilirler. Bu durumda bu sertifikalar geçersiz başka bir ifade ile güvenilmez olurlar. Örneğin bir sertifikaya ilişkin kapalı anahtarın istenmeyen kişiler tarafından ele geçirilmesi sebebiyle sertifika sahibi kullanıcının yayıncı SM'ye bildirmesi sonucu SM tarafından bu sertifikanın iptali işlemi gerçekleştirilir. Kullanıcı sertifikalarının doğrulanmaya ihtiyaç duyulduğu yerlerde güvenli bir şekilde doğrulanabilmesi için bu sertifika iptal bilgilerinin güvenli ve kolay bir şekilde ulaşılabilir olması gerekmektedir. Bu işlem için iki yöntem kullanılmaktadır. Sertifikaların iptal durumlarının çevrimiçi yollarla gerçek zamanlı olarak sorgulandığı Çevrimiçi Sertifika Durum Protokolü ve İptal edilmiş sertifikaların iptal bilgilerinin bir veri yapısı oluşturularak dosya tabanlı olarak yayınlandığı SİL'ler. Bu iki yöntem de yine bütün dünyada aynı şekilde uygulanması ve uyumluluk amacıyla belirli standartlar çerçevesinde gerçekleştirilir.

3.3.2 Çevrimiçi Sertifika Durum Protokolü (ÇİSDUP)

RFC 2560'da tanımlı protokole uygun olarak bir veya daha fazla sertifikanın iptal durumunun bir sunucu vasıtasıyla çevrimiçi olarak sorgulanması yöntemidir. İngilizce kısaltmasıyla(OCSP) daha yaygın bilinen bu yöntemde temel olarak bir OCSP sorgusuna karşın sununun cevap olarak gönderdiği bir OCSP cevabı söz konusudur. Bu cevapta sertifikanın geçerlilik durumu yeralmaktadır. Bu sorgu geçerlilik süresi dolmuş sertifikalar için yapılamaz. Protokolün detayları ilgili RFC'de yer almaktadır.

3.3.3 SİL Dosyaları

Yine X.509 standardında sertifika iptal bilgilerinin listelendiği SİL dosyalarının yapısal özellikleri detaylı olarak tanımlanmıştır. SİL dosyaları temel olarak iptal edilmiş sertifikaların listesi ve bu listenin SM tarafından ya da SM tarafından yetkilendirilmiş başka bir makam tarafından oluşturulmuş imzasından oluşur.

Bu listede geçerlilik süresi dolmuş sertifikalar bulunma. Bir sertifikanın iptal durumu kontrol edilirken bir SİL'e başvurulduğnda öncelikle bu SİL'in de X.509'a uygunluğu ve geçerliliği kontrol edilmelidir. SİL için yapılması gereken bu kontroller bütününe de **SİL Doğrulama** denilmektedir. SİL dosyalarının yapısal detayları RFX 5280'de yer almaktadır.

3.3.4 X.509 Sertifika ve SİL Doğrulama

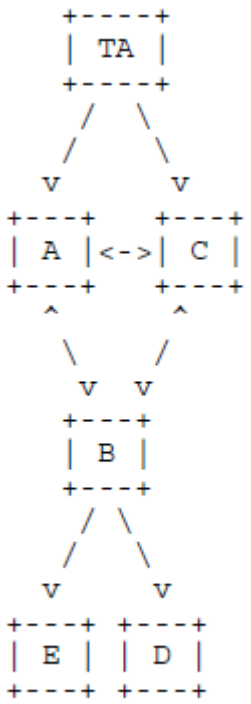
Sertifika ve SİL doğrulama işlemi sertifikanın ya da SİL'in güvenilir bir yetkili makam sertifikasına kadar uzanan sertifika zincirinin oluşturulması**(path building)** ve bu zincir üzerindeki bütün sertifikaların ve zincir ilişkisinin doğrulanmasından oluşan**(path validation)** iki alt kısımda incelenebilir.

Zincir doğrulama işlemi de zincir üzerindeki sertifikaların doğrulanması**(yapısal doğrulama)** ve zincir ilişkisinin doğrulanması**(zincir doğrulama)** şeklinde iki temel bölümden oluşur. Yapısal doğrulamada sertifika veya SİL'in X509'da belirtilen yapısal özellikleri karşılayıp karşılamadığına bakılır. Başka bir ifadeyle Sertifika veya SİL'in üzerinde yer alan bilgilerin standarda uygunluğu ve standarda gore geçerliliği kontrol edilir. Örneğin sertifikada seri numarası alanının olup olmadığına ve bu alanın pozitif bir sayı olduğuna bakılır. Ya da bir sertifikanın üzerinde yazan geçerlilik başlangıç ve bitiş tarihlerinin doğrulama zamanını kapsayıp kapsamadığı yapısal bir doğrulama adımıdır.

MA3 API Sertifika Doğrulama Kütüphanesi yapısal doğrulama kapsamında RFC 5280'de tanımlı bütün yapısal özellikleri kontrol eder. Zincir oluşturma algoritması olarak RFC 4158'de tanımlı zincir oluşturma algoritmasını ve zincir doğrulama algoritması olarak yine RFC 5280'de tanımlanmış zincir doğrulama algoritmasını gerçekler. Bölüm 3.4.1 ve Bölüm 3.4.2'de zincir doğrulama ve zincir oluşturmanın nasıl gerçekleştiği kısaca özetlenecektir.

3.3.4.1 Sertifika Zinciri Oluşturma

Zincir oluşturma algoritması RFC 4158'de detaylı bir şekilde anlatılmaktadır. Algoritma bir başlangıç sertifikasından başlayarak güvenilir bir kök sertifikaya ulaşıncaya kadar adım adım ilerleyerek bir sertifika zinciri oluşturur. Örneğin Şekil 13'deki gibi kurgulanmış bir sertifika ağacı olduğunu düşünelim.



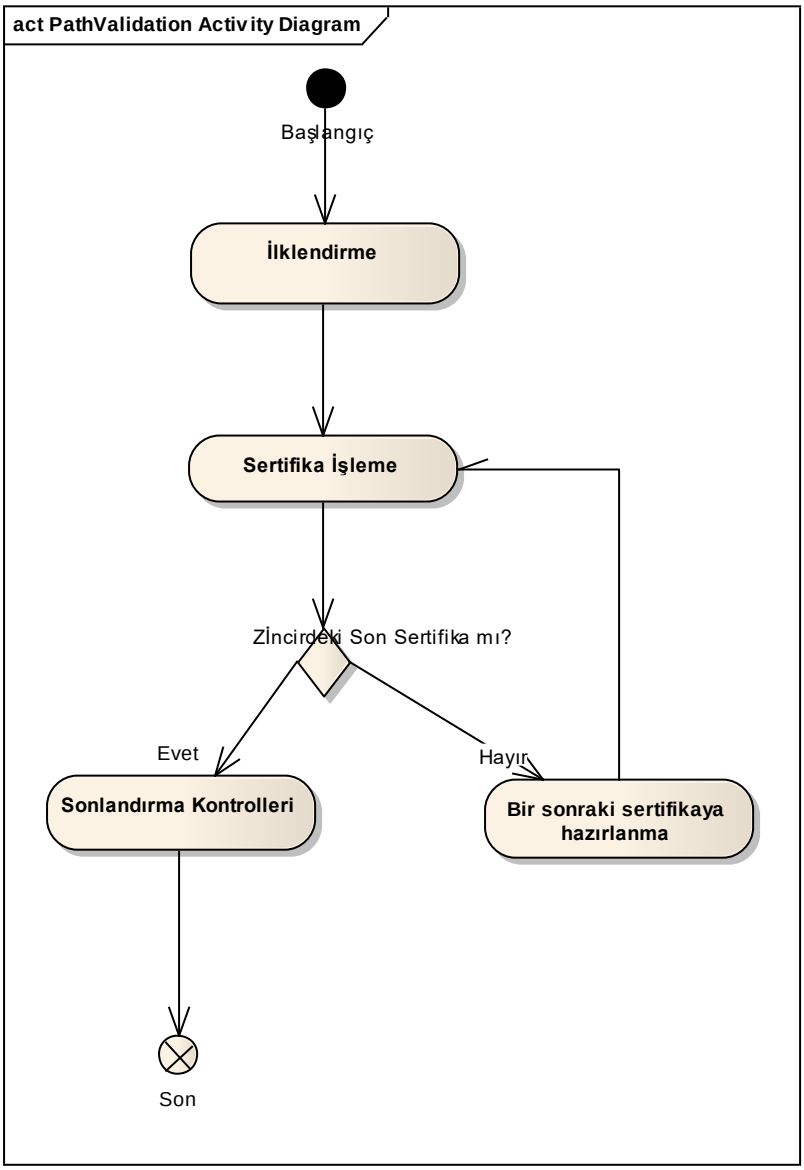
Şekil 13: Örnek Sertifika Ağacı

Sertifika ağacında **TA** ile gösterilen sertifika güvenilir bir SM ya da KSM sertifikasını temsil etmektedir. Bu durumda **E** sertifikasının **E**'den başlayarak **B** , **A** veya **C'**den geçerek **TA'** da biten bir sertifika zinciri oluşturulur.

Sertifika zinciri: **E → B → A → TA**

3.3.4.2 Sertifika Zinciri Doğrulama

Bir sertifika zinciri oluşturulduktan sonra bu zincirin doğrulanması gerekmektedir. Şekil 14'de zincir doğrulama algoritmasının akış diyagramı görülmektedir.



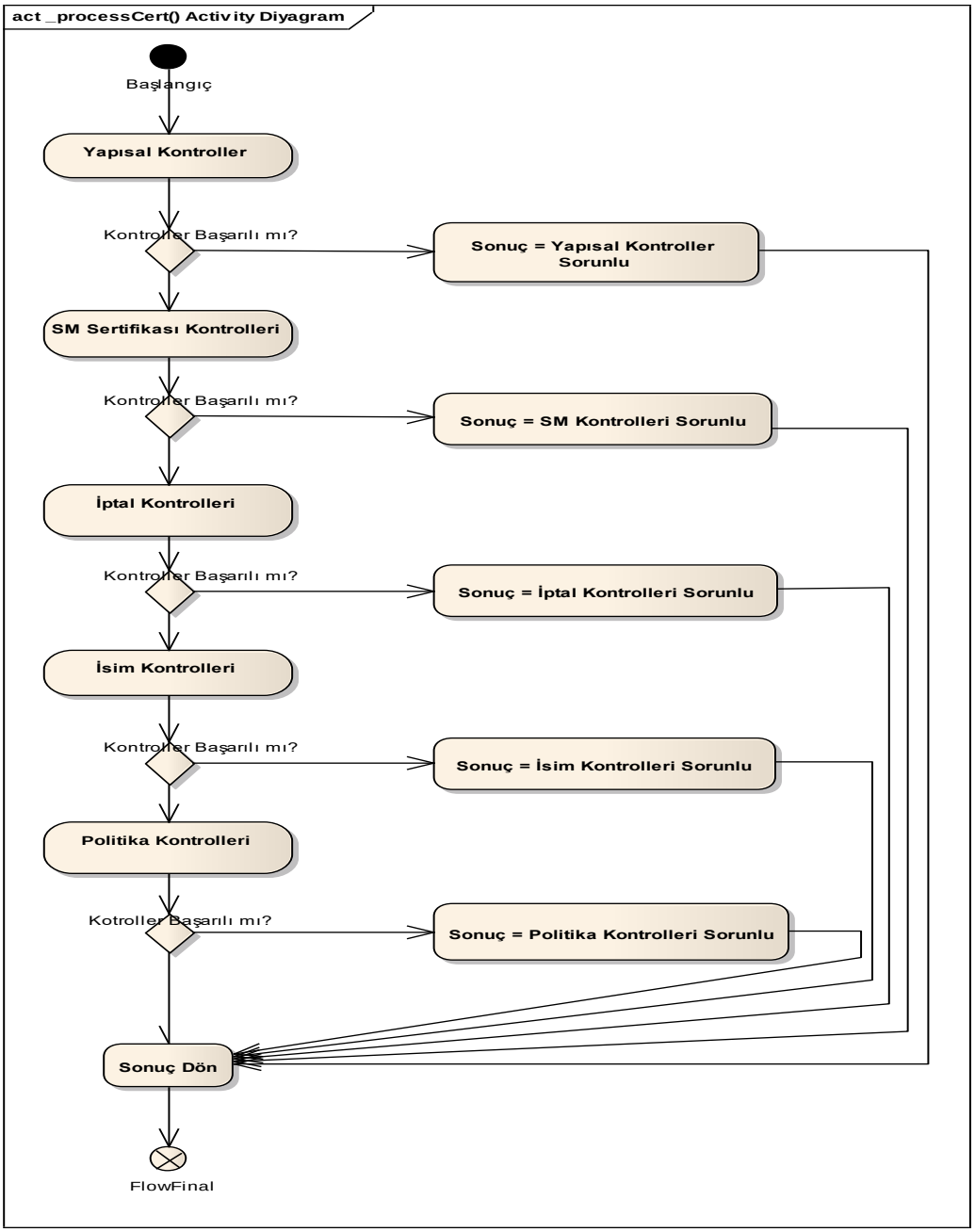
Şekil 14: Zincir Doğrulama Aktivite Diyagramı

Şekil 14 'de yer alan *Sertifika İşleme* ve *Sonlandırma Kontrolleri* işlemlerinin detayları RFC 5280'de verilmekte olup kısaca bir takım yapısal ve kriptografik kontrollerden oluştuğu söylenebilir.

SİL doğrulama işlemi sertifika doğrulama işlemi ile hemen hemen aynıdır. Burada tek fark sertifika doğrulamada oluşturulan zincirin ilk elemanı bir kullanıcı sertifikası iken SİL doğrulamada oluşturulan zincirin ilk elemanı bir SİL olmasıdır. Ve tabii ki SİL'In yapısal doğrulamasında sertifika yerine SİL'in yapısal özellikleri doğrulanmaktadır. Bunun dışında temel akış sertifika doğrulama ile aynıdır.

3.4 Sertifika Doğrulama Kütüphanesi Bileşenleri

MA3 API Sertifika Doğrulama kütüphanesi Bölüm 3.4.1 'de yer alan **zincir oluşturma** ve Bölüm 3.4.2'de açıklanan **zincir doğrulama** algoritmalarını kullanır. Özetle bir sertifika ya da SİL doğrulama isteğinde bulunulduğunda , önce güvenilir bir kök sertifikada sonlanan bir sertifika zinciri oluşturulur ve bu zincir doğrulanmaya çalışılır. Geçerli bir zincir bulununcaya kadar bu işlem tekrarlanır.



Şekil 15: MA3 API Sertifika Doğrulama - Sertifika İşleme Aktivite Diyagramı

MA3 API Sertifika Doğrulama Kütüphanesi zincir doğrulama sırasında gerçekleştirilen yapısal doğrulama ve zincir ilişkisinin doğrulanması ile ilgili kontrolleri Şekil 14' teki diyagramda yer alan *Sertifika İşleme* adımında gerçekleştirir. Sertifika İşleme adımının detayları Şekil 15' te görüntülenmektedir.

Bu aktivite gruplara ayrılmış bir takım kontrol işlemlerinden oluşur ve bu kontrol işlemleri bir **sertifika doğrulama politika dosyasında** çalışma zamanında yapılandırılabilir. Bu politika dosyasının yapısı Bölüm 4.4'teyer almaktadır. Bu kontrol işlemleri atomik olarak ayrı ayrı tanımlanmış ve her bir atomik kontrol için bir **kontrolcü**(`Checker`) sınıf tanımlanmıştır. Kontrolcü sınıflarının kontrol işlemlerini yaparken ihtiyaç duyduğu sertifika , SİL, OCSP cevabı gibi kaynakları çeşitli yollarla ve çeşitli yerlerden bulmak için **bulucu**(`Finder`) sınıfları tanımlanmıştır. Bulucuların bulduğu kaynakların doğru kaynaklar olup olmadığının test edilmesi gerekmektedir ve bu amaçla **eşleştirici**(`Matcher`) sınıfları oluşturulmuştur. MA3 API Sertifika Doğrulama Kütüphanesi hangi kontrollerin yapılacağını yani hangi kontrolcü sınıflarının kullanılacağını, ihtiyaç duyulan kaynaklar için nerelere bakılacağını yani hangi bulucu sınıflarının kullanılacağını ve bulunan kaynakların nasıl eşleştirileceğini yani hangi eşleştirici sınıflarının kullanılacağını belirlemek amacıyla çalışma zamanında işlemek üzere **doğrulama politikası dosyasını** kullanır. Doğrulama işleminin öncesinde bu politika dosyası okunarak gerekli kontrolcü, bulucu, ve eşleştirici alt sınıfları oluşturulur.

3.4.1 Kontrolcüler

Sertifika ve SİL'lerin yapısal özelliklerinin ve sertifika zincir ilişkisinin standartlara uygunluğunu kontrol eden sınıflardır. Bu sınıfların detayları bu dökümanın kapsamı dışındadır. Bu kontrolcülerin hangilerinin çalışacağı doğrulama politika dosyası aracılığıyla belirlenir. Sertifika doğrulama işleminde gerçekleştirilen RFC 5280'de tanımlanan kontrolcüler temel olarak şunlardır:

3.4.2 Yapısal Kontroller

Yapısal kontroller sertifikanın ve SİL'in üzerindeki bilgilerin içerik ve biçimsel olarak geçerliliğinin kontrollerini kapsar. Temel olarak yapısal kontroller aşağıdaki gibi listelenebilir.

- **Seri Numarası Kontrolü:** Sertifika üzerindeki seri numarasının pozitif bir tamsayı olduğunun kontrolüdür.
- **Sertifika Geçerlilik Tarihi Kontrolü:** Sertifika üzerindeki geçerlilik tarihleri doğrulama zamanını kapsadığının kontrolüdür.
- **Versiyon Kontrolü:** Sertifika üzerindeki versiyon bilgisinin geçerliliğinin kontrolüdür.
- **Eklenti Kontrolü:** Sertifika üzerindeki eklentilerin geçerliliğinin kontrolüdür.
- **İmza Algoritması Kontrolü:** Sertifika üzerinde yazan imza algoritmasının imzalanmış imza algoritması bilgisiyle aynı olması kontrolüdür.

3.4.3 Zincir İlişkisi Kontrolleri

Zincir İlişkisi kontrolleri, sertifika veya SİL ile bu seritifika veya SİL'i yayınlayan yetkili makam sertifikası arasındaki ilişkiyi inceleyen kontrolleri içerir.

- **İsim Kontrolü:** Sertifika veya SİL üzerindeki yayıncı(issuer) alanı ile yayıncı sertifikası üzerindeki özne(subject) alanının aynı olduğunun kontrolü.
- **İmza Kontrolü:** Sertifika veya SİL üzerindeki imzanın yayınlayan sertifikasındaki açık anahtar ile kriptografik olarak doğrulanması.
- **Temel Kısıtlar Kontrolü:** Yayıncı sertifikası üzerinde Temel Kısıtlar(Basic Constraints) eklentisinin bulunduğu ve bu eklentideki yayıncı işaretçisinin RFC 5280'de belirtildiği şekilde yeraldığı kontrolü.
- **Anahtar Tanımlayıcısı Kontrolü:** Sertifika veya SİL üzerindeki yetkili anahtar tanımlayıcısı(Authorithy Key Identifier) eklentisindeki değer ile yayıncı sertifikasındaki anahtar tanımlayıcısı değerinin uyuşması kontrolü.
- **Yol Uzunluğu Kontrolü:** Sertifika zincirinin uzunluğunun yayıncı sertifikasındaki yol uzunluğu eklentisinde yer alan değerden uzun olmaması kontrolü.
- **Anahtar Kullanımı Kontrolü:** Yayıncı sertifikasındaki anahtar kullanımı(Key Usage) eklentisinde sertifika imzalama özelliğinin bulunması kontrolü.

3.4.4 İptal Kontrolleri

Sertifikanın iptal durumu kontrollerini içerir.

- **SİL' den İptal Durumu Kontrolü:** Sertifikaya ilişkin SİL'den sertifikanın iptal durumunun kontrolü.
- **OCSP' den İptal Durumu Kontrolü:** Sertifikaya ilişkin OCSP cevabından sertifikanın iptal durumunun kontrolü.

3.4.5 İsim Kontrolleri

Sertifikadaki özne bilgisi yayıncı sertifikasında yer alan isim kısıtlamaları eklentisinde tanımlı bir takım kurallara göre değerler alabilir. Bu detaylar RFC 5280'de belirtilmiştir ve isim kontrolleri kapsamında bu kurallara uyulup uyulmadığı kontrol edilmektedir.

3.4.6 Politika Kontrolleri

Sertifikadaki politika bilgileri eklentisinde yer alan politika bilgileri yayıncı sertifikasındaki isim kısıtlamaları eklentisinde tanımlı bir takım kurallara göre değerler alabilir. Bu detaylar RFC 5280'de belirtilmiştir ve politika kontrolleri kapsamında bu kurallara uyulup uyulmadığı kontrol edilmektedir.

MA3 API Sertifika Doğrulama kütüphanesinde tanımlı ve politika dosyasında kullanılabilecek tüm kontrolcülerin listesi EK-A' daki **Table 1**' de yer almaktadır.

3.4.7 Bulucular

Kontrolcü sınıflar çalışırlarken bir takım dış verilere ihtiyaç duyabilirler. Bu bir SİL bilgisi, bir OCSP sorgusu ya da bir SM sertifikası olabilir. Yine zincir oluşturma sırasında SM sertifikalarının çeşitli yerlerden(http adresi, yerel seritifika deposu vb.) bulunması gerekebilir. Genel olarak sertifika doğrulama sırasında dışarıdan bulunması gereken sertifika , SİL, OCSP cevabı verilerini bulma işlemini bulucu sınıflar gerçekleştirir. Bir sertifika , SİL ya da OCSP cevabı bulunurken politika dosyasında tanımlı Bulucular sırayla çalıştırılır ve aranılan veri bulunduğunda bulma işlemi sonlandırılır. Bu nedenle bulucuların uygun bir sırayla politika dosyasına yerleştirilmiş olması performans açısından oldukça önemlidir. Örneğin yerel depodan sertifika bulucunun, uzak kaynaklardan(LDAP,HTTP vb.) sertifika buluculardan önce yerleştirilmesi sertifika arama işlemlerinin sertifika yerel depoda varsa uzak kaynaklara başvurulmadan sonuçlanmasını sağlar ve bu da ciddi performans kazanımları getirir. Sertifika Doğrulama Politikası aracılığıyla hangi bulucuların hangi sırayla çalıştırılacakları bilgisi tanımlanmaktadır.

MA3 API Sertifika Doğrulama Kütüphanesinde tanımlı bulucular şu şekilde gruplandırılabilir:

3.4.7.1 Güvenilir Sertifika Bulucular

Doğrulama kütüphanesinin güvenilir olarak tanıdığı SM sertifikalarını bulan sınıflardır.

Bulunan güvenilir sertifikalar varsayılan olarak kendinden imzalı (self-issued) sertifikaları kapsayacak şekilde filtrelenmektedir. Bu filtrelemeyi kaldırmak için sertifika doğrulama politika dosyasındaki parametre alanının aşağıdaki şekilde güncellenmesi gerekmektedir.

```
<parameters>
        ....
       <TrustOnlySelfSignedCertificate value="false"/>

</parameters>
```

3.4.7.2 Sertifika Bulucular

SM sertifikası bulmaktan sorumlu sınıflardır.Bu bulucular sertifikanın üzerinde yer alan Yetkili Erişim Noktası(AIA) eklentisine bakarak SM sertifikası bulan sınıflar olabileceği gibi bir http adresinden bir LDAP adresinden ya da dosya sistemindeki bir adresten ya da yerel sertifika deposundan sertifika bulan sınıflar olabilir.

3.4.7.3 SİL Bulucular

SİL bulmaktan sorumlu sınıflardır. Sertifika üzerinde yazan Sil Dağıtım Noktaları(CDP) eklentisine bakarak SİL bulan sınıflar olabileceği gibi bir http adresinden bir LDAP adresinden ya da dosya sistemindeki bir adresten ya da yerel sertifika deposundan SİL bulan sınıflar olabilir.

3.4.7.4 OCSP Cevabı Bulucular

OCSP cevabı bulmaktan sorumlu sınıflardır. Sertifika üzerinde yazan Yetkili Erişim Noktası(AIA) bakarak OCSP cevabı bulan sınıflar olabileceği gibi bir http adresinden bir LDAP adresinden ya da dosya sistemindeki bir adresten ya da yerel sertifika deposundan OCSP cevabı bulan sınıflar olabilir.

MA3 API Sertifika Doğrulama Kütüphanesinde tanımlı ve politika dosyasında kullanılabilecek tüm bulucuların listesi EK-A' daki

Table 3' te yer almaktadır.

3.4.8 Eşleştiriciler

Bulunan sertifika, SİL ya da OCSP cevabı bilgilerinin gerçekten aranılan veriler olup olmadığını anlamak için gerekli eşleştirmeden sorumlu sınıflardır.Bu eşleştirme kuralları RFC 5280'de yer almaktadır.

MA3 API Sertifika Doğrulama Kütüphanesi'nde tanımlı bulucular şu şekilde gruplanabilir.

3.4.8.1 Sertifika Eşleştiriciler

Sertifika ile yayıncı sertifikasını eşleştiren sınıflardır. Örneğin RFC 5280'de bir X.509 sertifikası üzerinde yazan *issuer* alanı ile yayıncı sertifikası üzerinde yer alan *subject* alanı aynı olmalıdır şeklinde bir ifade yer alır. MA3 API Sertifika Doğrulama ifadesinde bu eşleştirme işini gerçekleştiren eşleştirici(`IssuerSubjectMatcher`) sınıfı vardır. Eşleştiricilerin eşleştiremediği sertifikalar yayıncı sertifikası olarak sertifika zincirine eklenmezler.

3.4.8.2 SİL Eşleştiriciler

Sertifika ile SİL'i ya da SİL ile SİL yayınlayan sertifikayı eşleştiren sınıflardır. Örneğin RFC 5280'de , dolaylı SİL(indirect crl) kullanılmıyorsa, bir SİL üzerinde yazan *issuer* alanı ile yayıncı sertifikası üzerinde yer alan *subject* alanı aynı olmalıdır şeklinde bir ifade yer alır. Bir SİL'in yayıncı sertifikası olarak bir sertifikanın bulunması için bu ifadenin doğruluğu kontrol edilir. Bu kontrol işlemi için bir eşleştirici(`CRLIssuerMatcher`) tanımlıdır.

3.4.8.3 OCSP Cevabı Eşleştiriciler

Sertifika ile OCSP cevabını eşleştiren sınıflardır. Çalışma şekilleri sertifika ve SİL eşleştiriciler gibidir.

3.4.8.4 Delta SİL Eşleştiriciler

SİL ile Delta-SİL'i eşleştiren sınıflardır. Çalışma şekilleri sertifika ve SİL eşleştiriciler gibidir.

3.4.8.5 Çapraz Sertifika Eşleştiriciler

SM Sertifikası ile çapraz SM sertifikasını eşleştiren sınıflardır. Çalışma şekilleri sertifika ve SİL eşleştiriciler gibidir.

MA3 API Sertifika Doğrulama Kütüphanesi'sinde tanımlı ve politika dosyasında kullanılabilecek tüm eşleştiricilerin listesi EK-A'daki

Table 2'de yer almaktadır.

3.4.9 Kaydediciler

MA3 API Sertifika Doğrulama Kütüphanesi doğrulama sırasında bulduğu sertifika ve SİL'leri yerel depoya kaydeder. Bu işlemi kaydediciler(Saver) gerçekleştirir ve yine politika dosyası aracılığıyla hangi kaydedicilerin çalışacağı tanımlanabilir. Örneğin istenildiği kadar kaydedici tanımlanarak doğrulama sırasında bulunan ve doğrulanan bütün sertifikalar istenilen yerlere kaydedilirler.

MA3 API Sertifika Doğrulama Kütüphanesi'nde tanımlı ve politika dosyasında kullanılabilecek tüm kaydedicilerin listesi EK-A'daki **Table 4**'te yer almaktadır.

3.4.10 Sertifika Doğrulama Politikası

Sertifika doğrulama işlemi bir çok kontrol işleminin arka arkaya yapılmasından oluşan bir süreçtir. Bu kontrol işlemlerinin hangilerinin yapılıp hangilerinin yapılmayacağını ve bu kontroller sırasında kullanılan bir takım parametrelerin çalışma zamanında belirlenebilmesi için XML formatında bir doğrulama politikası dosyası kullanılmaktadır. Şekil 4'te örnek bir doğrulama politikası dosyasından bir bölüm görüntülenmektedir. Bu XML dosyasında yer alan her bir "class" elemanı bir sınıfı temsil eder ve bu sınıflar doğrulamada kullanılacak kontrol ,bulma ve eşleştirme sınıflarıdır. Bu dosya doğrulama işleminin başında okunarak doğrulama politikası nesnesi oluşur ve bütün doğrulama adımları bu politikada belirtilen yapılandırma bilgileri doğrultusunda gerçekleştirilir. Doğrulamada kullanılacak kontrol , bulma ve eşleştirme sınıfları , sertifika politikası kontrolleri ile ilgili ayarlar bu yapılandırma bilgilerinden bazılarıdır.

```
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <find>
        <trustedcertificate>
            <class name="tr.gov.tubitak.uekae.esya.api.certificate.validation.find.certificate.trusted.TrustedCertificateFinderFromECertStore">
                <param name="securitylevel" value="legal,organizational"/>
            </class>
            <<class name="tr.gov.tubitak.uekae.esya.api.certificate.validation.find.certificate.trusted.TrustedCertificateFinderFromFileSystem">
                <param name="dizin" value="T:\MA3\kok"/>
            </class>
        </trustedcertificate>
        <certificate>
            <class name="tr.gov.tubitak.uekae.esya.api.certificate.validation.find.certificate.CertificateFinderFromECertStore"/>
            <class name="tr.gov.tubitak.uekae.esya.api.certificate.validation.find.certificate.CertificateFinderFromHTTP"/>
            <class name = "tr.gov.tubitak.uekae.esya.api.certificate.validation.find.certificate.CertificateFinderFromLDAP"/>
        </certificate>
        <deltacrl/>
    </find>
    <match>
        <certificate>
            <class name="tr.gov.tubitak.uekae.esya.api.certificate.validation.match.certificate.IssuerSubjectMatcher"/>
        </certificate>
        <crl>
            <class name="tr.gov.tubitak.uekae.esya.api.certificate.validation.match.crl.CRLIssuerMatcher"/>
        </crl>
        <deltacrl>
            <class name = "tr.gov.tubitak.uekae.esya.api.certificate.validation.match.deltacrl.BaseCRLNumberMatcher"/>
            <class name = "tr.gov.tubitak.uekae.esya.api.certificate.validation.match.deltacrl.CRLNumberMatcher"/>
            <class name = "tr.gov.tubitak.uekae.esya.api.certificate.validation.match.deltacrl.DeltaCRLIssuerMatcher"/>
            <class name = "tr.gov.tubitak.uekae.esya.api.certificate.validation.match.deltacrl.ScopeMatcher"/>
        </deltacrl>
        <ocsp>
            <class name = "tr.gov.tubitak.uekae.esya.api.certificate.validation.match.ocsp.CertIDOCSPResponseMatcher"/>
```

Şekil 16: Örnek Sertifika Doğrulama Politikası dosyası

3.4.10.1 Sertifika Doğrulama Politika Dosyasının Güvenliği

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

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

Politika dosyası için aşağıdaki güvenlik önlemleri uygulanabilir.

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

Yukarıdaki güvenlik önlemleri birlikte veya tek olarak uygulanabilir. Politika dosyasının güvensiz bir şekilde istemcilerde durmasını **kesinlikle önermiyoruz.**

3.4.11 Sertifika - SİL Durum Bilgisi

MA3 API Sertifika Doğrulama Kütüphanesi sertifika ve SİL doğrulama işlemi sonucunda Sertifika Durum Bilgisi(**CertificateStatusInfo**) ve SİL Durum Bilgisi(**CRLStatusInfo**) sınıflarından bir nesne oluşturur. Bu sınıf içerisinde ayrıntılı kontrol detayları , oluşturulmuş ve doğrulanmaya çalışılan farklı sertifika zincirleri doğrulanma sonuçlarıyla bir şekilde detaylı olarak saklanır. Bu sınıfın nesnesinin kullanımı örnek kodlarla birlikte Bölüm 4'te daha detaylı olarak görülebilir.

3.4.12 Yerel Sertifika Deposu

Sertifika deposu güvenilir olduğu kabul edilen yayıncı sertifikalarının saklanabileceği bir sertifika deposu geliştirilmiştir.

Sertifika deposu aynı zamanda doğrulama işlemi sırasında bulunması gereken yayıncı sertifikası, SİL, OCSP cevabı gibi verilerin sürekli uzak kaynaklardan çekilmesini engellemek için bu verilerin yerelde saklanabilmesinden sorumludur.

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

Sertifika Deposu'unun daha detaylı bilgileri için *Sertifika Deposu Dökümanına* başvurulmalıdır.

MA3 API Sertifika Doğrulama Kütüphanesi yerel sertifika deposunda güvenilir kök sertifikası olarak tanımlanmış seritikaları güvenilir olarak kabul eder ve doğrulama sırasında oluşturduğu sertifika zincirlerinin bu sertifikalardan biriyle sonlanmasını bekler. API'de yer alan yerel sertifika deposundan sertifika ve sil bulucular aracılığıyla doğrulama verileri istenildiğinde yerel depodan bulunabilir.

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

```
<class
name="tr.gov.tubitak.uekae.esya.api.certificate.validation.find.certificate.trusted.T
rustedCertificateFinderFromECertStore">
<param name="storepath" value="T:\MA3\api-parent\certStore\SertifikaDeposu.svt"/>
</class>
```

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

```
<class
name="("tr.gov.tubitak.uekae.esya.api.certificate.validation.find.certificate.trusted
.TrustedCertificateFinderFromFileSystem">
<param name="dizin" value="T:\\MA3\\api-
cmssignature\\testdata\\support\\UGRootCerts\\"/>
</class>
```

3.4.13 XML Sertifika Deposu

Dosya sistemine herhangi bir şekilde erişimin olmadığı durumlarda kullanılması için XML tabanlı sertifika deposu dosyası geliştirilmiştir. Bu şekilde uzaktaki bir depo ile çalışabilmek mümkün olmaktadır.

Sertifika doğrulama konfigürasyon dosyasında XML depodan güvenilir sertifika bulucu aşağıdaki gibi tanımlanır:

```
<class
name="tr.gov.tubitak.uekae.esya.api.certificate.validation.find.certificate.trusted.T
rustedCertificateFinderFromXml">

<param name="storepath" value="http://www.kamusm.gov.tr/depo/xml/XmlDepo.xml"/>

</class>
```

Yine xml depodan sertifika bulucu aşağıdaki şekilde tanımlanır:

```
<class
name="tr.gov.tubitak.uekae.esya.api.certificate.validation.find.certificate.Certifica
teFinderFromXml">
<param name="storepath" value="http://www.kamusm.gov.tr/depo/xml/XmlDepo.xml"/>

</class>
```

XML depodan güvenilir sertifika bulucu parametre olarak bir URL almaktadır. Performans arttırmak için yerel ağlarda bu dosyanın bir kopyasını bulundurmak ve onunla çalışmak faydalı olur.

Bunun yanında xml sertifika deposu sadece MA3 API Sertifika Doğrulama Kütüphanesi ile kullanılmalıdır. Aksi taktirde man-in-the-midle saldırısına kapı açılmış olur. MA3 API Kütüphanesi bu dosyadaki güvenilir sertifikalar için imza kontrolü yapmaktadır.

Bir diğer göz önünde bulundurulması gereken konu performanstır. Yerel sertifika deposu kullanılırken imza doğrulamada kullanılan iptal listeleri depoya kaydedilip tekrar kullanılabilirken, xml sertifika deposu sadece okuma özellikli olduğu için bu katkıdan mahrum kalınacak, iptal listeleri her doğrulamada tekrar indirilmek zorunda kalınacaktır. Bu da ciddi performans kaybına yol açar.

3.5 Sertifika Doğrulama Kütüphanesi Kullanımı

Sertifika doğrulama kütüphanesinin kullanı için gerekenlere GEREKLER bölümünden bakabilirsiniz. Sertifika doğrulama kütüphanesi aldığı sertifikayı verilen politika dosyasına uygun olarak doğrulamaktadır.

Sertifika doğrulama işlemi için kütüphaneye sertifikanın hangi tarihte doğrulanacağı bilgisi verilmelidir. Sertifika iptal kontrolü verilen tarihe göre yapılacaktır. Verilen tarihten önce iptal edilmiş sertifikaların durumu geçersiz olacaktır. Şu anki zamanda sertifika doğrulama için aşağıdaki örnek kod kullanılabilir.

Java

```
ValidationSystem vs = CertificateValidation.createValidationSystem(policy);
vs.setBaseValidationTime(Calendar.getInstance());
CertificateStatusInfo csi = CertificateValidation.validateCertificate(vs, cert);
```

C#

```
ValidationSystem vs = CertificateValidation.createValidationSystem(policy);
vs.setBaseValidationTime(DateTime.UtcNow);
CertificateStatusInfo csi = CertificateValidation.validateCertificate(vs, cert);
```

3.6 Sertifika Doğrulama Sonucunun Yorumlanması

Sertifika doğrulama sonucunda *CertificateStatusInfo* nesnesi dönmektedir. Bu nesnenin *toString()* methodu çağrılarak sertifika doğrulamanın sonucunu metin olarak alabilirsiniz. *getDetailedMessage()* fonksiyonu ile de sertifika doğrulama sonucununun kullanıcı dostu mesajını alabilirsiniz. Aynı nesnenin *getCertificateStatus()* fonksiyonu kullanılarak *CertificateStatus* tipinde sertifika doğrulamanın durumu alınabilir. Bu nesne aşağıdaki değerleri alabilir.

VALID: Sertifika geçerlidir.

REVOCATION_CHECK_FAILURE: Sertifika iptal edilmiştir.

***CERTIFICATE_SELF_CHECK_FAILURE*:** Sertifikada yapısal bozukluk vardır.

***NO_TRUSTED_CERT_FOUND*:** Hiç güvenilir sertifikanız yok.

***PATH_VALIDATION_FAILURE*:** Güvenilir bir sertifika zinciri oluşturulamadı. Sertifikanızın kök sertifikası güvenilir sertifikalarınızın içinde yok.

***NOT_CHECKED*:** Sertifika iptal kontrolü yapılamadığından sertifika doğrulanamadı.

3.7 Politika Dosyası

İndirdiğiniz kütüphane ile birlikte gelen politika dosyası bir sertifika doğrulamanın başarılı sayılabilmesi için gerekli yeterliliği sağlamaktadır. Değiştirilme ihtiyacının duyulabileceği iki ayar olabilir.

Örnek olarak verilen politika dosyası sadece sertifika deposu ile çalışmaktadır. Sertifika deposunda ise sadece kanuni geçerliliği olan kök sertifikalar bulunmaktadır. Dolayısıyla test sistemi ile çalışmak için test sisteminin köklerinin de kütüphaneye güvenilir kök olarak gösterilmesi gerekmektedir. Bunun için Test Sistemi için Politika Dosyasının Düzenlenmesi bölümüne bakabilirsiniz.

Politika dosyasında yapılacak bir diğer düzenleme sertifika iptal kontrollerinde ÇİSDUP veya SİL kullanılmasının ayarlanmasıdır. Bir sertifikanın doğrulanması sırasında sertifika zincirinin tamamının doğrulanması gerekmektedir. Zincirdeki bütün sertifikalar için ÇİSDUP desteği verilmeyebilir. Örneğin KamuSM sisteminde alt kökler için ÇİSDUP hizmeti verilmemektedir. Dolayısıyla sadece ÇİSDUP kullanarak bir sertifikayı doğrulamak mümkün değildir. Bu durumda ÇİSDUP öncelikli SİL ve ÇİSDUP'un bulunduğu konfigürasyon veya sadece SİL ile çalışan konfigürasyon yaratılabilir.

Politika dosyasında ÇİSDUP veya SİL'den hangisine öncelik verilmesi isteniyorsa o sınıf üste yazılmalıdır.

3.7.1 Politikanın Çalışma Zamanında Düzenlenmesi

Politika dosyası dosyadan okuduktan sonra çalışma zamanında düzenlenebilir. Aşağıdaki kod parçası klasörden güvenilir sertifikaları gösteren *TrustedCertificateFinderFromFileSystem* sınıfını çalışma zamanında eklemektedir.

Java

```
ValidationPolicy POLICY = PolicyReader.readValidationPolicy(new
FileInputStream(POLICY_FILE));

//For UEKAE Test Environment, we add our test roots.
HashMap<String, Object> parameters = new HashMap<String, Object>();
parameters.put("dizin", "T:\\MA3\\api-
cmssignature\\testdata\\support\\UGRootCerts\\");
POLICY.bulmaPolitikasiAl().addTrustedCertificateFinder("tr.gov.tubitak.uekae.esya.api
.certificate.validation.find.certificate.trusted.TrustedCertificateFinderFromFileSyst
em", parameters);
```

C#

```
ValidationPolicy POLICY = PolicyReader.readValidationPolicy(POLICY_FILE);

//For UEKAE Test Environment, we add our test roots.
Dictionary<String, Object> parameters = new Dictionary<String, Object>();
parameters.Add("dizin", "T:\\MA3\\api-
cmssignature\\testdata\\support\\UGRootCerts\\");
POLICY.bulmaPolitikasiAl().addTrustedCertificateFinder("tr.gov.tubitak.uekae.esya.api
.certificate.validation.find.certificate.trusted.TrustedCertificateFinderFromFileSyst
em", parameters);
```

3.8 Poltika Dosyası Elemanları

Politika dosyasında tanımlanan sınıflar aşağıdaki tablolarda listelenmiştir.

Bu tablolardaki gösterimde;

- Parametrenin alabileceği değerlerin listelendiği yere * ile belirtilen değer parametrenin varsayılan değeridir.
- [O] değeri ile belirtilen parametre opsiyonel parametre anlamına gelir.
- Başlık satırlarındaki başlığın yanındaki XML tagları o başlık altında yer alan sınıfların politika dosyasında hangi XML tag değerinin altında listelenmesi gerektiğini(yani politika dosyasındaki yerini) belirtir.

Table 1 MA3 API Kontrolcü (`Checker`) Listesi

GÜVENİLİR SERTİFİKA YAPISAL KONTROLCÜLER<policy><validate><certificate><trustedcertificate>			
CertificateDateChecker		Sertifika üzerinde yer alan geçerlilik zaman aralığının doğrulama zamanını kapsaması kuralını doğrular.	
SelfSignatureChecker		Sertifika üzerindeki imzayı sertifikanın açık anahtarı ile kriptografik olarak doğrular.	
SERTİFİKA YAPISAL KONTROLCÜLER<policy><validate><certificate><self>			
CertificateDateChecker		Sertifika üzerinde yer alan geçerlilik zaman aralığının doğrulama zamanını kapsaması kuralını doğrular.	
CertificateExtensionChecker		Sertifika üzerinde yer alan eklenti bilgilerinin RFC 5280 uyumluluğunu kontrol eder.	
PositiveSerialNumberChecker		Sertifika seri numarasının pozitif bir tamsayı olması kuralını doğrular.	
SignatureAlgConsistencyChecker		Sertifika üzerinde yer alan İmza Algoritması bilgilerinin uyuşması kuralını doğrular.	
VersionChecker		Sertifika versiyon bilgisinin RFC 5280 uyumluluğunu kontrol eder.	
QualifiedCertificateChecker		Sertifikanın nitelikli kontrolü yapılır.	
	Parametreler	`statementoids`	Verilen OID'lere göre sertifikanın nitelikli kontrolleri yapılır. Bazı nitelikli sertifika kontrollerinde iki tane OID kontrol edilmek istenebilir. Bu gibi durumlarda AND ile OID'ler birleştirilebilir. Değişik ülkeler değişik OID'ler kullanabilmektedir. Bu tür durumlar için de OID'ler OR ile ayrılabilir. Örn: "(0.4.0.1862.1.1 AND 2.16.792.1.61.0.1.5070.1.1) OR (4.3.2.1 AND 1.2.3.4)"

SERTİFİKA ZİNCİR KONTROLCÜLERİ<policy><validate><certificate><issuer>	
BasicConstraintCAChecker	Yayıncı Sertifikası üzerindeki Temel Kısıtlamalar (*BasicConstraints*) eklentisinin *RFC 5280* uyumluluğunu kontol eder.
CertificateKeyUsageChecker	Yayıncı Sertifikası üzerindeki Anahtar Kullanımı (*KeyUsage*) eklentisinin *RFC 5280* uyumluluğunu kontol eder.
CertificateNameChecker	Sertifika üzerindeki yayıncı özne adı (issuer) alanı ile yayıncı sertifikası üzerindeki özne adı (subject) alanlarının eşleşmesi kuralını doğrular.
CertificateSignatureChecker	Sertifika üzerindeki imzayı yayıncı sertifikasının açık anahtarı ile kriptografik olarak doğrular.
KeyIdentifierChecker	Sertifika üzerindeki Yetkili Anahtar Tanımnlayıcısı (AuthorityKeyIdentifier) eklentisi ile yayıncı sertifikası üzerinde yer alan Özne Anahtar Tanımlayıcısı (SubjectKeyIdentifier) eklentilerin uyumluluğunu kontrol eder.
NameConstraintsChecker	Sertifikanın özne adının yayıncı sertifikasında (varsa) yer alan İsim Kısıtlamaları (NameConstraints) arasındaki ilişkinin RFC 5280 uyumluluğunu kontrol eder.
PathLenConstraintChecker	Yayıncı Sertifikası üzerindeki Yol Uzunluğu Kısıtlamaları (PathLengthConstraints) eklentisinin RFC 5280 uyumluluğunu kontrol eder.
PolicyConstraintsChecker	Yayıncı Sertifikası üzerindeki Politika Kısıtlamaları (PolicyConstraints) eklentisinin RFC 5280 uyumluluğunu kontrol eder.

SERTİFİKA İPTAL KONTROLCÜLERİ <policy><validate><certificate><revocation>			
RevocationFromCRLChecker		SİL'e bakarak sertifikanın iptal durumunu kontrol eder.	
	Parametreler	`cevrimdisicalis`	`[true,false*]` Bu parametre doğrulamanın çevrimdışı olarak yürütüleceğini belirtir. True olarak tanımlanmışsa iptal kontrolcü hiçbir SİL bulamasa bile iptal kontrolünü başarılı olarak sonlandırır. Çevrimdışı ortamlarda SİL'e ulaşılamadığında bile sertifika doğrulamanın gerçekleşebilmesi için tanımlanmıştır.. Dikkatli kullanılmalıdır!
		`checkAllCRLs`	`[true,false*]` Doğrulama sırasında normal olarak bir adet geçerli SİL kontrolü iptal kontrolünün tamamlanması için yeterlidir. Ancak bazı durumlarda kullanıcılar politika tanımlı bütün SİL bulucuların getirdikleri SİL'lere bakılmasını isteyebilirler. Bu durumda bu değer `true` yapılmalıdır.
		`devam`	`[true,false*]` Doğrulama sırasında normal olarak bir adet iptal kobtrolcünün başarılı sonuçlanması iptal kontrolünün tamamlanması için yeterlidir. Ancak kullanıcılar daha güvenli olması amacıyla bir iptal kontrolü başarılı sonuçlansa bile diğer iptal kontrolcülerin kontrolüne devam etmesini isteyebilirler. Bu durumda bu parametre `true` yapılmalıdır.

RevocationFromOCSPChecker			
	Parametreler	`devam`	`[true,false*]` Doğrulama sırasında normal olarak bir adet iptal kobtrolcünün başarılı sonuçlanması iptal kontrolünün tamamlanması için yeterlidir. Ancak kullanıcılar daha güvenli olması amacıyla bir iptal kontrolü başarılı sonuçlansa bile diğer iptal kontrolcülerin kontrolüne devam etmesini isteyebilirler. Bu durumda bu parametre `true` yapılmalıdır.
SİL YAPISAL KONTROLCÜLER<policy><validate><crl><crlself>			
CRLDateChecker		SİL üzerinde yer alan geçerlilik zaman aralığının doğrulama zamanını kapsaması kuralını doğrular.	
CRLExtensionChecker		SİL üzerinde yer alan eklenti bilgilerinin RFC 5280 uyumluluğunu kontrol eder.	
SİL ZİNCİR KONTROLCÜLERİ<policy><validate><crl><crlissuer>			
CRLKeyUsageChecker		Yayıncı Sertifikası üzerindeki Anahtar Kullanımı (KeyUsage) eklentisinin RFC 5280 uyumluluğunu kontol eder.	
CRLSignatureChecker		SİL üzerindeki imzayı yayıncı sertifikasının açık anahtarı ile kriptografik olarak doğrular.	
DELTA SİL KONTROLCÜLERİ<policy><validate><deltacrl>			
FreshestCRLChecker		Delta SİL üzerindeki En Güncel SİL (*FreshestCRL*) eklentisinin RFC 5280 uyumluluğunu kontol eder.	
DeltaCRLIndicatorChecker		Delta SİL üzerindeki Delta SİL Belirteci(*DeltaCRLIndicator*) eklentisinin RFC 5280 uyumluluğunu kontol	

	eder.
OCSP CEVABI KONTROLCÜLERİ<policy><validate><ocsp>	
SigningCertificateChecker	OCSP cevabını imzalayan yayıncı sertifikasını doğrular.
OCSPSignatureChecker	OCSP cevabı üzerindeki imzayı yayıncı sertifikasının açık anahtarı ile kriptografik olarak doğrular.
ResponseStatusChecker	OCSP cevabı üzerindeki cevap durumu alanının geçerliliğini doğrular.
OCSPResponseDateChecker	OCSP üzerinde yer alan geçerlilik zaman aralığının doğrulama zamanını kapsaması kuralını doğrular.

Table 2 MA3 API Eşleştirici (Matcher) Listesi

SERTİFİKA EŞLEŞTİRİCİLER<policy><match><certificate>	
IssuerSubjectMatcher	Sertifika doğrulama sırasında bulunan ve doğrulanan sertifikaları yerel sertifika deposuna kaydeder.
CertStoreCRLSaver	Sertifika doğrulama sırasında uzaktan(LDAP,http vb.) bulunan SİL'leri yerel sertifika deposuna kaydeder.
SİL EŞLEŞTİRİCİLER<policy><match><certificate>	
CRLDistributionPointMatcher	Sertifika üzerindeki SİL Dağıtım Noktaları (*CRLDistributionPoints*) eklentisinin üzerindeki bilgi ile SİL üzerindeki varsa Yayıncı Dağıtım Noktası (*IssuingDistributionPoin*t) eklentisine bakarak SİL ile sertifikayı eşleştirir.

CRLDistributionPointOnlyContainsMatcher	SİL üzerindeki Yayıncı Dağıtım Noktası *(Issuing Distribution Point*)eklentisindeki onlyContains özellikleri ile Sertifika üzerindeki Temel Kısıtlamalar (*BasicConstraint*) eklentisine bakarak SİL ile sertifikayı eşleştirir
CRLIssuerMatcher	SİL üzerindeki yayıncı adı (*issuer*) alanı ile Sertifika üzerindeki yayıncı acı (*issuer*) alanına bakarak SİL ile sertifikayı eşleştirir.
CRLKeyIDMatcher	SİL üzerindeki Yetkili Anahtar Tanımlayıcısı *(AuthorityKeyIdentifier)* eklentisi ile Sertifika üzerindeki Yetkili Anahtar Tanımlayıcısı *(AuthorityKeyIdentifier* eklentisine bakarak SİL ile sertifikayı eşleştirir.
Delta SİL EŞLEŞTİRİCİLER<policy><match><certificate>	
BaseCRLNumberMatcher	Delta SİL üzerindeki Temel SİL Numarası (*BaseCRLNumber*) eklentisi ile temel SİL üzerindeki SİL Numarası (*CRLNumber*) eklentisine bakarak temel SİL ile delta SİL'i eşleştirir.
CRLNumberMatcher	Delta SİL üzerindeki SİL Numarası (*CRLNumber*) eklentisi üzerindeki SİL numarasının temel SİL üzerindeki SİL Numarası (*CRLNumber*) eklentisi üzerindeki SİL Numarasından büyük olduğunu doğrulayarak temel SİL ile delta SİL'i eşleştirir.
DeltaCRLIssuerMatcher	Delta SİL üzerindeki yayıncı adı (issuer) alanı ile temel SİL üzerindeki yayıncı acı (issuer) alanına bakarak temel SİL ile delta SİL'i eşleştirir.
ScopeMatcher	Delta SİL üzerindeki Yayıncı Dağıtım Noktası (IssuingDistributionPoint) eklentisi ile temel SİL üzerindeki Yayıncı Dağıtım Noktası (IssuingDistributionPoint) eklentisine bakarak temel SİL ile delta SİL'i eşleştirir.

OCSP CEVABI EŞLEŞTİRİCİLER<policy><match><ocsp>	
CertIDOCSPResponseMatcher	OCSP cevabı üzerindeki Sertifika Tanımlayıcısı (CertID) alanına bakarak sertifika ile OCSP cevabını eşleştirir.
ÇAPRAZ SERTİFİKA EŞLEŞTİRİCİLER<policy><match><crosscertificate>	
PublicKeyMatcher	Yayıncı sertifikasındaki açık anahtar ile Çapraz sertifika üzerindeki açık anahtarı eşleştirir.
SKIMatcher	Yayıncı sertifikasındaki Özne Anahtar Tanımlayıcısı(SubjectKeyIdentifier) ile Çapraz sertifika üzerindeki Özne Anahtar Tanımlayıcısını(SubjectKeyIdentifier) eşleştirir.
SubjectMatcher	Yayıncı sertifikasındaki özne adı (subject) alanı ile Çapraz sertifika üzerindeki özne adı (subject) alanını eşleştirir.

Table 3 MA3 API Bulucu (Finder) listesi

GÜVENİLİR SERTİFİKA BULUCULAR<policy><find><trustedcertificate>

TrustedCertificateFinderFromECertStore		Yerel sertifika deposundaki yayıncı sertifikalarını getirir ve bu sertifikaları güvenilir kabul eder.	
	Parametreler	`securitylevel`	`[PERSONAL*, ORGANIZATIONAL, LEGAL]` Depodan getirilecek sertifikaların güven seviyesini belirler. Kişisel sertifikalar için PERSONAL kurumsal sertifikalar için ORGANIZATIONAL kanuni sertifikalar için LEGAL olarak belirtilmelidir.
		`storepath [O]`	Yerel sertifika deposunun dosya sistemindeki yerini belirler.
TrustedCertificateFinderFromFileSystem		Dosya sisteminde verilen dizin adresindeki yayıncı sertifikalarını getirir. Bu sertifikaları güvenilir kabul eder.	
	Parametreler	dizin	Dizin adresi
TrustedCertificateFinderFromXml		Verilen URL adresindeki yayıncı sertifikalarını getirir. Bu sertifikaları güvenilir kabul eder.	
	Parametreler	`storepath [O]`	URL adresi

SERTİFİKA BULUCULAR<policy><find><certificate>			
CertificateFinderFromECertStore		Yerel sertifika deposundaki sertifikaları getirir.	
	Parametreler	`storepath [0]`	Yerel sertifika deposunun dosya sistemindeki yerini belirler.
CertificateFinderFromFile		Dosya sisteminde verilen dosya adresindeki sertifikaları getirir.	
	Parametreler	`dosyayolu`	Dosya adresi
CertificateFinderFromXml		Verilen URL adresindeki sertifikalarını getirir.	
	Parametreler	`storepath [O]`	URL adresi
CertificateFinderFromHTTP		Sertifika üzerinde yeralan Yetkili Erişim Bilgileri (AuthorityInfoAccess) eklentisindeki HTTP adresinden yayıncı sertifikasını getirir.	
CertificateFinderFromLDAP		Sertifika üzerinde yeralan Yetkili Erişim Bilgileri (AuthorityInfoAccess) eklentisindeki LDAP adresinden yayıncı sertifikasını getirir.	

SİL BULUCULAR<policy><validate><certificate><revocation><find>			
CRLFinderFromECertStore		Yerel sertifika deposundaki SİL'leri getirir.	
	Parametreler	`storepath [0]`	Yerel sertifika deposunun dosya sistemindeki yerini belirler.
		`getactivecrl`	`[true,false*]` Sertifika doğrulamanın kesin bir şekilde yapılması için sertifika doğrulamanın yapıldığı tarihinden sonra yayınlanan SİLkullanılmaktadır. Yalnız yeni yayınlanacak SİL beklenmek istenmeyebilir ve yayında olan SİL kullanılmak istenebilir. Bu durumda `getactivecrl` parametresi için `true`verilmelidir**.**
CRLFinderFromFile		Dosya sisteminde verilen dosya adresindeki SİL'leri getirir.	
	Parametreler	`dosyayolu`	Dosya adresi
CRLFinderFromHTTP		Sertifika üzerinde yeralan SİL Dağıtım Noktaları(CRLDistributionPoints) eklentisindeki HTTP adresinden SİL'i getirir.	
CRLFinderFromLDAP		Sertifika üzerinde yeralan SİL Dağıtım Noktaları(CRLDistributionPoints) eklentisindekiLDAP adresinden yayıncı sertifikasını getirir.	

OCSP CEVABI BULUCULAR<policy><validate><certificate><revocation><find>			
OCSPResponseFinderFromECertStore		Yerel sertifika deposundaki kayıtlı OCSP Cevaplarını getirir.	
	Parametreler	`storepath [0]`	Yerel sertifika deposunun dosya sistemindeki yerini belirler.
OCSPResponseFinderFromAIA		Sertifika üzerinde yeralan Yetkili Erişim Bilgileri (AuthorityInfoAccess) eklentisindeki OCSP adresine OCSP sorgusu yaparak OCSP cevabı getirir.	
DELTA SİL BULUCULAR<find><deltacrl>			
DeltaCRLFinderFromECertStore		Yerel sertifika deposundaki Delta SİL'leri getirir.	
	Parametreler	`storepath [0]`	Yerel sertifika deposunun dosya sistemindeki yerini belirler.
DeltaCRLFinderFromFile		Dosya sisteminde verilen dosya adresindeki Delta SİL'leri getirir.	
	Parametreler	`dosyayolu`	Dosya adresi

ÇAPRAZ SERTİFİKA BULUCULAR<find><crosscertificate>			
CrossCertificateFinderFromECertStore		Yerel sertifika deposundaki çapraz sertifikaları getirir.	
	Parametreler	`storepath [0]`	Yerel sertifika deposunun dosya sistemindeki yerini belirler.
CrossCertificateFinderFromFile		Dosya sisteminde verilen dosya adresindeki çapraz sertifikaları getirir.	
	Parametreler	`dosyayolu`	Dosya adresi

Table 4 MA3 API Kaydedici (Saver) Listesi

KAYDEDİCİLER<policy><save>			
CertStoreCertificateSaver		Sertifika doğrulama sırasında bulunan ve doğrulanan sertifikaları yerel sertifika deposuna kaydeder.	
	Parametreler	`storepath [O]`	Yerel sertifika deposunun dosya sistemindeki yerini belirler
CertStoreCRLSaver		Sertifika doğrulama sırasında uzaktan(LDAP,http vb.) bulunan SİL'leri yerel sertifika deposuna kaydeder.	
	Parametreler	`storepath [O]`	Yerel sertifika deposunun dosya sistemindeki yerini belirler
CertStoreOCSPResponseSaver		Sertifika doğrulama sırasında uzaktan(LDAP,http vb.) bulunan OCSP'leri yerel sertifika	

		deposuna kaydeder.	
	Parametreler	`storepath [O]`	Yerel sertifika deposunun dosya sistemindeki yerini belirler

4. CMS İMZA

4.1 Giriş

Bu doküman MA3 API CMS Signature kütüphanesinin nasıl kullanılacağı hakkında bilgi vermektedir. CMS Signature kütüphanesi ile akıllı kartları yönetebilir, sertifika doğrulayabilir, elektronik imza atabilir ve elektronik imza doğrulayabilirsiniz. Kütüphaneyi kullanabilmeniz için API lisansına ihtiyacınız vardır.

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

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

4.2 Gerekler

MA3 API CMS Signature kütüphanesinin kullanılabilmesi için lisans dosyasına, sertifika doğrulama politika dosyasına, sertifika deposu dosyasına ihtiyacınız vardır.

İmza doğrulama işlemi için ise yukarıdaki dosyalarla birlikte MA3 API kütüphanesi yeterli olacaktır. Kanuni geçerliliği olan nitelikli imzaların atılabilmesi için ise güvenli bir donanım kullanılması zorunluğu vardır. Genel kullanım olarak akıllı kart kullanılmaktadır.

Akıllı karta erişilebilmesi için akıllı kart okuyucusu sürücüsünün ve akıllı kartın sürücüsünün kurulması gerekmektedir. Akıllı kartın üreticisinin sağladığı kart izleme programı ile bilgisayarın karta erişimi kontrol edilebilir.

Hızlı bir başlangıç için hızlı başlangıç bölümüne bakabilirsiniz.

4.3 İmza Tipleri

API, ETSI TS 101 733 dokümanında anlatılan aşağıdaki imza tiplerini desteklemektedir:

1. CAdES-BES (Basic Elektronic Signature-Basit Elektronik İmza)
2. CAdES-EPES (Explicit Policy Based Electronic Signature- Belirlenmiş Politika Temelli Elektronik İmza)
3. CAdES-T (Electronic Signature with Time- Zaman Damgası Eklenmiş Elektronik İmza)
4. CAdES-C (Electronic Signature with Complete Validation Data References-Tüm Doğrulama Verilerinin Referanslarının Eklendiği İmza)

5. CAdES-X-Long (EXtended Long Electronic Signature- Genişletilmiş Uzun Elektronik İmza)
6. CAdES-X-Type 1 (EXtended Electronic Signature with Time Type 1- Genişletilmiş Elektronik İmza Tip 1 Zamanlı)
7. CAdES-X-Type 2 (EXtended Electronic Signature with Time Type 1- Genişletilmiş Elektronik İmza Tip 2 Zamanlı)
8. CAdES-X-Long-Type 1 or Type 2 (EXtended Long Electronic Signature with Time Type 1- Genişletilmiş Uzun Elektronik İmza Tip 1 veya Tip 2 Zamanlı)
9. CAdES-A (Archival Electronic Signature- Arşiv Elektronik İmza)

Yukarıdaki imza tiplerinin detayı için ETSI TS 101 733 dokümanına bakılabilir. Ancak burada da imza tipleri için kısa açıklamalar verilmiştir. Açıklamalarda ETSI TS 101 733 dokümanında yer alan şekillerden faydalanılmıştır.

4.4 İmza Atma İşlemleri

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

4.5 İmzasız Bir Verinin İmzalanması

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

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

Java

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

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

//if the user does not want certificate validation at generating signature,he can
add
//P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to false
//params.put(EParameters.P_VALIDATE_CERTIFICATE_BEFORE_SIGNING, false);

//necessary for certificate validation.By default,certificate validation is done
```

```
params.put(EParameters.P_CERT_VALIDATION_POLICY, getPolicy());

//By default, QC statement is checked,and signature wont be created if it is not a
//qualified certificate.
boolean checkQCStatement = isQualified();

//Get qualified or non-qualified certificate.
ECertificate cert =
SmartCardManager.getInstance().getSignatureCertificate(checkQCStatement);
BaseSigner signer = SmartCardManager.getInstance().getSigner(getPin(), cert);

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

SmartCardManager.getInstance().logout();

byte [] signedDocument = bs.getEncoded();

//write the contentinfo to file
AsnIO.dosyayaz(signedDocument, getTestDataFolder() + "testdata/BES-1.p7s");
```

C#

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

Dictionary<String, Object> params_ = new Dictionary<String, Object>();
//if the user does not want certificate validation at generating signature,he can
add
//P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to false
params_[EParameters.P_VALIDATE_CERTIFICATE_BEFORE_SIGNING] = false;

//necessary for certificate validation.By default,certificate validation is done
params_[EParameters.P_CERT_VALIDATION_POLICY] = getPolicy();

//By default, QC statement is checked,and signature wont be created if it is not a
//qualified certificate.

bool checkQCStatement = isQualified();

//Get qualified or non-qualified certificate.
ECertificate cert =
SmartCardManager.getInstance().getSignatureCertificate(checkQCStatement);

BaseSigner signer = SmartCardManager.getInstance().getSigner(getPin(), cert);
```

```
//add signer
 //Since the specified attributes are mandatory for bes,null is given as parameter
//for optional attributes
try
{
   bs.addSigner(ESignatureType.TYPE_BES, cert, signer, null, params_);
}
catch (CertificateValidationException cve)
{
    Console.WriteLine(cve.getCertStatusInfo().getDetailedMessage());
}

SmartCardManager.getInstance().logout();

byte[] signedDocument = bs.getEncoded();

//write the contentinfo to file
DirectoryInfo di = Directory.CreateDirectory(testDataDic+@"\testVerileri");
            AsnIO.dosyayaz(signedDocument, di.FullName + @"\BES-1.p7s");
```

4.6 İmzalı Bir Veriye İmza Eklenmesi

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

- Paralel İmza Ekleme
- Seri İmza Ekleme

4.6.1 Paralel İmza

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

Java

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

//create parameters necessary for signature creation
HashMap<String, Object> params = new HashMap<String, Object>();
ValidationPolicy policy =  PolicyReader.readValidationPolicy(new
FileInputStream(POLICY_FILE));
params.put(EParameters.P_CERT_VALIDATION_POLICY, policy);
/*necessary for certificate validation.By default,certificate validation is done.But
if the user does not want certificate validation,he can add
P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to false*/

//By default, QC statement is checked, and signature wont be created if it is not a
//qualified certificate.
```

```
boolean checkQCStatement = isQualified();

//Get qualified or non-qualified certificate.
ECertificate cert =
SmartCardManager.getInstance().getSignatureCertificate(checkQCStatement,
!checkQCStatement);
BaseSigner signer = SmartCardManager.getInstance().getSigner(getPin(), cert);

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

//write the contentinfo to file
AsnIO.dosyayaz(bs.getEncoded(),NEW_SIGNATURE_ADDED_FILE);

SmartCardManager.getInstance().logout();
```

C#

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

//create parameters necessary for signature creation
Dictionary<String, Object> params_ = new Dictionary<String, Object>();
ValidationPolicy policy = PolicyReader.readValidationPolicy(new
FileStream(POLICY_FILE, FileMode.Open, FileAccess.Read));
params_[EParameters.P_CERT_VALIDATION_POLICY] = policy;

/*necessary for certificate validation.By default,certificate validation is done.But
if the user does not want certificate validation,he can add
P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to false*/

bool checkQCStatement = isQualified();

//Get qualified or non-qualified certificate.
ECertificate cert =
SmartCardManager.getInstance().getSignatureCertificate(checkQCStatement);

BaseSigner signer = SmartCardManager.getInstance().getSigner(getPin(), cert);

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

//write the contentinfo to file
AsnIO.dosyayaz(bs.getEncoded(), NEW_SIGNATURE_ADDED_FILE);

SmartCardManager.getInstance().logout();
```

4.6.2 Seri İmza

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

Aşağıdaki kod örneğinde ilk imzacıya seri imza eklenmektedir.

Java

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

//create parameters necessary for signature creation
HashMap<String, Object> params = new HashMap<String, Object>();
ValidationPolicy policy =  PolicyReader.readValidationPolicy(new
FileInputStream(POLICY_FILE));
params.put(EParameters.P_CERT_VALIDATION_POLICY, policy);
/*necessary for certificate validation.By default,certificate validation is done.But
if the user does not want certificate validation,he can add
P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to false*/

//By default, QC statement is checked,and signature wont be created if it is not a
//qualified certificate.
boolean checkQCStatement = isQualified();

//Get qualified or non-qualified certificate.
ECertificate cert =
SmartCardManager.getInstance().getSignatureCertificate(checkQCStatement);
BaseSigner signer = SmartCardManager.getInstance().getSigner(getPin(), cert);

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

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

//write the contentinfo to file
AsnIO.dosyayaz(bs.getEncoded(),NEW_SIGNATURE_ADDED_FILE);
SmartCardManager.getInstance().logout();
```

C#

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

//create parameters necessary for signature creation
Dictionary<String, Object> params_ = new Dictionary<String, Object>();
ValidationPolicy policy = PolicyReader.readValidationPolicy(new
FileStream(POLICY_FILE,
FileMode.Open,
FileAccess.Read));
```

```
params_[EParameters.P_CERT_VALIDATION_POLICY] = policy;
/*necessary for certificate validation.By default,certificate validation is done.But
if the user does not want certificate validation,he can add
P_VALIDATE_CERTIFICATE_BEFORE_SIGNING parameter with its value set to false*/

bool checkQCStatement = isQualified();

//Get qualified or non-qualified certificate.
ECertificate cert =
SmartCardManager.getInstance().getSignatureCertificate(checkQCStatement);

BaseSigner signer = SmartCardManager.getInstance().getSigner(getPin(), cert);

Signer firstSigner = bs.getSignerList()[0];

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

//write the contentinfo to file
AsnIO.dosyayaz(bs.getEncoded(), NEW_SIGNATURE_ADDED_FILE);
SmartCardManager.getInstance().logout();
```

4.7 Ayrık İmza

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

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

Java

```
BaseSignedData bs = new BaseSignedData();

File file = new File(MOVIE_FILE);
ISignable signable = new SignableFile(file,2048);
bs.addContent(signable,false);

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

ValidationPolicy policy =  PolicyReader.readValidationPolicy(new
FileInputStream(POLICY_FILE));
params.put(EParameters.P_CERT_VALIDATION_POLICY, policy);

//By default, QC statement is checked,and signature wont be created if it is not a
//qualified certificate.
boolean checkQCStatement = isQualified();

//Get qualified or non-qualified certificate.
ECertificate cert =
```

```
SmartCardManager.getInstance().getSignatureCertificate(checkQCStatement);
BaseSigner signer = SmartCardManager.getInstance().getSigner(getPin(), cert);

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

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


SmartCardManager.getInstance().logout();
```

C#

```
BaseSignedData bs = new BaseSignedData();

FileInfo file = new FileInfo(MOVIE_FILE);
ISignable signable = new SignableFile(file, 2048);
bs.addContent(signable, false);

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

ValidationPolicy policy = PolicyReader.readValidationPolicy(new
FileStream(POLICY_FILE,
FileMode.Open,
FileAccess.Read));
params_[EParameters.P_CERT_VALIDATION_POLICY] = policy;

bool checkQCStatement = isQualified();

//Get qualified or non-qualified certificate.
ECertificate cert =
SmartCardManager.getInstance().getSignatureCertificate(checkQCStatement);

BaseSigner signer = SmartCardManager.getInstance().getSigner(getPin(), cert);

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

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

SmartCardManager.getInstance().logout();
```

4.8 Farklı İmza Tiplerinin Oluşturulması

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

4.8.1 BES

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

Java

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

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

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

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

C#

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

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

Dictionary<String, Object> params_ = new Dictionary<String, Object>();
params_[EParameters.P_CERT_VALIDATION_POLICY] = VALIDATION_POLICY;

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

İmzanın beyan edilen zamanını almak için aşağıdaki örnek kod kullanılabilir.

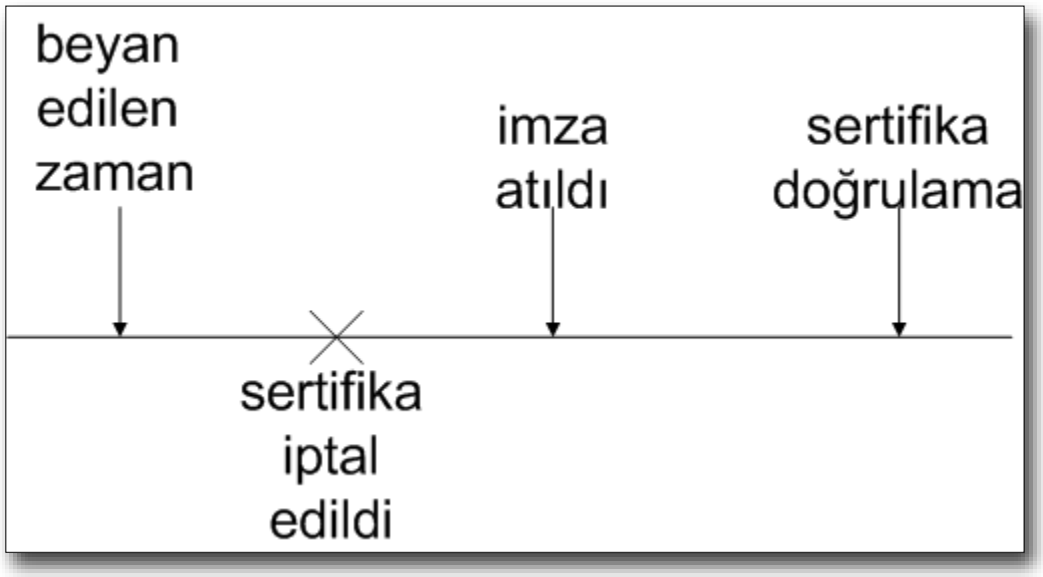
Java

```
byte[] input = AsnIO.dosyadanOKU(BESwithSIGNING_TIME);
BaseSignedData bs = new BaseSignedData(input);

List<EAttribute> attrs =
bs.getSignerList().get(0).getSignedAttribute(SigningTimeAttr.OID);

Calendar time = SigningTimeAttr.toTime(attrs.get(0));
System.out.println(time.getTime().toString());
```

C#

```
byte[] input = AsnIO.dosyadanOKU(BESwithSIGNING_TIME);
BaseSignedData bs = new BaseSignedData(input);

List<EAttribute> attrs = bs.getSignerList()[0].
                         getSignedAttribute(AttributeOIDs.id_signingTime);

DateTime? time = SigningTimeAttr.toTime(attrs[0]);
Console.WriteLine(time.Value.ToString());
```

Beyan edilen zamanın kullanımında oluşabilecek kötü senaryo Şekil 17'daki gibidir. İmza atıldığı sırada imzacı sertifikası iptal edilmiştir; yalnız kullanıcı imza zamanı olarak daha önceki bir zamanı beyan etmiştir. Beyan edilen zamana güvenildiği durumda geçersiz olan bu imza doğrulanacaktır.



Şekil 17: Yanlış Beyan İle Kötü Kullanım Senaryosu

4.8.2 EST

ESTimza, BESimzadan türemiştir. İçerisinde imzalama zamanını gösterir zaman damgası bulundurmaktadır. İmzaatılırken zaman damgası ayarlarının verilmesi gerekmektedir. Aşağıdaki örnek kodda bir EST imzanın nasıl atılacağını bulabilirsiniz.

Java

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

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

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

//add signer
bs.addSigner(ESignatureType.TYPE_EST, cert, signer, null, params);
```

C#

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

Dictionary<String, Object> params_ = new Dictionary<String, Object>();

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

//add signer
bs.addSigner(ESignatureType.TYPE_EST, cert, signer, null, params_);
```

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

Java

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

C#

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

4.8.3 ESXLong

ESXLong imza aynı zamanda bir EST imzadır. ESXLong, ESTimzadan farklı olarak imza içerisinde sertifika doğrulamada kullanılacak doğrulama verisini içerir. . Bundan dolayı ESXLong imza atılırken veya bir imza ESXLong'a çevrilirken sertifika doğrulama işlemi yapılmak zorundadır. Sertifika ömrü dolduktan sonra doğrulamada kullanılacak veriye ulaşmada sorunlar çıkabilir. ESXLong imza doğrulama verisini içinde barındırdığından bu tür sorunların çıkmasını engeller.EST imza atılırken kullanılan örnek kodun aynısı sadece eklenecek imza türü değiştirilerek kullanılabilir.

4.8.4 ESA

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

ESA'yı kullanmanın bir diğer amacı imza üzerinde değişiklik yapılmasını engellemek olabilir. Eğer seri imza kullanılıyorsa; ESA'ya çevrilen imzanın altındaki imzaların veri yapısında bir değişiklik yapılamaz. Bu değişiklikler imza türünün değiştirilmesi, imzanın silinmesi, yeni imza eklenmesi olabilir. Arşiv zaman damgası v2 özelliği bulunduran ES-A tipindeki imzaya, yeni bir seri imza veya doğrulama verisi eklenemez. Arşiv zaman damgası v3 özelliği bulunduran ES-A tipindeki imzada ise bu eklemeler yapılabilir.

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

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

Java

```
byte [] signatureFile = AsnIO.dosyadanOKU(getTestDataFolder() + "ESXLong-1.p7s");
BaseSignedData bs = new BaseSignedData(signatureFile);

Map<String,Object>parameters = new HashMap<String, Object>();

//Archive time stamp is added to signature, so time stamp settings are needed.
parameters.put(EParameters.P_TSS_INFO, getTSSettings());
parameters.put(EParameters.P_CERT_VALIDATION_POLICY, getPolicy());

bs.getSignerList().get(0).convert(ESignatureType.TYPE_ESA, parameters);

AsnIO.dosyayaz(bs.getEncoded(), getTestDataFolder() + "ESA-2.p7s");
```

C#

```
byte[] content = AsnIO.dosyadanOKU(getTestDataFolder() + "ESXLong-1.p7s");
BaseSignedData bs = new BaseSignedData(content);

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

//Archive time stamp is added to signature, so time stamp settings are needed.
parameters[EParameters.P_TSS_INFO] = getTSSettings();
parameters[EParameters.P_CERT_VALIDATION_POLICY] = getPolicy();

bs.getSignerList()[0].convert(ESignatureType.TYPE_ESA, parameters);

AsnIO.dosyayaz(bs.getEncoded(), getTestDataFolder() + "ESA-2.p7s");
```

4.9 Zorunlu Olmayan Özelliklerin Eklenmesi

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

SigningTimeAttr	Beyan edilen imza zamanını içerir. Ancak beyan edilen bu tarih güvenilir bir tarih olmadığından imzanın bu tarihte atıldığını garanti etmez, bilgi amaçlı kullanılabilir.
SignerLocationAttr	İmzacının adresi hakkında bilgiler içerir. İmzacının ülkesi, şehri ve posta adresi belirtilebilir. Bilgi amaçlı olduğundan bu bilgilerden bazılarının değeri null olabilir.
CommitmentTypeIndicationAttr	İmza amacını belirtmek için kullanılabilir. İmzalanan verinin tarafınızdan oluşturulmuş olduğunu, sadece imzanın içeriğini onayladığınızı vs. belirtebilirsiniz. CommitmentType sınıfında tanımlanmış aşağıdaki değerler verilebilir.
RECEIPT	İmza sahibinin imzalı belgeyi aldığın(bir yerden geliyor ise) belirtmek için kullanılır.
SENDER	İmzalı veriyi gönderenin (imzalı veri bir yere gönderiliyor ise) veriyi gönderen kişi olduğunu belirtmek için kullanılır. Yani imza sahibinin gönderilen verinin içeriğini onayladığı anlamına gelmez sadece bunu ben gönderdim demektir
APPROVAL	İmza sahibinin belgenin içeriğini onayladığını belirtmek için kullanılır.
APPROVAL, DELIVERY	bir mesaj gönderildiğinde, bu mesajın karşı tarafa iletildiğini belirtmede kullanılır. Bu tür bir imza genelde güvenilir servis sağlayıcılar (TSP - Trusted Service Provider) tarafından kullanılır.
CREATION	imza sahibinin belgeyi oluşturan kişi olduğunu belirtmek için

	kullanılır. Belge içeriğini onayladığı veya gönderdiği anlamına gelmez.
ORIGIN	imza sahibinin belgeyi oluşturduğunu, içeriğini onayladığını ve gönderenin de kendisi olduğunu belirtmek için kullanılır
ContentIdentifierAttr	imzalanan içeriği tanımlamak için kullanılır. Özellikle ayrık imzada imzalanan dökümanı imza ile eşleştrimek için kullanılabilir. `byte []` olacak şekilde herhangi bir değer olabilir.
ContentHintsAttr	imzalanan içerik hakkında alıcıya fikir vermek amacıyla kullanılır.
SignerAttributesAttr	imzalayan kişi hakkında bilgiler içerir. İmzalayanın iddia ettiği özellikleri veya imzalayanın yetki sertifikasını barındırabilir.

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

Java

```
List<IAttribute> optionalAttributes = new ArrayList<IAttribute>();
optionalAttributes.add(new SigningTimeAttr(Calendar.getInstance()));
optionalAttributes.add(new SignerLocationAttr("TURKEY", "KOCAELİ", new
String[]{"TUBITAK UEKAE","GEBZE"}));
optionalAttributes.add(new CommitmentTypeIndicationAttr(CommitmentType.CREATION));
optionalAttributes.add(new ContentIdentifierAttr("PL123456789".getBytes("ASCII")));

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

SmartCardManager.getInstance().logout();

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

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

attrs = aSigner.getAttribute(SignerLocationAttr.OID);
ESignerLocation sl = SignerLocationAttr.toSignerLocation(attrs.get(0));
StringBuilder sb = new StringBuilder();
```

```
for (String address : sl.getPostalAddress())
sb.append(" " + address);
System.out.println("\nCountry: " + sl.getCountry() +
             "\nCity: " + sl.getLocalityName() +"\nAdress: "     + sb);
attrs = aSigner.getAttribute(ContentIdentifierAttr.OID);
byte [] ci = ContentIdentifierAttr.toIdentifier(attrs.get(0));
System.out.println("\n" + Arrays.toString(ci));

attrs = aSigner.getAttribute(CommitmentTypeIndicationAttr.OID);
CommitmentType ct = CommitmentTypeIndicationAttr.toCommitmentType(attrs.get(0));

System.out.println("\n" + ct);
```

C#

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

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

SmartCardManager.getInstance().logout();

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

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

attrs = aSigner.getAttribute(SignerLocationAttr.OID);
ESignerLocation sl = SignerLocationAttr.toSignerLocation(attrs[0]);
StringBuilder sb = new StringBuilder();
foreach (String address in sl.getPostalAddress())
     sb.Append(" " + address);

Console.WriteLine("\nCountry: " + sl.getCountry() +
"\nCity: " + sl.getLocalityName() + "\nAdress: " + sb);

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

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

4.10 Sertifika Doğrulama

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

Eğer sertifika doğrulamasında bir hata çıkarsa `CertificateValidationException` hatası fırlatılmaktadır. Sertifika doğrulama ile ilgili ayrıntılı bilgi için *Sertifika Doğrulama* bölümüne bakabilirsiniz.

4.11 İmza Doğrulama İşlemleri

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

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

Java

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

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

SignedDataValidation sdv = new SignedDataValidation();

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

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

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

C#

```
byte[] signedData = AsnIO.dosyadanOKU(SIGNATURE_FILE);
ValidationPolicy policy = PolicyReader.readValidationPolicy(new
FileStream(POLICY_FILE,
FileMode.Open, FileAccess.Read));

Dictionary<String, Object> params_ = new Dictionary<String, Object>();
params_[EParameters.P_CERT_VALIDATION_POLICY] = policy;

SignedDataValidation sdv = new SignedDataValidation();

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

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

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

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

4.12 İmza Doğrulama Sonucu

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

Java

```
BaseSignedData bs = getBaseSignedData();
SignedDataValidationResult sdvr = getValidationResult();
SignerfirstCounterSigner = bs.getSignerList().get(0).getCounterSigners().get(0);
SignatureValidationResultfirstCounterSignerVR = sdvr.
                                                    getSDValidationResults().get(0).
                                                    getCounterSigValidationResults().
                                                    get(0);
```

C#

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

getCounterSigValidationResults()[0];
```

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

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

4.13 Ön Doğrulama

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

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

*SignatureValidationResult*nesnesinin *getValidationState()*fonksiyonu ile ön doğrulama yapılıp yapılmadığı bilgisi alınabilir.

Java

```
SignedDataValidationResultsdvr = getValidationResult();

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

C#

```
SignedDataValidationResult sdvr = getValidationResult();

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

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

ÇiSDuP için kesinleşme süresi kısa olabilmektedir. Yalnız iptal bilgileri için SİLkullanılıyorsa bu süre uzayabilmektedir.

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

4.14 Ayrık İmzanın Doğrulanması

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

Java

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

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

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

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

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

C#

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

ValidationPolicy policy =
```

```
PolicyReader.readValidationPolicy(new FileStream(POLICY_FILE,
FileMode.Open, FileAccess.Read));
Dictionary<String, Object> params_ = new Dictionary<String, Object>();
params_[EParameters.P_CERT_VALIDATION_POLICY] = policy;
params_[EParameters.P_EXTERNAL_CONTENT] = content;

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

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

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

4.14.1 Ayrık İmzanın Bütünleşik İmzaya Çevrilmesi

İmzalanan içerik attachExternalContent fonksiyonuna verilerek, ayrık imza bütünleşik imzaya dönüştürülebilir.

Java

```
byte[] input = AsnIO.dosyadanOKU(AYRIK_IMZA);
BaseSignedData bs = new BaseSignedData(input);

File file = new File(IMZALANAN_ICERIK);
ISignable signable = new SignableFile(file,2048);
bs.attachExternalContent(signable);
```

C#

```
byte[] input = AsnIO.dosyadanOKU(AYRIK_IMZA);
BaseSignedData bs = new BaseSignedData(input);

FileInfo file = new FileInfo(IMZALANAN_ICERIK);
ISignable signable = new SignableFile(file, 2048);
bs. attachExternalContent(signable);
```

4.15 Sertifika Doğrulama

İmza atarken ve imza doğrulama sırasında CMS Signature kütüphanesi imzacıların sertifikalarını doğrular. Sertifika doğrulama işleminin detayları için "*Sertifika Doğrulama Dökümanı"*na başvurulmalıdır.Bu dökümanda ayrıntılı bir şekilde açıklanan ve sertifikaların doğrulama işlemlerinin nasıl yapılacağını belirtensertifika doğrulama politikası *EParameters.P_CERT_VALIDATION_POLICY* parametresi ile ValidationPolicy nesnesi tipinde verilir.

4.15.1 İmzacıların Alınması

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

İmza işlemlerinde imza atan kişiyi tanımlama işlemi kişinin sertifikası üzerinden yapılmaktadır. Yalnız genel davranış olarak sertifika imza yapısına konur. MA3 API kütüphanesinde de sertifika imza yapısına eklenmektedir. Sertifikadan, kişinin ismi ve Türkiye için T.C. kimlik numarası alınabilir.

Java

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

if(cert == null){

      System.out.println("İmzaci bilgisi yok");
} else {

      System.out.println("İsim & Soyisim: " +
                         cert.getSubject().getCommonNameAttribute());
      System.out.println("TC Kimlik No: " +
                         cert.getSubject().getSerialNumberAttribute());
}
```

C#

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

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

4.16 İmza Tipleri Arasında Dönüşüm

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

BES tipinden EST tipine dönüşüm yapan örnek kod bloğu:

Java

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

BaseSignedData sd = new BaseSignedData(inputBES);

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

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

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

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

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

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

C#

```
byte[] inputBES = AsnIO.dosyadanOKU(BES_SIGNATURE_FILE);
BaseSignedData sd = new BaseSignedData(inputBES);
Dictionary<String, Object> params_ = new Dictionary<String, Object>();

//necessary for getting signaturetimestamp
params_[EParameters.P_TSS_INFO] = getTSSettings();

ValidationPolicy policy = PolicyReader.readValidationPolicy(new
FileStream(POLICY_FILE,
FileMode.Open,
FileAccess.Read));

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

4.17 İmza Zamanının Belirlenmesi

API'de imza zamanının belirlenmesi farklı yöntemlerle gerçekleştirilebilir.

- Zaman damgası
- Signingtime İmza özelliği kullanılarak imza zamanının belirlenmesi
- İmza zamanının dışarıdan verilen zaman parametresine göre belirlenmesi

Bu yöntemlerden en güvenli ve geçerli olanı Zaman damgası yöntemidir.

4.18 Zaman Damgası

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

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

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

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

Java

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

C#

```
TSSettings tsSettings = new TSSettings("http://zd.ug.net", 21,
"12345678",,DigestAlg.SHA1);
params_[EParameters.P_TSS_INFO] = tsSettings;
```

Zaman damgası ayarlarından ilki zaman damgası adresi, ikincisi kullanıcı numarası, üçüncüsü kullanıcı şifresi ve dördüncüsü zaman damgasının özet algoritmasıdır.

4.18.1 İmzadaki Zaman Damgasından İmza Zamanı Alınması

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

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

Java

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

C#

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

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

Java

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

C#

```
byte[] input = AsnIO.dosyadanOKU(BESwithSIGNING_TIME);
BaseSignedData bs = new BaseSignedData(input);
List<EAttribute> attrs = bs.getSignerList()[0].
                          getSignedAttribute(AttributeOIDs.id_signingTime);
DateTime? time = SigningTimeAttr.toTime(attrs[0]);
Console.WriteLine(time.Value.ToString());
```

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

Java

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

C#

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

4.18.2 Zaman Damgası Sunucusunun Test Edilmesi

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

Java

```
byte [] sha1Digest = new byte [20];
Random rand = new Random();
rand.nextBytes(sha1Digest);
TSClient tsClient = new TSClient();
TSSettings settings  = new TSSettings("http://zd.ug.net", 21,
```

```
"12345678".toCharArray(),,DigestAlg.SHA1);
tsClient.setDefaultSettings(settings);
System.out.println("Remaining Credit: " +
                                        tsClient.requestRemainingCredit(settings));
ETimeStampResponse response = tsClient.timestamp(sha1Digest, settings);
ESignedData sd = new ESignedData(response.getContentInfo().getContent());
ETSTInfo tstInfo = new ETSTInfo(sd.getEncapsulatedContentInfo().getContent());
System.out.println("Time Stamp Time" + tstInfo.getTime().getTime());
System.out.println("Remaining Credit:" + tsClient.requestRemainingCredit(settings));
```

C#

```
byte[] sha1Digest = new byte[20];
Random rand = new Random();
rand.NextBytes(sha1Digest);
TSClient tsClient = new TSClient();
TSSettings settings = new TSSettings("http://zd.ug.net", 1,
"12345678",,DigestAlg.SHA1);
tsClient.setDefaultSettings(settings);
Console.WriteLine("Remaining Credit: " + tsClient.requestRemainingCredit(settings));
ETimeStampResponse response = tsClient.timestamp(sha1Digest, settings);
ESignedData sd = new ESignedData(response.getContentInfo().getContent());
ETSTInfo tstInfo =new ETSTInfo(sd.getEncapsulatedContentInfo().getContent());
Console.WriteLine("Time Stamp Time" + tstInfo.getTime());
Console.WriteLine("Remaining Credit: " + tsClient.requestRemainingCredit(settings));
```

4.18.3 Zaman Damgası Alma

MA3 API kütüpanesini kullanarak sadece zaman damgası da alabilirsiniz. Bunun için *asn1rt.jar*, *slf4j.jar, ma3api-asn.jar*, *ma3api-common.jar*, ma3api-crypto.jar, *ma3api-crypto-gnuprovider.jar*, *ma3api-infra.jar* dosyalarına ihtiyacınız vardır. Şu anda sadece *SHA-1* özet algoritması desteklenmektedir.

Java

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

C#

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

4.19 Parametreler

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

P_EXTERNAL_CONTENT	Ayrık imzanın doğrulaması yapılacaksa doğrulaması yapılacak veriye işaret eder. Atanacak nesne Isignable tipinde olmalıdır.
P_CONTENT_TYPE	İmzalanan içeriğin tip bilgisine işaret eder. Varsayılan değeri ise'veri' anlamına gelen "1, 2, 840, 113549, 1, 7, 1" nesne belirtegeçidir (object identifier).
P_CERT_VALIDATION_POLICY	Sertifika doğrulamada kullanılacak olan politika bilgisine işaret eder. Eğer *P_VALIDATE_CERTIFICATE_BEFORE_SIGNING* parametresine false atanmışsa politika parametresinin atanmasına gerek yoktur. Diğer her durumda politika bilgisine ihtiyaç duyulur.
P_SIGNING_TIME	Doğrulama sırasında imzanın ne zaman atıldığını belirtmek için kullanılır. Yalnız verilen zaman sadece zaman damgası bulunmayan imzalarda kullanılır. Örneğin BES tipinde bir imza 12.12.2010 tarihinde alındı ve alınma tarihi bir yere kaydedildi. Bu tarih daha sonra imzayı doğrularken kullanılabilir. Yalnız hukuki bir yükümlülüğü yoktur.
P_TRUST_SIGNINGTIMEATTR	İmza doğrulama sırasında kullanılır. İmzaya imzayı atan tarafından eklenen zaman bilgisine olan güveni belirler. *Boolean* tipinde nesne atanır; varsayılan değeri ise true'dur. Eğer kullanıcının eklediği zaman bilgisine güvenilmeyecekse false atanmalıdır.
P_TSS_INFO	İmza atarken zaman damgası (Time Stamp) kullanılacaksa, TSSettings tipinde bir nesne atanmalıdır. EST ve üzeri imza tiplerinde zaman damgası kullanılması mecburidir.
P_POLICY_ID	Birbirinden bağımsız iki taraf imzaların nasıl doğrulanacağı konusunda kendi aralarında çeşitli politikalar belirleyebilirler.İmzanın hangi politika ile doğrulanacağı bilgisini imzalanmış dosyaya eklemeleri gerekmektedir. Yalnız bu özelliğin

	kullanılabilmesi için imza türünün EPES veya daha üzeri bir imza türü olması gerekmektedir. EPES türü imzalarda varsayılandavranış olarak *SignaturePolicyIdentifierAttr* imzaya eklenmektedir.Diğer tür imzalarda kullanılabilmesi için *SignaturePolicyIdentifierAttr* imzaya eklenmesi gerekmektedir. *P_POLICY_ID* parametresi ise kullanılacak politika işaret eder.
P_POLICY_VALUE	*P_POLICY_ID* parametresinde anlatıldığı gibi bir politika belirleneceği zaman kullanılır. Kullanılacak politikanın kendisine işaret eder.
P_POLICY_DIGEST_ALGORITHM	*P_POLICY_ID* parametresinde anlatıldığı gibi bir politika belirleneceği zaman kullanılır. *SignaturePolicyIdentifierAttr* imzalanan bir özellik olduğundan, imzalanması sırasında kullanılacak özet bilgisini işaret eder. Varsayılan değer *SHA-1*'dir.
P_INITIAL_CERTIFICATES	İmzalanın doğrulanmasıiçin gerekli olabilecek sertifikaları işaret eder.
P_INITIAL_CRLS	İmzalanın doğrulanması için gerekli olabilecek SİL'leri işaret eder.
P_INITIAL_OCSP_RESPONSES	İmzalanın doğrulanması için gerekli olabilecekÇisDuP cevaplarına işaret eder.
P_VALIDATE_CERTIFICATE_BEFORE_SIGNING	Varsayılan durumda imza atarken ve imza doğrulaması yaparken sertifika doğrulaması yapılmaktadır. BES ve EST türü imza atarken sertifika doğrulaması, bu parametreye false verilerek es geçilebilir. Yalnız bu kullanım önerilmemektedir.
P_REFERENCE_DIGEST_ALG	*SigningCertificateV1/V2*,*CompleteCertificateReferences*,*CompleteRevocationReferences* özellikleri için kullanılacak özet algoritmasını belirler.
P_GRACE_PERIOD	Sertifika doğrulama verisinin kesinleşmesi için geçecek kesinleşme süresini saniye olarak belirtir. Varsayılan değeri 86400 saniyedir (24 saat).
P_REVOCINFO_PERIOD	Sertifika doğrulama verisinin hangi süre aralığında toplanacağını saniye olarak belirtir. Örneğin bu süre 2 günü işaret ediyorsa; imzanın atıldığı tarihten 2 gün sonrasına kadarki sürede geçerli olan doğrulama verisi sertifika doğrulamada kullanılır. Bu süre en az *P_GRACE_PERIOD* süresi kadar olmalıdır.

	Eğer*P_REVOCINFO_PERIOD* parametresine herhangi bir süre verilmezse, bu süre mümkün olduğunca geniş tutulmaya çalışılacaktır. Bu sürenin geniş tutulması sertifika iptal durumlarının yakalanmasında bir dezavantaj getirmez. Bu aralığın kısa tutulması ise askı durumlarının yakalanma ihtimalini arttırır.
P_FORCE_STRICT_REFERENCE_USE	İmzada sertifika doğrulama verisi veya referansı varsa, sadece bu referansları ve bu referanslara ait verilerin doğrulama için kullanılacağını belirtir. Ayrıca bu kontrol, doğrulama verileri ve sertifika referanslarının sırasını da kontrol eder.
P_VALIDATE_TIMESTAMP_WHILE_SIGNING	Zaman damgası içeren imzalarda, imzalama esnasında alınan zaman damgasının kontrol edilmesini sağlar.

5. XML İMZA

5.1 Giriş

XML İmza, referans ile gösterilen veri ve bir anahtar arasında ilişkiyi kuran xml formatındaki veri yapısıdır. Anahtar nitelikli sertifika aracılığı ile bir kişiye bağlandığında imzanın bir kişiye ait olması sağlanır.

Nitelikli sertifikalar bir akıllı kart aracılığı ile kullanılmaktadır.

İmzanın geçerli olması için sertifikanın da geçerli olması gerekmektedir.

Bu sebepler ile XML imza kavramlarının yanında;

- sertifika doğrulama
- akıllı kart gibi kütüphanelerin konfigürasyon ve kullanım bilgisi gerekir/gerekebilir.

İlgili kütüphanelerin kendi dökümantasyonları bulunmaktadır ve bu döküman içerisinde bu konular derinlemesine incelenmeyecektir.

Kod ve konfigürasyon örnekleri için dağıtım dizininde docs/ klasörüne göz atabilirsiniz.

5.2 XML İmza Çeşitleri

5.2.1 Yapısına Göre

İmza ve imzalanan verinin yapısal özelliklerine göre imzalar 3 kategoriye ayrılır.

Enveloping(Zarflayan): Veri BASE64 yada XML formatında imza içinde yer alır.

Enveloped(Zarflanmış): XML Verisi, içinde imzayı barındırır.

Detached(Ayrık): İmzave imzalanan veri ayrı ayrıdır. Özellkle büyük ebatlı veri imzalarken bu yapıda bir imza kullanılması tavsiye edilir.

5.2.2 İmza Tipleri

BES (Basit Elektronik İmza) : BES, içinde imza zamanına ait bilgi bulundurabilir, ancak imza zamanını ispatlama özelliğinden yoksundur.

EPES (Belirlenmiş Politika Temelli Elektronik İmza) : EPES tipindeki bir elektronik imza, oluşturma ve doğrulama kurallarını belirleyen bir politikaya sahip olarak yaratılmış bir BES tipinde imzadır.

ES-T (Zaman Damgalı Elektronik İmza) : BES veya EPES tipindeki elektronik imzaya zaman damgası eklenerek elde edilen e-imza formatıdır.

ES-C (Doğrulama Verisi Taşıyan Elektronik İmza) : Doğrulama verilerinin referanslarını içeren ES-T formatından türetilen bir e-imzadır.

ES-X (Genişletilmiş Elektronik İmza) : ES-Cformatından türetilen ve sadece doğrulama verilerinin referanslarına veya ES-C'nin tamamına zaman damgası eklenerek elde edilen e-imza formatıdır.

ES-XL (Genişletilmiş Uzun Elektronik İmza) : Doğrulama verisini kendi içinde barındıran e-imza tipidir.

ES-A (Arşiv Elektronik İmza) : Kriptografik metodların zaman içinde koruyucu özelliğini yitirmesine karşı periyodik olarak alınan zaman damgası ile korunan e-imzadır.

5.3 İmza Atma

5.3.1 Detached

docs/samples altında xmlsig.samples.Detached class'ına göz atın.

```
// create context with working dir
Context context = new Context(BASE_DIR);

// create signature according to context,
// with default type (XADES_BES)
XMLSignature signature = new XMLSignature(context);

// add document as reference, but do not embed it
// into the signature (embed=false)
signature.addDocument("./sample.txt", "text/plain", false);

// add certificate to show who signed the document
signature.addKeyInfo(CERTIFICATE);

// now sign it by using private key
signature.sign(PRIVATE_KEY);
```

5.3.2 Enveloping

docs/samples altında xmlsig.samples.Enveloping class'ına göz atın.

```
// create context with working dir
Context context = new Context(BASE_DIR);

// create signature according to context,
// with default type (XADES_BES)
XMLSignature signature = new XMLSignature(context);

// add document as reference, and keep BASE64 version of data
// in an <Object tag, in a way that reference points to
// that <Object
// (embed=true)
signature.addDocument("./sample.txt", "text/plain", true);

// add certificate to show who signed the document
signature.addKeyInfo(CERTIFICATE);

// now sign it by using private key
signature.sign(PRIVATE_KEY);

signature.write(new FileOutputStream(SIGNATURE_FILENAME));
```

5.3.3 Enveloped

docs/samples altında xmlsig.samples. Enveloped classına göz atın.

```
// here is our custom envelope xml
org.w3c.dom.Document envelopeDoc = new Envelope();

// create context with working dir
Context context = new Context(BASE_DIR);

// define where signature belongs to
context.setDocument(envelopeDoc);

// create signature according to context,
// with default type (XADES_BES)
XMLSignature signature = new XMLSignature(context, SignatureType.XAdES_BES, false);

// attach signature to envelope
envelopeDoc.getDocumentElement().appendChild(signature.getElement());

// add document as reference,
signature.addDocument("#data1", "text/xml", false);

// add certificate to show who signed the document
signature.addKeyInfo(CERTIFICATE);
```

```
// now sign it by using private key
signature.sign(PRIVATE_KEY);

// output xml,
// this time we dont use signature.write because we need to write
// whole document instead of signature
Source source = new DOMSource(envelopeDoc);
Transformer transformer = TransformerFactory.newInstance().newTransformer();

// write to file
transformer.transform(source,
new StreamResult(new FileOutputStream(SIGNATURE_FILENAME)));
```

5.4 İmza Doğrulama

Basit bir doğrulama kod örneği. Bu örnekte imza dosyadan yüklenmekte ve imza içerisinde yer alan imzalama sertifikası kullanılarak doğrulanmaktadır.

```
XMLSignature signature = XMLSignature.parse(
new FileDocument(new File(FILE_NAME)),
new Context(BASE_DIR)) ;

// no params, use the certificate in key info
ValidationResult result = signature.verify();
```

Bu örnekte yalnız ana imzanın doğrulaması yapıldığına dikkat edilmelidir. Seri imza doğrulaması da yapan örnek kod için xmlsig.samples.Validation sınıfına göz atabilirsiniz.

5.5 Akıllı Kart İşlemleri

Akıllı kart kütüphanesinin detaylı kullanımı için akıllı kart kullanım kılavuzuna göz atınız.

Akılı kart kullanarak imzalama yapmak için uygun BaseSigner sınıfı oluşturmalısınız. Gizli anahtar kart içinde bulunduğu için imza kütüphanesine parametre oarak verilemez.

tr.gov.tubitak.uekae.esya.api.smartcard.util. SCSignerWithCertSerialNo ve tr.gov.tubitak.uekae. esya.api.smartcard.util. SCSignerWithKeyLabel sınıfları sizin için BaseSigner arayüzünü implement etmektedir.

Örnek kod için xmlsig.samples.SmartCardSigner class'ına bakabilirsiniz:

```
public BaseSigner getCardSigner() throws Exception {
    SmartCard smartCard = new SmartCard(CardType.AKIS);
long sessionID = -1;
long slot = 0;
```

```
    slot = smartCard.getTokenPresentSlotList()[0];

    sessionID = smartCard.openSession(slot);

// use label or serial number to read certificate
    List<byte[]> value = smartCard.readCertificate(sessionID,
"bilen.ogreten#work.netSIGN0");
SMARTCARD_PUBLIC_CERTIFICATE = new ECertificate(value.get(0));

// login now to use private certificate for signing later
    smartCard.login(sessionID, "12345");

// create a signer by using serial number of label
return new SCSignerWithCertSerialNo(smartCard, sessionID, slot,
SMARTCARD_PUBLIC_CERTIFICATE.getSerialNumber().toByteArray(),
SignatureAlg.RSA_SHA256.getName());
}
// sample usage
{
BaseSigner signer = getCardSigner();

    signature.getSignedInfo().setSignatureMethod(SignatureMethod.RSA_SHA256);

// add certificate to show who signed the document
    signature.addKeyInfo(SMARTCARD_PUBLIC_CERTIFICATE);

// now sign it by using smart card
signature.sign(signer);
```

5.6 XML İmza Konfigürasyonu

XML imza kütüphanesinin imza yaratma ve doğrulama ayarlarına erişim için kullandığı konfigurasyon dosyasıdır. Bu konfigürasyon içinde kaynaklara erişim için proxy, zaman damgası, kaynak çözücüleri(Resolver arayüzü), varsayılan algoritmalar ve doğrulama ayarları yer alır.

```
<?xml version="1.0" encoding="UTF-8"?>
<xml-signature-config>
```

XML imza konfigürasyonu, xml-signature-config kök elemanı ile başlar.

5.6.1 Yerelleştirme

Yerelleştirme seçenekleri, locale elementi ile ayarlanır.

```
<locale language="EN" country="EN"/>
```

5.6.2 Proxy Ayarları

```
<http>
<proxy-host></proxy-host>
<proxy-port></proxy-port>
<proxy-username></proxy-username>
<proxy-password></proxy-password>
<basic-authentication-username></basic-authentication-username>
<basic-authentication-password></basic-authentication-password>
<connection-timeout-in-milliseconds>2000</connection-timeout-in-milliseconds>
</http>
```

Eğer proxy üzerinden bağlantı kurulacaksa ilgili ayarlar http elementi ile yapılır.

5.6.3 Kaynak Çözücüler

```
<resolvers>
<resolver class="tr.gov.tubitak.uekae.esya.api.xmlsignature.resolver.IdResolver"/>
<resolver class="tr.gov.tubitak.uekae.esya.api.xmlsignature.resolver.DOMResolver"/>
<resolver class="tr.gov.tubitak.uekae.esya.api.xmlsignature.resolver.HttpResolver"/>
<resolver
class="tr.gov.tubitak.uekae.esya.api.xmlsignature.resolver.XPointerResolver"/>
<resolver class="tr.gov.tubitak.uekae.esya.api.xmlsignature.resolver.FileResolver"/>
</resolvers>
```

Bu alanda imza içideki URL'lerin çözümünde kullanılacak sınıflar yer almaktadır. Standart bir kullanım için buradaki çözücü tanımlamaları yeterlidir. Farklı protokoller eklenmek istenirse IResolver arayüzü gerçeklenip, konfigürasyonda resolvers elementine eklenebilir. Örneğin bir URI'ye karşılık gelen veriyi bir veritabanında tutmak isterseniz, kendi DatabaseResolver sınıfınızı yazabilirsiniz.

```
public interface IResolver
{

boolean isResolvable(String aURI, Context aContext);

    Document resolve(String aURI, Context aContext)
throws IOException;
}
```

5.6.4 Zaman Damgası

```
<timestamp-server>
<host>http://timestamp_server_address</host>
<userid>fill_id_here</userid>
<password>pass</password>
<digest-alg>SHA-1</digest-alg>
</timestamp-server>
```

Userid ve password alanları ESYA tabanlı zaman damgası içindir. Açık bir zaman damgası servisi için bu alanları boş bırakın.

5.6.5 Varsayılan Algoritmalar

```
<algorithms>
<digest-method>http://www.w3.org/2001/04/xmlenc#sha256</digest-method>
</algorithms>
```

5.6.6 Doğrulama Parametreleri

```
<validation>
<!-- default policy for certificate validation -->
<certificate-validation-policy-file>//path/to/certval-policy.xml</certificate-
validation-policy-file>
<!-- grace period is the time that needs to pass to get exact revocation info-->
<grace-period-in-seconds>86400</grace-period-in-seconds>
<!-- how old revocation data should be accepted? -->
<last-revocation-period-in-seconds>172800</last-revocation-period-in-seconds>
<!-- compare resolved policy with the one at policy uri, if indicated -->
<check-policy-uri>false</check-policy-uri>

<!-- loosening below 2 settings will cause warnings instead of validation failure --
>
<!-- referenced validation data must be used for cert validation is set true -->
<force-strict-reference-use>false</force-strict-reference-use>
<!-- validation data must be published after creation ifs set true, requires grace
period for signers -->
<use-validation-data-published-after-creation>false</use-validation-data-published-
after-creation>

<!-- used to provide validation profile information, P1_1 refers to turkish
profile#1 and so on---->
<validation-profile>P1_1</validation-profile>

<validators>
...
</validators>
</validation>
```

certificate-validation-policy-file: Sertifika doğrulamada kullanılacak konfigürasyon

grace-period-in-seconds: Saniye cinsinden grace period. Grace period doğrulama verisinin olgunlaşması için beklenmesi gereken süredir.

last-revocation-period-in-seconds Doğrulama verisinin üretim zamanı ile, doğrulama zamanı arasında ne kadar süre olabileceğini gösterir.

force-strict-reference-use: Referanslarla işaret edilen iptal bilgisi doğrulamada kullanılmalı. Güvenlik açısında normal şartlarda bu değer true olmalıdır.

use-validation-data-published-after-creation:Doğrulama verisinin imza tarihinden sonra üretilmesini şart koş. Default değer true. Sağlıklı ve güevnilir bir imzada doğrulama verisi imza tarihinden sonra yayınlanmış olmalıdır.

validation-profile: Opsiyonel bir parametredir. Kullanıldığında parametre değerinde belirtilen profil bilgisine göre doğrulama yapılır.

- İmza oluşturulurken SignaturePolicyIdentifier özelliği ile eklenmiş profil bilgisi varsa bu profil bilgisine göre doğrulama yapılmayıp parametrede belirtilen profil bilgisine göre doğrulama yapılır.
- Parametrenin alabileceği değerler P1_1 (P1-Anlık-İmza Profili), P2_1 (P2-Kısa Süreli-İmza Profili), P3_1 (P3-Uzun Süreli-İmza Profili) ve P4_1 (P4-Uzun Süreli-İmza Profili)'dir.

5.6.7 Doğrulayıcılar

XML İmza doğrulayıcı sınıflar validation/validators tagları arasında yer almaktadır. Normalde bu kısımları değiştirmeniz gerekmez. Ekstra(custom) doğrulayıcılar geliştirmek isterseniz bu kısma ekleme yapmak için örnek konfigürasyonları inceleyebilirsiniz. Dikkat etmeniz gereken tek nokta profil tanımlarında "inherit-validators-from" attribute'ı aracılığıyla profiller arası ortak doğrulayıcı kullanımı yapılabilmesidir.

5.7 Sertifika Doğrulama

İmza atarken ve imza doğrulama sırasında CMS Signature kütüphanesi imzacıların sertifikalarını doğrular. Sertifikaların doğrulama işlemleri sertifika doğrulama politikasına göre yapılmaktadır.

Sertifika doğrulama ve konfigürasyonu için ilgili API dökümanlarına başvurabilirsiniz.

docs/confid dizini altında çok kullanılacak sertifika doğrulama konfigürasyonları örnek olarak yer almaktadır. Bu konfigürasyonlar şunlardır:

5.8 XAdES Çoklu İmza Atma

Gerçek hayatta olduğu gibi, elektronik dökümanlarda da bir dökümanda birden fazla imzanın yer alması mümkündür.

Bu bölümde yer alan örnekleri docs/samples altında xmlsig.samples.MultipleSignatures class'ında inceleyebilirsiniz.

esya.api.xmlsignature.SignedDocument.java sınıfı çoklu imzaları kolayca yönetmek için yazılmıştır. Bu sınıfı incelemenizde yarar var. Basitçe bu sınıfı kullandığımızda XML formatında bir imza oluşur. Bu format üzerinde görüldüğü gibi aynı döküman üzerinde birden fazla veri ve imza yer alabilir.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ma3:envelope xmlns:ma3="http://uekae.tubitak.gov.tr/xml/signature#"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ma3:data>
<ma3:data-item>...</ma3:data-item>
<ma3:data-item>...</ma3:data-item>
...
<ma3:data>
<ma3:signatures>
<ds:signature>...</ds:signature>
<ds:signature>...</ds:signature>
...
</ma3:signatures>
</ma3:envelope>
```

5.8.1 Paralel İmza

Aynı döküman içindeki birbirinden bağımsız imzalardır.

Aşağıdaki örnekte SignedDocument classını kullanarak önce imzalanacak veri dökümana eklenmekte, daha sonra bu veriyi imzalayan birbirinden bağımsız iki imza yaratılmaktadır. *ParallelEnveloped* sınıfını inceleyebilirsiniz.

```
Context context = new Context(BASE_DIR);

SignedDocument signatures = new SignedDocument(context);

Document doc = Resolver.resolve("./sample.txt", context);
String fragment = signatures.addDocument(doc);

XMLSignature signature1 = signatures.createSignature();

// add document as inner reference
signature1.addDocument("#"+fragment, "text/plain", false);
```

```
// add certificate to show who signed the document
signature1.addKeyInfo(CERTIFICATE);

// now sign it by using private key
signature1.sign(PRIVATE_KEY);

XMLSignature signature2 = signatures.createSignature();

// add document as inner reference
signature2.addDocument("#"+fragment, "text/plain", false);

// add certificate to show who signed the document
signature2.addKeyInfo(CERTIFICATE);

// now sign it by using private key
signature2.sign(PRIVATE_KEY);

// write combined document
signatures.write(new FileOutputStream(FILENAME_PARALLEL_ENVELOPED));
```

Şimdi bir de dışarıdaki veriyi imzalayan iki imzayı örnekleyelim. Örnek Kodları *ParallelDetached* sınıfı içinde bulabilirsiniz.

```
Context context = new Context(BASE_DIR);

SignedDocument signatures = new SignedDocument(context);

XMLSignature signature1 = signatures.createSignature();

// add document as reference, but do not embed it
// into the signature (embed=false)
signature1.addDocument("./sample.txt", "text/plain", false);

// add certificate to show who signed the document
signature1.addKeyInfo(CERTIFICATE);

// now sign it by using private key
signature1.sign(PRIVATE_KEY);

XMLSignature signature2 = signatures.createSignature();

// add document as reference, but do not embed it
// into the signature (embed=false)
signature2.addDocument("./sample.txt", "text/plain", false);

// add certificate to show who signed the document
signature2.addKeyInfo(CERTIFICATE);

// now sign it by using private key
signature2.sign(PRIVATE_KEY);

// write combined document
```

```
signatures.write(new FileOutputStream(FILENAME_PARALLEL_DETACHED));
```

Aradaki tek farkın imzalanacak veriyi, SignedDocument yerine sadece referans olarak XMLsigneture'a eklenmesi olduğunu fark ettiniz mi?

5.8.2 Seri İmza

Veriyi imzalayan imzanın imzalanmasıdır. Buna "counter signature" denir. Örnek Kodları *CounterDetached* sınıfı içinde bulabilirsiniz.

Aşağıdaki örnekte önceden yaratılmış imzanın okunup, seri imzalanması gösterilmektedir.

```
// read previously created signature, you need to run Detached.java first
Document doc = Resolver.resolve(Detached.SIGNATURE_FILENAME, context);
XMLSignature signature = XMLSignature.parse(doc, context);

// create counter signature
XMLSignature counterSignature = signature.createCounterSignature();

// sign
counterSignature.addKeyInfo(CERTIFICATE);
counterSignature.sign(PRIVATE_KEY);

// signature contains itself and counter signature
signature.write(new FileOutputStream(FILENAME_COUNTER_EXISTING));
```

Şimdi daha karışık bir senaryoda iki paralel imzanın olduğu bir dökümana bir de seri imza ekleyelim.

```
Context context = new Context(BASE_DIR);

// read previously created signature
Document signatureFile = Resolver.resolve(FILENAME_PARALLEL_DETACHED, context);
SignedDocument signedDocument = new SignedDocument(signatureFile, context);

// get First signature
XMLSignature signature = signedDocument.getSignature(0);

// create counter signature
XMLSignature counterSignature = signature.createCounterSignature();

// sign
counterSignature.addKeyInfo(CERTIFICATE);
counterSignature.sign(PRIVATE_KEY);
```

```
// signed doc contains both previous signature and now a counter signature
// in first signature
signedDocument.write(new FileOutputStream(FILENAME_PARALLEL_COUNTER));
```

Bir önceki örnekten pek de farklı değil. Paralel imzaların SignedDocument içinde tutulması dışında.

5.8.3 P1: Anlık- İmza Profili

Anlık doğrulama gerektiren güvenlik ihtiyacı düşük seviyede olan uygulamalarda kullanılır. İmza doğrulayıcının eline geçtiği an, imza zamanı sayılır. Gelecekte imzayı tekrar doğrulama ihtiyacı olmayacak senaryolarda tercih edilmelidir.

```
// create signature
XMLSignature signature = new XMLSignature(createContext());
signature.addDocument("./sample.txt", "text/plain", false);
signature.addKeyInfo(CERTIFICATE);

// set time now It is better to use TimeProvider!
signature.setSigningTime(Calendar.getInstance());

signature.sign(PRIVATE_KEY);

signature.write(new FileOutputStream(SIGNATURE_FILENAME_P1));
```

Bu imza profili tanımlanmış bir imza politikasına referans vermemektedir.

5.8.4 P2: Kısa Süreli- İmza Profili

Kısa süreli kullanım ömrü olan imzalarda, ÇiSDuP erişimi bulunmayan ortamlarda tercih edilmelidir. ÇisDuP erişimi olan ortamlarda P4 profili kullanılmalıdır.

```
XMLSignature signature = createSignature();

// set policy info defined and required by profile
signature.setPolicyIdentifier(OID_POLICY_P2,
"Kısa Dönemli ve SİL Kontrollü Güvenli Elektronik İmza Politikası",
"http://www.eimza.gov.tr/EimzaPolitikalari/XML_216792121157511.xml");


signature.sign(PRIVATE_KEY);

// add timestamp
```

```
signature.upgradeToXAdES_T();

signature.write(new FileOutputStream(SIGNATURE_FILENAME_P2));
```

5.8.5 P3: Uzun Süreli - İmza Profili

ÇiSDuP erişimi olmayan ortamlarda kullanılabilir. Aksi durumda P3 profilinin, P4 profiline tercih edilebilecek herhangi bir avantajı yoktur, SİL boyutlaır ve imzadan sonra SİL yayınlanmasını bekme gerekliliği gibi sebeplerle mecbur kalmadıkça tercih edilmemelidir.

```
XMLSignature signature = createSignature();

// set policy info defined and required by profile
signature.setPolicyIdentifier(OID_POLICY_P3,
"Uzun Dönemli ve SİL Kontrollü Güvenli Elektronik İmza Politikası",
"http://www.eimza.gov.tr/EimzaPolitikalari/XML_216792121157521.xml");

signature.sign(PRIVATE_KEY);

signature.upgradeToXAdES_T();

// since P3 requires CRLs as revocation info
// below code should be executed after grace period elapses
signature.upgradeToXAdES_C();
signature.upgradeToXAdES_X1();
signature.upgradeToXAdES_XL();

signature.write(new FileOutputStream(SIGNATURE_FILENAME_P3));
```

5.8.6 P4: Uzun Süreli- İmza Profili

En güvenilir, uzun ömürlü ve sorunsuz imza profilidir. Validasyon verisi imza içinde yer alır.

```
XMLSignature signature = createSignature();

// set policy info defined and required by profile
signature.setPolicyIdentifier(OID_POLICY_P4,
"Uzun Dönemli ve ÇİSDuP Kontrollü Güvenli Elektronik İmza Politikası",
"http://www.eimza.gov.tr/EimzaPolitikalari/XML_216792121157531.xml");

signature.sign(SIGNER);

// upgrade
signature.upgradeToXAdES_T();
signature.upgradeToXAdES_C();
signature.upgradeToXAdES_X1();
```

```
signature.upgradeToXAdES_XL();

signature.write(new FileOutputStream(SIGNATURE_FILENAME_P4));
```

5.8.7 Arşiv İmza

Uzun süreli imza içindeki son zaman damgasını imzalayan sertifikanın ömrü dolmadan önce, yada imza içinde kullanılan algoritmalar güvenilirlik derecelerini yitirdikçe imza üzerine arşiv zaman damgası eklenmelidir.

Arşiv formatına erişmek için X-Long(Profil 4) imza üzerinde

```
signature.upgradeToXAdES_A();
```

metodu;

halihazırda arşiv tipindeki imzaya her arşiv zaman damgası eklenmek istendiğinde ise

```
signature.addArchiveTimeStamp();
```

metodu kullanılır.

6. PAdES İMZA

6.1 Giriş

PAdES(Pdf Advanced Electronic Signatures) PDS ISO 32000-1 standardında nasıl imza atılması gerektiğini açıklayan ve ETSI tarafından TS 102 778 teknik spesifikasyonu döküman seti ile açıklanmış imza standardıdır.

Bu dökümanlar ilk kısmı genel bakış olmak üzere 6 parçadan oluşmaktadır. ESYA PAdES kütüphanesi 3. kısım (BES ve EPES imza), 4. kısım (LTV- uzun dönemde doğrulanabilir imza) bölümlerini desteklemektedir.

Bu kütüphane imza kütüphanesi arayüzleri ile kullanılmakta ve Elektronik imza ortak kütüphanesi tanımları ile;

- ES_BES,
- ES_EPES,
- ES_T,
- ES_XL ve
- ES_A

tiplerinde imza oluşturulabilmektedir.

PAdES kütüphanesi altyapısında iText pdf işleme kütüphanesi kullanmaktadır. Proje bağımlılıklarında iText kütüphanesi ayrıca temin edilmelidir. PAdES kütüphanesi sadece Java platformu için mevcuttur.

6.2 ES_BES İmza Atma

PDF dosyası imza konteyneri olarak okunur. Daha sonra createSignature metodu ile imza yapısı PDF içerinde yaratılır.

```
// pdf dökümanı oku
SignatureContainer pc = SignatureFactory.readContainer(
                        SignatureFormat.PAdES,
                        new FileInputStream("hello.pdf"), new Context());

 // imza ekle
Signature signature = pc.createSignature(SIGNER_CERTIFICATE);
signature.setSigningTime(Calendar.getInstance());
signature.sign(SIGNER);

// dosyaya yaz
pc.write()
```

6.3 İmza Doğrulama

```
// imzalı dosyayı oku
SignatureContainer pc = SignatureFactory.readContainer(
                            SignatureFormat.PAdES,
                            new FileInputStream("signed-est.pdf"), new Context());
// dogrula
ContainerValidationResult cvr = pc.verifyAll();

System.out.println(cvr);
```

6.4 ES_T İmza Atma

```
PAdESContext context = new PAdESContext();
context.setSignWithTimestamp(true);

// pdf'i oku
SignatureContainer pc = SignatureFactory.readContainer(
        SignatureFormat.PAdES,
        new FileInputStream("hello.pdf"), context);

// add signature
Signature signature = pc.createSignature(SIGNER.getSignersCertificate());
signature.setSigningTime(Calendar.getInstance());
signature.sign(SIGNER);

// dosyaya yaz
pc.write(new FileOutputStream("signed-est.pdf"));
```

6.5 LTV (ES_A) İmza Atma

```
PAdESContext context = new PAdESContext();
// ilk imzayı ES_T olarak at
context.setSignWithTimestamp(true);

SignatureContainer pc = SignatureFactory.readContainer(SignatureFormat.PAdES,
        new FileInputStream("hello.pdf"), context);

// imza ekle
Signature signature = pc.createSignature(SIGNER.getSignersCertificate());
signature.setSigningTime(Calendar.getInstance());
signature.sign(SIGNER);

// imzayı LTV tipine upgrade et
signature.upgrade(SignatureType.ES_A);

// dosyaya yaz
pc.write(new FileOutputStream("signed-lta.pdf"));
```

7. ASİC E-İMZA

7.1 Giriş

MA3 ASiC kütüphanesi ile ETSI TS 102 918 standardına uygun şekilde imza paketleri oluşturulabilmektedir. Bu sayede bir ya da daha fazla imza, imzalanan veri(ler) ve doğrulama verileri bir zip dosyası içinde taşınabilmektedir.

7.2 Gerekler

ESYA API ASiC Signature kütüphanesinin kullanılabilmesi için lisans dosyasına, sertifika doğrulama politikası ve sertifika deposu dosyalarına ihtiyacınız vardır. Bunu yanında CAdES ya da XAdES imza atma kütüphanesi atılacak imza türüne göre ihtiyaçlar arasındadır.

İmza doğrulama işlemi için ise yukarıdaki dosyalarla birlikte ESYA kütüphanesi yeterli olacaktır. Kanuni geçerliliği olan nitelikli imzaların atılabilmesi için güvenli bir donanım kullanılması zorunluğu vardır. Genel kullanım olarak akıllı kart kullanılmaktadır. Akıllı karta erişilebilmesi için akıllı kart okuyucusu sürücüsünün ve akıllı kartın sürücüsünün kurulması gerekmektedir. Akıllı kart üreticilerinin sağladığı bir kart izleme programı ile bilgisayarın karta erişimi ve kart içeriği kontrol edilebilir.

7.3 Kavramlar

Paket tipi:

```
public enum PackageType {
    ASiC_S,
    ASiC_E
}
```

Basit (ASIC_S)	Paket içinde bir imza ve veri bulunur.
Extended (ASIC_E)	Paket içinde bir yada birden fazla veri ve bir yada birden fazla imza bulunur. Bir imza birden fazla veriyi imzalamış olabilir.

7.4 Anahtar API Arayüzleri ve Tasarım

SignaturePackageFactory imza paketlerini oluşturmakta kullanılan static metotları barındırmaktadır. *SignaturePackage* sınıfı, içinde *SignatureContainer* ve imzalı veriyi veya verileri barındıran ZIP yapısını temsil eden sınıfdır.

7.5 Kütüphane Kullanımı

7.5.1 Basit Paket Oluşturma

```
Context c = new Context();
SignatureFormat format = SignatureFormat.CAdES; // SignatureFormat.XAdES de
olabilir.

SignaturePackage signaturePackage = SignaturePackageFactory
                                    .createPackage(c, PackageType.ASiC_S, format);

// imzalanacak dosyayı pakete ekle
Signable inPackage = signaturePackage
                     .addData(new SignableFile(dataFile, "text/plain"),
"sample.txt");
SignatureContainer container = signaturePackage.createContainer();
Signature signature = container.createSignature(CERTIFICATE);

// paketteki imzalanacak veriyi imzaya ver(false=veriyi imzanın içine ayrıca ekleme)
signature.addContent(inPackage, false);
signature.sign(SIGNER);

// paketi dosyaya yaz
signaturePackage.write(new FileOutputStream(fileName));
```

7.5.2 Çoklu Paket Oluşturma

Paketi okuyup imza(SignatureContainer) ekleme:

```
// read package from file
SignaturePackage sp = SignaturePackageFactory.readPackage(new Context(), inputFile);

// create new container in package
SignatureContainer sc = sp.createContainer();

// create new signature in container
Signature s = sc.createSignature(CERTIFICATE);


// get signable from package
s.addContent(sp.getDatas().get(0), false);
s.sign(SIGNER);

// write
sp.write(new FileOutputStream(outFileName));
```

7.5.3 Paket Doğrulama

```
// read package from file
SignaturePackage sp = SignaturePackageFactory.readPackage(new Context(), inputFile);

// doğrula
PackageValidationResult pvr = sp.verifyAll();
System.out.println(pvr);
```

7.5.4 İmza Geliştirme

```
// read package from file
Context c = new Context();
SignaturePackage sp = SignaturePackageFactory.readPackage(c, new File(fileName));

// get first signature container
SignatureContainer sc = signaturePackage.getContainers().get(0);

// get first signature in container
Signature signature = sc.getSignatures().get(0);

// upgrade
signature.upgrade(SignatureType.ES_T);

signaturePackage.write(new FileOutputStream(outFileName));
```

Bu dökümanda kodlar okunaklılık açısından kısa tutulmaya çalışılmıştır. Buradaki İmza Profillerine Uygun İmza Atma bölümüne de göz atılmalıdır.

8. CMS ZARF

8.1 Giriş

CMS Envelope yapısı şifreli veri oluşturmak için kullanılan bir yapıdır. Temel olarak iki işlevden oluşmaktadır: Gönderilecek verinin şifrelenmesi ve alınan şifreli verinin çözülmesidir.

CMS Envelope yapısında asimetrik ve simetrik şifreleme islemi birlikte kullanılır. Sifrelenecek olan veri, simetrik anahtarla şifrelenir. Her alıcı için bu simetrik anahtar, asimetrik algoritmalar kullanılarak şifrelenir ve şifreli verinin sonuna eklenir. Dokümanın şifresini çözmek isteyen kişi, asimetrik algoritma ile kendisi için şifrelenmis alanın şifresini çözer. Böylelikle dokümanın şifrelendigi simetrik anahtarı elde etmiş olur. Bu simetrik anahtar ile de dokümanın şifresini çözebilir. Bu işlemler MA3 API CMS Envelope kütüphanesi tarafından yapılır, bu kütüphaneyi kullanan geliştiricinin bu yapılarla ilgilenmesine gerek yoktur.

Bir veriyi bir kullanıcıya şifreli olarak göndermeden önce sertifikasının geçerliliği kontrol edilmelidir. Kullanıcı akıllı kartını çaldırmış ve sertifikasını iptal ettirmiş olabilir.

CMS Envelope modülü dağıtımı yapılan E-İmza kütüphanesi içeriğinde bulunmamaktadır. Kullanım lisansı ile birlikte sadece devlet kurumlarına verilmektedir.

8.2 Veri Şifreleme

Bir veriyi şifrelemek için CmsEnvelopeGenerator veya CmsEnvelopeStreamGenerator sınıfları kullanılabilir. Büyük verilerin şifrelenmesi için CmsEnvelopeStreamGenerator sınıfı kullanılmalıdır. CmsEnvelopeGenerator sınıfı bellek üzerindeki bir veriyi şifrelediğinden, şifreleyebilecegi dosyanın boyutu .Net Framework veya JRE'nin bellek sınırı kadardır.

CMS Zarf yapısının ayarları için EnvelopeConfig sınıfı kullanılmalıdır. Bu sınıf yardımıyla varsayılan anahtar taşıma (*RSA_OAEP_SHA256*) ve anahtar anlaşma (*ECDH_SHA256KDF*) algoritmaları değiştirilebilir.

Bir doküman bir kişiye veya daha fazla kisiye şifrelenebilir. Eğer şifrelenmis dokümanının şifresi çözülebiliyorsa, bu dokümana yeni alıcılar eklenebilir. Sifreli veri göndermek için, verinin gönderilecegi kişi veya kişilerin sertifikalarının elimizde ve geçerli olması gerekmektedir. addRecipients metodu sertifikası geçerli olan alıcının dokümana eklenmesini sağlar.

Java

```
EnvelopeConfig config = new EnvelopeConfig ();
config.setPolicy(TestConstants.getPolicyFile());

CmsEnvelopeStreamGenerator cmsGenerator = new
CmsEnvelopeStreamGenerator(plainInputStream, CipherAlg.AES256_CBC);
cmsGenerator.addRecipients(config,recipientCert);
cmsGenerator.generate(encryptedOutputStream);
```

C#

```
EnvelopeConfig config = new EnvelopeConfig();
config.setPolicy(TestConstants.GetPolicyFile());

CmsEnvelopeStreamGenerator cmsGenerator = new
CmsEnvelopeStreamGenerator(plainInputStream, CipherAlg.AES256_CBC);
cmsGenerator.addRecipients(config,recipientCert);
cmsGenerator.generate(encryptedOutputStream);
```

8.3 Şifreli Verinin Çözülmesi

Şifreli verinin çözülmesi için CmsEnvelopeParser veya CmsEnvelopeStreamParser sınıfı kullanılabilir. Büyük şifreli verilerin çözülmesi için CmsEnvelopeStreamParser sınıfı kullanılmalıdır.

Şifrelenmiş verinin çözülmesi için hangi sertifika için şifrelenmiş ise o sertifika ve o sertifikaya ait şifre çözücünün elimizde olması gerekmektedir. Şifre çözücü olarak akıllı kart veya bellek kullanılabilir.

Aşagıda akıllı kart ile şifreli veriyi çözen örnek kod bloğu verilmiştir.

Java

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
long sessionID = sc.openSession(slot);
sc.login(sessionID, PIN);

ByteArrayInputStream EncryptedInputStream = new
ByteArrayInputStream(encryptedOutputStream.toByteArray());
ByteArrayOutputStream decryptedOutputStream = new ByteArrayOutputStream();

CmsEnvelopeStreamParser cmsParser = new
CmsEnvelopeStreamParser(EncryptedInputStream);
IDecryptorStore decryptor = new SCDecryptor(sc, sessionID);
cmsParser.open(decryptedOutputStream, decryptor);
```

C#

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
long sessionID = sc.openSession(slot);
sc.login(sessionID, PIN);

MemoryStream EncryptedInputStream = new
MemoryStream(encryptedOutputStream.ToArray());
MemoryStream decryptedOutputStream = new MemoryStream();
```

```
CmsEnvelopeStreamParser cmsParser = new
CmsEnvelopeStreamParser(EncryptedInputStream);
IDecryptorStore decryptor = new SCDecryptor(sc, sessionID);
cmsParser.open(decryptedOutputStream, decryptor);
```

8.4 Çözücüler

Kullanılacak çözücülerin IDecryptorStore arayüzünden türemis olmaları gerekmektedir. IDecryptorStore arayüzünün iki metodu bulunmaktadır. getEncryptionCertificates() metodu çözücünün çözebilecegi sertifikaları verir. decrypt(...) metoduna, hangi sertifika ile sifre çözme yapılması isteniyorsa o sertifika ve çözülmek istenen veri parametre olarak verilir; metot çözülmüs veriyi geri döner. Bu arayüze uyacak sekilde kendi çözücünüzü de yazabilirsiniz.

CMS Envelope kütüphanesinde hali hazırda akıllı kartta, bellekte ve Microsoft sertifika deposu üzerinden çözme islemlerini yapan çözücüler bulunmaktadır.

8.4.1 Akıllı Kart Çözücü

Akıllı kart ile çözme işlemleri için SCDecryptor sınıfı kullanılmalıdır. Bu sınıf akıllı kart işlemlerini yapacağı SmartCard nesnesini ve oturum numarasını parametre olarak alır. Verinin akıllı kart ile çözülebilmesi için akıllı karta giriş(login) yapılması gerekmektedir.

Akıllı kart işlemleri hakkında daha geniş bilgi almak için "MA3 API SmartCard Kullanım Kılavuzu"na bakınız. Aşağıda örnek bir çözücünün yaratılışı vardır.

Java

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
long sessionID = sc.openSession(slot);
sc.login(sessionID, PIN);

IDecryptorStore decryptor = new SCDecryptor(sc, sessionID);
```

C#

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
long sessionID = sc.openSession(slot);
sc.login(sessionID, PIN);


IDecryptorStore decryptor = new SCDecryptor(sc, sessionID);
```

8.4.2 Bellekte Çözücü

Eğer sertifika ve özel anahtara ulaşılabiliniyorsa, şifrelenmiş dosyalar bellekte de çözülebilir. Bu islem için MemoryDecryptor sınıfı kullanılmalıdır. Nesne yaratılırken sertifika ve özel anahtar çiftleri parametre olarak geçilir.

Java

```
Pair<ECertificate,PrivateKey> recipient =
                                new Pair <ECertificate,PrivateKey>(cert, privKey);
IDecryptorStore decryptor = new MemoryDecryptor(recipient);
```

C#

```
Pair<ECertificate,IPrivateKey> recipient =
                                 new Pair<ECertificate,IPrivateKey>(cert, privKey);
IDecryptorStore decryptor = new MemoryDecryptor(recipient);
```

8.4.3 Microsoft Sertifika Deposundan Çözücü

Microsoft sertifika deposundaki anahtar kullanılarak şifreli veri çözülür. MSCerStoreDecryptor sınıfı kullanılarak çözme işlemi yapılabilir. Bu sınıf herhangi bir parametreye ihtiyaç duymaz. Yalnız kullanıcının sertifika deposuna erişim hakkı olması gerekmektedir.

8.5 Şifrelenmiş Veriye Yeni Alıcı Ekleme

Şifrelenmiş veriye yeni alıcılar eklenebilir. Bunun için sifreli dokümanın hepsini çözmeye gerek yoktur. Sadece dokümanın şifrelendiği simetrik anahtarı elde edip, yeni alıcı için bu simetrik anahtarı şifreleyip dokümana eklemek gerekmektedir. Simetrik anahtarı elde edebilmek için kütüphaneye bir çözücü verilmelidir. Simetrik anahtarı elde etme işlemi dokümanın çözülmesi gibi uzun sürmez.

Java

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
long session = sc.openSession(slot);
sc.login(session, PIN);

List<byte[]> certs = sc.getEncryptionCertificates(session);
ECertificate cert = new ECertificate(certs.get(0));

//Add first recipient
InputStream plainInputStream = new
ByteArrayInputStream(TestData.plainString.getBytes());
ByteArrayOutputStream envelopeWithOneRecipient = new ByteArrayOutputStream();
```

```
EnvelopeConfig config = new EnvelopeConfig ();
config.setPolicy(TestConstants.getPolicyFile());

CmsEnvelopeStreamGenerator cmsGenerator = new
CmsEnvelopeStreamGenerator(plainInputStream );
cmsGenerator.addRecipients(config,cert);
cmsGenerator.generate(envelopeWithOneRecipient);
//CMS Envelope with one recipient is generated.

//Add a new recipient to envelope
ByteArrayInputStream bais = new
ByteArrayInputStream(envelopeWithOneRecipient.toByteArray());
ByteArrayOutputStream envelopeWithTwoRecipient = new ByteArrayOutputStream();

CmsEnvelopeStreamParser cmsParser = new CmsEnvelopeStreamParser(bais);
IDecryptorStore decryptor = new SCDecryptor(sc, session);
cmsParser.addRecipientInfo(envelopeWithTwoRecipient, decryptor, cert);

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

C#

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
long sessionID = sc.openSession(slot);
sc.login(sessionID, PIN);

List<byte[]> certs = sc.getEncryptionCertificates(sessionID);
ECertificate cert = new ECertificate(certs[0]);

using (Stream plainInputStream = new
MemoryStream(Encoding.ASCII.GetBytes(TestData.plainString)),
envelopeWithOneRecipient = new MemoryStream())
{

   EnvelopeConfig config = new EnvelopeConfig();
   config.setPolicy(TestConstants.GetPolicyFile());

    CmsEnvelopeStreamGenerator cmsGenerator = new
    CmsEnvelopeStreamGenerator(plainInputStream);
    cmsGenerator.addRecipients(config,cert);
    cmsGenerator.generate(envelopeWithOneRecipient);
    //CMS Envelope with one recipient is generated.

      //Add a new recipient to envelope
      using (MemoryStream bais = new
      MemoryStream(((MemoryStream)envelopeWithOneRecipient).ToArray()),
      envelopeWithTwoRecipient = new MemoryStream())
      {
            CmsEnvelopeStreamParser cmsParser = new CmsEnvelopeStreamParser(bais);
            IDecryptorStore decryptor = new SCDecryptor(sc, sessionID);
```

```
            cmsParser.addRecipientInfo(envelopeWithTwoRecipient, decryptor, cert);
            sc.logout(sessionID);
            sc.closeSession(sessionID);
      }
}
```

8.6 Dizin Sisteminden Sertifika Bulma

Şifreli veri oluşturmak için şifreleme yapılacak kişinin şifreleme sertifikasının elde edilmesi gerekmektedir. Bunun için kullanılabilecek yöntemlerden bir tanesi dizin sisteminden e-posta adresiyle kullanıcı sertifikasının bulunmasıdır.

Dizin sistemi işlemleri MA3 API kütüphanesinde MA3 API Infra projesinde bulunmaktadır. Bu proje için " ma3api-infra-....jar"ı edinmeniz gerekmektedir.

Cmsenvelope\example dizinindeki LDAPUtil classı bizim test sistemimizde e-posta adresinden sertifikayı bulan kod örneğini içerir. Kendi sisteminize göre uyarlamanız gerekmektedir.

8.7 Sertifika Doğrulama

Şifreli veri göndereceğimiz kişinin sertifikası kaybolma veya çalınma gibi durumlar yüzünden iptal edilmiş olabilir. Sertifikanın iptal edilmesi durumunda o sertifika için şifreleme yapılmaması gerekmektedir. Şifrelenmiş bir verinin şifresini çözerken ise sertifika dogrulama yapılmasına gerek yoktur.

Sertifikanın geçerliliğini kontrol eden modül "MA3 API CertValidation" modülüdür. Bu modülün kullanılabilmesi için "ma3api-certvalidation-....jar" dosyasına sahip olmanız gerekmektedir.

CMS Zarf kütüphanesi şifreleme işlemini yapmadan önce şifrelemenin yapılacağı sertifikayı doğrular. Sertifika doğrulamanın ana unsurlarından bir tanesi de doğrulama sırasında kullanılacak ögeleri tanımlayan politika dosyasıdır. Örnek bir politika dosyasını API ile birlikte edininiz. Politika dosyasını ihtiyaçlarınıza göre sekillendirmek için 3.7 Politika Dosyası bölümünü inceleyiniz. Politika dosyasını CMS Zarf kütüphanesindeki EnvelopeConfig sınıfını kullanarak verebilirsiniz.

8.8 Lisans Ayarları

LicenseUtil. setLicenseXml(InputStream ...) metodu ile lisans dosyası verilebilir. Eğer bu metot ile bir lisans bilgisi verilmemisse, 'working directory' altında lisans klasörü altında "lisans.xml" dosyası aranır. Lisans dosyasının satış yapılan müsteri dısında baskasının eline geçmemesi için bu dosyanın sunucuda tutulması, istemci yazılımlarının lisansa sunucudan erişmesi tavsiye edilmektedir.

- **Deneme Lisansı ile Çalışma**

MA3 API ile denemeler yapmanız için test lisansı edinebilirsiniz. Lisans için verilen xml dosyasında, test alanının degeri "test" ise deneme lisansına sahipsinizdir. Bu lisans ile yaptıgımız çalısmalarda ancak "common name" alanında "test" metni geçen sertifikalar kullanabilirsiniz. Ayrıca akıllı kart ile yapacağınız işlemlerde birkaç saniye gecikme olacaktır.

8.9 Donanım Tabanlı Rassal Sayı Üretme

API içerisindeki rassal sayı üretecekleri tohum(seed) dediğimiz bir rassal veri kaynağını kullanarak rassal veri üretirler. Tohum verisinin rassallığı nihai üretilmiş verinin rassallığını belirler.

Rastgele üretilmiş veri CMS Envelope API'sinde simetrik şifreleme anahtarı olarak kullanıldığı için kritik öneme sahiptir. Bundan dolayı CMS Envelope API'si kullanılırken donanım tabanlı rassal sayı üretme kaynağı kullanılması önerilmektedir. Donanım olarak rassal sayı üreticisinin güvenli olduğuna dair sertifikası olan akıllı kartlar veya HSM'ler kullanılabilir.

API'ye eklediğiniz rassal sayı kaynağı uygulama sonlanıncaya kadar geçerli olacaktır. Dolayısıyla kaynak ekleme işlemini uygulama yaşam döngüsünün başında ve sadece bir kere yapılması gerekmektedir. Eğer eklediğiniz kaynağın artık kullanılmamasını istiyorsanız; *Crypto.getRandomGenerator().removeAllSeeders()* fonksiyonu ile bütün kaynakları silip; sadece kullanılmasını istediğiniz kaynakları eklemeniz gerekmedir.

8.9.1 Akıllı Kart Tabanlı Rassal Sayı Üretme

Akıllı karttan alınan verinin rassal sayı tohumu olarak kullanılması için *SmartCardSeed* sınıfından yararlanılabilir.

Java

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
SmartCardSeed scSeed = new SmartCardSeed(CardType.AKIS, slot);
Crypto.getRandomGenerator().addSeeder(scSeed);
```

C#

```
SmartCard sc = new SmartCard(CardType.AKIS);
long slot = sc.getSlotList()[0];
SmartCardSeed scSeed = new SmartCardSeed(CardType.AKIS, slot);
Crypto.getRandomGenerator().addSeeder(scSeed);
```

9. AKILLI KART

9.1 Giriş

Kriptografik işlemlerin güvenli bir ortamda yapı lması amacıyla akıllı kartlara ihtiyaç duyulmaktadır. Akı llı kartlar özel anahtarın(private key) dışarıdan erişilmesine izin vermeyerek açık anahtar altyapısı için gerekli güvenliği sağlarlar. Akıllı kart içinde kullanıcının sertifikaları , özel anahtarları ve açık anahtarları bulunmaktadır. Her sertifikanın bir açık anahtarı ve bir özel anahtarı yine kart içinde yer almaktadır. Sertifikalar ve açık anahtarlar kart içinden okunabilmektedir. Özel anahtar ise dışarı kart dışına çıkartılamaz, anahtar ile kart içinde kriptografik işlemler yapılabilir.

MA3 API SmartCard modülü akıllı kart işlemlerine yardımcı olur, PKCS7 yapısında basit imza atabilir.

9.2 Gereksinimler

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

.NET SmartCard API 'si ise "ma3api-smartcard.dll" kütüphanesinin yanısıra bağımlı olduğu "ma3api-asn.dll", "ma3api-common.dll", " ma3api-crypto.dll", "asn1rt.dll", ",ma3api-crypto-bouncyprovider.dll", "ma3api-iaik_wrapper.dll", "log4net.dll" ve "ma3api-pkcs11net.dll" kütüphanelerine ihtiyaç duymaktadır.

9.3 Akıllı Karta Erişim

SmartCard sınıfı akıllı kart ile ilgili işlemlerden sorumlu sınıftır. SmartCard sınfının çalıştırılabilmesi için hangi akıllı kartın kullanıldığının bilinmesi gerekmektedir. Çünkü akıllı kart işlemleri akıllı kartın sürücüsü üzerinden yapı lmaktadır. Java 6 ile java kütüphaneleri kullanılarak kart bilgilerine erişilip hangi kart olduğu belirlenebilmektedir. Java 5 ve .NET'te ise hangi akıllı kartın kullanıldığı bilinmelidir.

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

Java 6 ve C#

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

Eğer görsel bir arayüzün API tarafından gösterilmesini istemiyorsanız; SmartOp sınıfının getCardTerminals() fonksiyonu ile akıllı kart okuyucularının isimlerini alabilirsiniz. Bu isimler ile kartı kullanıcıya seçtirdikten sonra getSlotAndCardType(String terminal) fonksiyonuyla kullanıcının seçtiği kartın slot numarası ve kart tipi alınabilir.

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

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

9.4 Akis Kartlara Erişim

Akis kartlara Java 6 kullanıldığında Akis'in java kütüphanesi kullanılarak komut(APDU) gönderilebilmektedir. Sistemde akis sürücüsü yüklü olmasa bile AkisCIF.x.x.x.jar oldugunda karta erişilmektedir. AkisCIF üzerinden akıllı karta erişmek akıllı kart işlemlerinin süresini dolayısıyla imza süresini kısaltmaktadır. Yalnız AkisCIF üzerinden karta erişildiğinde diğer programlar karta erişememektedir. Yeni bir sürüm akis kart kullanmaya başladığınızda AkisCIF'i de yenilemeniz gerekecektir.

Akıllı karta APDU komutları ile AkisCIF.x.x.x.jar üzerinden erişilmesinden APDUSmartCard sınıfı sorumludur. Örnek bir kullanım aşağıdaki gibidir.

Java

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

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

Yukarıda da belirtildiği gibi AkisCIF arayüzü bütün kartları desteklemeyebilir. AkisCIF desteklendiğinde AkisCIF ile desteklenmediğinde dll ile işlemlerinizi yapmak için aşağıdaki şekilde kullanabilirsiniz.

Java

```
BaseSmartCard bsc = null;
int index = 0;
String [] terminals = SmartOp.getCardTerminals();
String selectedTerminal = terminals[index];
long slot;

if(APDUSmartCard.isSupported(selectedTerminal))
{
	bsc = new APDUSmartCard();
	slot = index + 1;
}
else
{
	Pair<Long, CardType> slotAndCardType =
SmartOp.getSlotAndCardType(selectedTerminal);
	slot = slotAndCardType.getObject1();
	bsc = new P11SmartCard(slotAndCardType.getObject2());
}
bsc.openSession(slot);
```

9.5 Akıllı Kartan Sertifikanın Okunması

Akıllı karttan sertifika SmartCard sınıfının getSignatureCertificates() veyagetEncryptionCertificates() fonksiyonları ile okunabilir. Eğer imzalama işlemi yapılacaksa getSignatureCertificates() fonksiyonu, şifreleme işlemi yapılacaksa getEncryptionCertificates() fonksiyonu kullanılmalıdır.Bu fonksiyonlar sertifikaların kodlanmış hallerini byte [] olarak dönerler. Eğer MA3 APIasn modülünü (ma3api-asn-....jar / ma3api-asn.dll) kullanabiliyorsanız, karttan aldığınız bytedeğerlerini anlamlı hale getirmek için ECertificate sınıfını kullanabilirsiniz.

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

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

Ayrıca ECertificate sınıfının getSubject().getCommonNameAttribute() fonksiyonusertifika sahibinin ismini dönmektedir. Karttaki sertifikaların isim bilgisi hepsi için aynıolacağından, karttaki sertifikaları ayırt etmek amacıyla kullanılamaz. Kimin imzayı attığını göstermek için kullanılabilir.

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

Java

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

C#

```
List<byte[]> certBytes = sc.getSignatureCertificates(session);
foreach (byte[] bs in certBytes)
{
ECertificate cert = new ECertificate(bs);
//cert.isQualifiedCertificate()
Console.WriteLine(cert.getSubject().getCommonNameAttribute());
}
```

Eğer MA3 API asn sınıflarına erişim yoksa, ECertificate yerine java'nın x509Certificate sınıfı kullanılabilir. ECertificate sınıfının isQualifiedCertificate() fonksiyonu yerine aşağıdaki örnek kodda gösterildiği gibi kontrol yapılabilir. Sertifikaları birbirinden ayırt etmek amacıyla x509Certificate sınıfının getSubjectDN().toString() metodu kullanılabilir.

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

Java

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

9.6 Akıllı Karttaki Nesne Adlarının Okunması

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

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

9.7 Akıllı Karta İmzalama - Şifreleme İşlemlerinin Yapılması

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

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

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

Java

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

C#

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

Aşağıdaki örnek kodda anahtar adı ile işlem yapan sınıflar vardır.

Java

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

C#

```
BaseSigner signer = new SCSignerWithKeyLabel(sc, session, slot,
"ahmet.uzun#ug.netSIGN0",
SignatureAlg.RSA_SHA1.getName());
```

9.8 Akıllı Kart Kütüphanesi Konfigürasyonu

Akıllı kart kütüphanesinde konfigürasyon aracılığı ile

- Yeni kart tipi tanımlanabilir
- Kart driver değiştirilebilir
- Tanımlı kart tiplerine kartı tanımaya yarayan yeni ATR değerleri eklenebilir.

Böylece sistemde yeni bir kart(marka/versiyon) kullanılabilir.

Bu işlemler API ile birlikte dağıtılan smartcard-config.xml adındaki konfigürasyon dosyası ile yapılmaktadır. Bu dosyada ilk değerler vardır. Bu değerler API içinde gömülü olduğu için, yeni bir ayar yapılmayacaksa bu dosyayı kullanmaya gerek yoktur.

Eğer konfigürasyonda değişiklik yapılırsa bu konfigürasyon dosyası SmartCardConfigParser aracılığı ile okunmalı ve CardType sınıfı bu konfigürasyondan haberdar edilmelidir.

Örnek Kod: Çalışma dizini içindeki smartcard-config.xml ile akıllı kart konfigürasyonu yapma

```
List<CardTypeConfig> cards = new SmartCardConfigParser().readConfig();
CardType.applyCardTypeConfig(cards);
```

Eğer konfigürasyon dosyası farklı bir isimde yada başka bir lokasyonda ise SmartCardConfigParser sınıfının InputStream nesnesi alan metodu da kullanılabilir:

```
List<CardTypeConfig> cards = new SmartCardConfigParser().readConfig(inputStream);
CardType.applyCardTypeConfig(cards);
```

Konfigürasyon ile Kart Tipi Tanımlama

Aşağıda örnek bir akıllı kart tanımlaması görülmektedir.

```
<card-type name="AKIS">
        <lib name="akisp11"/>
        <atr value="3BBA11008131FE4D55454B41452056312E30AE"/>
        <atr value="3B9F968131FE45806755454B41451112318073B3A180E9"/>
        ...
</card-type>
```

<card-type> elemanı ile kart tipi tanımlanır. <lib> elemanı işletim sistemine özel driver belirtmek için kullanılır.

Eğer 32 ve 64 bit mimariler için farklı driver gerekiyorsa bu ayrım "arch" özelliği ile aşağıdaki gibi belirtilir.

```
    <card-type name="NCIPHER">
        <lib name="cknfast" arch="32"/>
        <lib name="cknfast-64" arch="64"/>
```

ATR Değeri

Kart tipini anlamak için karta özel ATR değerini bilmek gerekmektedir. ATR -Answer To Reset- ifadesinin kısaltmasıdır. Protokol gereği kart resetlendiğinde tanıtıcı bir byte dizisi gönderir. Eğer bu byte dizisi biliniyorsa, ve karta özel ayırdedici bilgi içeriyorsa(historical bytes) kart tipi tespit edilebilir. Bilinen her bir ATR byte dizisi için <card-type> elemanı içine yukarıdaki örnekteki gibi bir <atr> elemanı eklenmelidir.

Java ile ATR değeri hesaplamak için aşağıdaki kod parçasını kullanabilirsiniz. Kod ilk terminaldeki(kart okuyucu) akıllı kartın ATRsini ekrana yazdırmaktadır:

```
Card card = TerminalFactory.getDefault().terminals().list().get(0).connect("*");
ATR atr = card.getATR();
String historicalBytesStr = StringUtil.toString(atr.getHistoricalBytes());
System.out.println("historical bytes >"+historicalBytesStr);
String atrHex = StringUtil.toString(card.getATR().getBytes());
System.out.println("ATR >"+atrHex);
```

9.9 SmartCardManager Sınıfı

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

- Sisteme bir kart takılı ve kartta belirtilen özellikte bir sertifika varsa doğrudan bu kart ve bu sertifika üzerinden işlem yapar.
- Birden fazla kart takılı ise kullanıcıya kart seçtirir. Birden fazla belirtilen özellikte sertifika yüklü ise kullanıcıya sertifika seçtirir.
- Eğer APDU ile karta erişilmek isteniyorsa ve kart APDU ile erişimi destekliyorsa APDU ile karta erişim sağlar. C# tarafında APDU erişimi olmadığından sadece pkcs11 erişimi sağlanmaktadır.
- Aynı kart ile imzalama işlemlerinde sertifikayı ve imzacıyı bellekten çekerek hızlanma sağlar.
- Bir kart ile işlem yaptıktan sonra eğer yeni bir kart takılmışsa veya işlem yapılan kart çıkartılmışsa kart ve sertifika seçme işlemlerini tekrarlar.

Örnek bir kullanım aşağıdaki gibi olabilir.

Java

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

//Get qualified certificate. If more than one qualified certificate, user selects
one of them.
ECertificate cert = scm.getSignatureCertificate(true, false);
//Create signer
BaseSigner signer = scm.getSigner("12345", cert);
/**
 * Create signature
 */
//If not sign again with selected card logout.
scm.logout();
//To select new card and new certificate, call reset.
scm.reset();
```

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

9.10 SmartCard Modülü ile BES Tipi İmza Atılması

PKCS7 yapısı en basit imza yapılarından biridir. PKCS7Signature sınıfı PKCS7 formatında imza atılmasından sorumlu sınıftır. Ayrık imza veya bütünleşik imza atılabilir.

signExternalContent fonksiyonu ile ayrık imza, signInternalContent fonksiyonu ile bütünleşik imza atılabilir.

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

Aşağıdaki örnek kod sadece Java için geçerli olup, PKCS7 yapısının .NET MA3 API tarafında desteği bulunmamaktadır.

Java

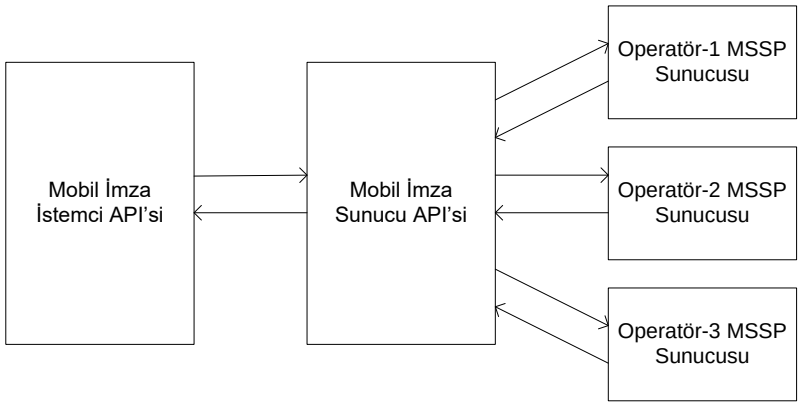
```
PKCS7Signature pkcsSignature = new PKCS7Signature();
ByteArrayOutputStream signature = new ByteArrayOutputStream();

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

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

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

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

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

Java

```
PKCS7 p = new PKCS7(signature);
//validates the signature, not the person.
SignerInfo  [] signerInfo = p.verify(toBeSigned);
if(signerInfo == null)
      return false;
else
{
      //Checks whether the expected person signed the data.
      return signerInfo[0].getCertificateSerialNumber()
                          .equals(cert.getSerialNumber()) == true;
}
```

10. MOBİL İMZA

Mobil mza desteği API'nin 1.4.2 versiyonu ile birlikte gelmektedir. Mobil imzanın kullanılması için istemci ve sunucu yapısının kurulması gerekmektedir. Mobil servis sağlayıcılar (operatörler), mobil imza isteklerini tanımladıkları bir sunucudan almak istemektedirler. Bunun için mobil imza isteklerinin geleceği sunucunun IP'si operatör tarafında tanımlanmalı, istekte bulunacak sunucu kullanıcı adı ve parolaya sahip olmalıdır.



İmza işlemine girecek veri istemci tarafında hesaplanır ve istek mobil imza sunucu API'sine gönderilir. Mobil imza sunucusu API'si MSSP sunucusundan istekte bulunur. MSSP sunucusu isteği imza atmak isteyen kişinin cep telefonuna iletir. Cep telefonundan dönen imzanan veri önce MSSP tarafından mobil imza sunucu API'sine iletilir. Mobil imza sunucu API'si de sonucu mobil imza istemci API'sine iletir ve imza yapısı oluşturulur.

10.1 Mobil İmza İstemci Tarafı

Mobil imza istemci API'si MA3 imzalama API'si paketi içinde infra modülü içerisinde bulunur. İmza oluştururken mobil imza kullanımında tek fark akıllı kart erişim işlemlerinde olmaktadır. Akıllı kart yerine cep telefonu kullanılıyor gibi düşünülebilir. Aşağıdaki örnek kod istemci tarafında mobil imza kullanarak imza atmaktadır. Sadece gri tabanlı işlemler normal imzalama işleminden farklıdır.

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

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

```
MSSPClientConnector connector = null; //Create a communication interface for your
system.
UserIdentifier user = new PhoneNumberAndOperator("05336564727", Operator.TURKCELL);
ECertificate cert = connector.getCertificates(user)[0];
BaseSigner mobileSigner = new MobileSigner(connector, user, cert,
             "Doc1234 numaralı dökümanı imzayı onaylıyorum.",
             Algorithms.SIGNATURE_RSA_SHA1, null);

bs.addSigner(ESignatureType.TYPE_BES, cert , mobileSigner, null, params);
//write the contentinfo to file
AsnIO.dosyayaz(bs.getEncoded(),SIGNATURE_FILE);
```

Sunucu API'si ve istemci API'si arasındaki iletişim için "MSSPClientConnector" arayüzü tanımlanmıştır. Sunucu API'sinin ihtiyacı olan bilgiler bu arayüz üzerinden gönderilecektir. Farklı teknolojilerle bilgiler iletilebileceğinden burada sadece bir arayüz tanımlanmıştır. API kullanıcıları kendi sistemlerine özgü bir iletişim altyapısı kurabilirler.

10.2 Mobil İmza Sunucu Tarafı

Mobil imza API'si kullanılarak yazılacak olan Mobil imza sunucusu MSSP'ye bağlanarak kullanıcı sertifika sorgulaması ve gönderilmiş olan veriyi imzalama işlemi yapar. Bunun için imzalanacak veri, operatör, kullanıcı telefon numarası …vbbilgilerle istemci tarafından çağrılıp (örneğin bir web servisi aracılığı ile)bu bilgilerle MSSP'ye kullanıcı sertifika bilgisi sorgulama ve imzalama isteği göndermelidir.

Bunun için yazılacak olan servisin en az iki adet metodu olmalı ve istemci'ler bu metodları çağırarak işlem yapmalıdırlar.

Servisteki Sertifika sorgulama metodu en az mobil operatör ve telefon numarası almalı ve geriye kullanıcı sertifikası dönmelidir.

Aşağıdaki örnek sertifika alma metodu görülebilir. Metod kullanımına örnek projeden bakılabilir.

```
public String getUserCertificate(String phoneNumber, int iOperator)
{
 Operator mobileOperator = fromInt(iOperator);

 PhoneNumberAndOperator phoneNumberAndOperator =
                        new PhoneNumberAndOperator(phoneNumber, mobileOperator);
 MSSParams mobilParams =
 new MSSParams("http://MImzaTubitakBilgem", "********", "www.turkcelltech.com");
 EMSSPRequestHandler msspRequestHandler = new EMSSPRequestHandler(mobilParams);

 ECertificate[] eCertificates;
   try {
```

```
      eCertificates = msspRequestHandler.getCertificates(phoneNumberAndOperator);

      } catch (Exception e) {
            e.printStackTrace();
            return null;
     }
     if((eCertificates==null)||(eCertificates.length == 0)) {
       Return null;
     }
     ECertificate eCert = eCertificates[0];
     byte[] certBytes = eCert.getEncoded();
         return Base64.encode(certBytes);
     }
```

Servisteki diğer bir metod ise imzalama işlemini yapan metoddur. Bu metod kullanıcının göndermiş olduğu imzalanacak yapısal veriyi imzalamalı ve geri imza değeri dönmelidir.

Örnek olarak imzalanacak verinin Base64 değerini, kullanıcının telefonunda görülecek imzalama mesajını, kullanıcı telefon numarası ve operatör'ü alan örnek servis metodu aşağıdaki gibi olabilir. Metodun detaylı kullanımına örnek projeler içinden bakılabilir.

```
public String SignHash(String hashForSign64, String displayText,
                       String phoneNumber, int iOperator {
 Operator mobileOperator = fromInt(iOperator);
 PhoneNumberAndOperator phoneNumberAndOperator =
                        new PhoneNumberAndOperator(phoneNumber, mobileOperator);
 MSSParams mobilParams =
           new MSSParams("http://MImzaTubitakBilgem", "*******",
"www.turkcelltech.com");
 EMSSPRequestHandler msspRequestHandler = new EMSSPRequestHandler(mobilParams);

 byte[] dataForSign = Base64.decode(hashForSign64);
 byte[] signedData;
 try {
     signedData = msspRequestHandler.sign(dataForSign,SigningMode.SIGNHASH,
     phoneNumberAndOperator, displayText, SignatureAlg.RSA_SHA1.getName(),null);
     } catch (Exception e) {
       e.printStackTrace();
        return null;
      }
        return Base64.encode(signedData);
      }
```

11. ANDROID'DE İMZA ATMA

Android sisteminde temel imza formatında(BES) imza oluşturulabilmektedir. Mevcut sürümde AKİS marka kart ve ACS marka kart okuyucular ile imza oluşturulabilmektedir. İmzalama sırasında kart okuyucunun android kütüphanesi kullanılmaktadır. Mevcut sürümde sadece ACS kart okuyucunun android kütüphanesi kullanılmaktadır. ACS android kütüphanesinin desteklediği kart okuyucular http://android.acs.com.hk adresinden görülebilir. Diğer kart okuyucu markalarının android kütüphaneleri yayınlanınca onlar için de destek sağlanacaktır.

Android sisteminde imza oluşturulurken kart ile iletişim apdu komutları üzerinden sağlanmaktadır.İmzalama işlemleri için APDUSmartCard sınıfı uygun parametrelerle oluşturulmakta ve getSigner() metodunda alınan BaseSigner nesnesi imzalamada kullanılmaktadır. İmzalama işlemlerinde kullanılacak olan APDUSmartCard sınıfı oluşturulurken uygun TerminalHandler sınıfı oluşturulmalı ve bu kullanılmalıdır. Mevcut sürümde ACSTerminalHandler sınfı oluşturulabilmektedir. Kartla yapılan işlemlerde, android sistemi ilk kullanımda karta erişmek için kullanıcıdan usb erişim onayı istemektedir. Bu onay ekranının görülebilmesi ve düzgün çalışması amacıyla ACSTerminalHandler sınıfına usb erişim hakları için oluşturulmuş bir android sınıfı olan PendingIntent nesnesi verilmelidir. Android sistemindeki benzer akışlardan dolayı kart işlemleri doğrudan ana gui sınıfında yapılmamalı, AsyncTask sınıfından türetilen bir sınıf içerisinde işlemler yapılmalıdır.

Takılı olan kart okuyuculardaki sertifikaları listeleyen ve seçilen dosyayı imzalayan örnek bir android uygulaması eclipse projesi paket içerisinde bulunmaktadır. İmzalama için gerekli jar dosyaları bu örnek eclipse projesine bakılarak görülebilir.

Android imzada test lisansı ile çalışırken sadece test sertifikaları ile işlem yapılabilecek ve işlemlerde 5 sn'lik bir gecikme yaşanacaktır.

Yukarıda bahsedilen akışla ilgili örnek bir fonksiyon aşağıdadır. Kod içerisinde gerekli kısımlarda yorumlar bulunmaktadır.

```
public void signWithFirstCertificate() {
try {
    //Burada gömülü lisans dosyası yüklenmektedir.
    Resources res = getResources();
    InputStream lisansStream = res.openRawResource(R.raw.lisans);
    LicenseUtil.setLicenseXml(lisansStream);
    lisansStream.close();
    Activity callerActivity = this;
    //ACSTerminalHandler oluşturulurken bunu çağıran Activity parametre olarak
verilmelidir.
    ACSTerminalHandler acsTerminalHandler =  new ACSTerminalHandler((Activity)this);
    //APDUSmartCard sınıfı uygun TerminalHandler sınıfı ile çağrılmalıdır.
    APDUSmartCard apduSmartCard = new APDUSmartCard(acsTerminalHandler);
```

```
//Kullanıcıdan usb erişim onayı alınabilmesi için oluşturulmuş olan PendingIntent
nesnesi terminal handler sınıfına verilmelidir.
    PendingIntent permissionIntent = PendingIntent.getBroadcast(callerActivity, 0,
new Intent("tr.gov.tubitak.bilgem.esya.android.signexample.USB_PERMISSION"), 0);
    acsTerminalHandler.setPermissionIntent(permissionIntent);
    //Akis kart iletişimi için SecureMessaging devre dışı bırakılmalıdır.
    apduSmartCard.setDisableSecureMessaging(true);
    //Bağlı kart okuyucular okunuyor.
    CardTerminal[] terminalList = apduSmartCard.getTerminalList();
    if(terminalList == null || terminalList.length == 0)
    {
        throw new Exception("Bağlı kart okuyucu sayısı 0");
    }
    CardTerminal cardTerminal = terminalList[0];
    apduSmartCard.openSession(cardTerminal);
    //İlk kart okuyucudan sertifika listesi alınıyor.
    List<byte[]> signCertValueList = mApduSmartCard.getSignatureCertificates();
    if(signCertValueList == null || signCertValueList.size() == 0)
    {
        throw new Exception("Kart içerisinde sertifika sayısı 0");
    }
    //İlk sertifika ile işlem yapılacak.
   ECertificate signingCert = new ECertificate(signCertValueList.get(0));
   String cardPin = "511661";
   apduSmartCard.login(cardPin);
   //İmzalamada kullanılacak BaseSigner APDUSmartCard sınıfından alınıyor.
   BaseSigner signer = apduSmartCard.getSigner(signingCert.asX509Certificate(),
Algorithms.SIGNATURE_RSA_SHA1);
   BaseSignedData bsd = new BaseSignedData();
   //İmzalanacak olan dosya yolu
   String sourceFilePath = "/tmp/TextForSign.txt";
   ISignable content = new SignableFile(new File(sourceFilePath));
   bsd.addContent(content);
   //Since SigningTime attribute is optional,add it to optional attributes list
   List<IAttribute> optionalAttributes = new ArrayList<IAttribute>();
   optionalAttributes.add(new SigningTimeAttr(Calendar.getInstance()));
   HashMap<String, Object> params = new HashMap<String, Object>();
   //Android ile imza atılırken sertifika kontrolü devre dışı bırakılmalıdır.
   //Mevcut sürümde sertifika doğrulama desteği bulunmamaktadır.
   params.put(EParameters.P_VALIDATE_CERTIFICATE_BEFORE_SIGNING,false);
   bsd.addSigner(ESignatureType.TYPE_BES, signingCert, signer, optionalAttributes,
params);
   byte [] signedDocument = bsd.getEncoded();
   String destFilePath = sourceFilePath+ ".imz";
   //İmzalı hali dosyaya yazılıyor.
   AsnIO.dosyayaz(signedDocument, destFilePath);
   apduSmartCard.logout();
   apduSmartCard.closeSession();
}
catch (Exception e) {
    e.printStackTrace();
}
}
```

Ek A. Hızlı Başlangıç

Kütüphaneyi kullanabilmeniz için kütüphane ile birlikte lisans dosyasına, sertifika doğrulama politika dosyasına ve sertifika deposu dosyasına, imza atabilmek için sertifika ve özel anahtara ihtiyacınız vardır.

İndirdiğiniz paket içersinden test lisansı gelmektedir. Lisans dosyasını LicenseUtil.setLicense(...) fonksiyonu ile verebilirsiniz. Lisans dosyalarına 1.4.0 sürümü ile birlikte parola koruması gelmiştir. Lisans dosyanızı kütüphaneye verirken parolanızı da vermeniz gerekmektedir.

Örnek bir politika dosyası paket içersinden çıkmaktadır. Bu politika dosyasını doğrudan kullanmanızda hiçbir sakınca yoktur.

Kullanmanız gereken sertifika deposu dosyası indirdiğiniz paket ile birlikte gelmektedir. Politika dosyasından sertifika deposunun dosya yolunu ayarlayabilir veya dosya yolu ayarını silerek sertifika deposunu “user_home” altında “.sertifikadeposu” klasörü altında “SertifikaDeposu.svt” dosyası olarak kaydedebilirsiniz.

İmza atmanız için sertifika ve özel anahtara ihtiyacınız vardır.Pfx dosyaları sertifikaları ve özel anahtarı şifreli olarak saklayabilmektedir. Test amacıyla kullanılmak üzere bir pfx dosyasını indirdiğiniz paketin içinde bulabilirsiniz. Dosya ismindeki sayılar pfx dosyasının şifresidir. Bu pfx dosyasını bir akıllı karta yükleyebilirsiniz veya bu pfx dosyasını doğrudan kullanabilirsiniz. Pfx içinden sertifikayı nasıl alacağınıza, pfx’den nasıl imzacı oluşturacağınızı görmek için örnekler içindeki *PfxSigner* sınıfına bakabilirsiniz.

Java standart kütüphanesine akıllı kart işlemlerini yapan sınıflar dahil değildir. Çoğu IDE derleme sırasında hata vermektedir. IDE’nizin ayarlarından “restricted API”ler için hata vermesini kapatmanız gerekmektedir.

Ek B. Lisans Ayarları

`LicenseUtil.SetLicenseXml(…)` fonksiyonu ile lisans dosyasını ve lisans dosyasının parolası verilmelidir. Eğer bu fonksiyon ile bir lisans bilgisi verilmemişse, 'working directory' altında lisans klasörü altında *"lisans.xml"* dosyası aranır. Yalnız lisansınızın parolasını `LicenseUtil.setLicensePassword(...)` fonksiyonu ile vermeniz gerekmektedir.

Elinizdeki lisansı windows üzerinde WordPad programı ile görüntüleyebilirsiniz.

1. Deneme Lisansı ile Çalışma

MA3 API ile denemeler yapmanız için test lisansı edinebilirsiniz. Lisans için verilen xml dosyasında, test alınının değeri *"test"* ise deneme lisansına sahipsinizdir.

Bu lisans ile yaptığımız çalışmalarda ancak *"common name"* alanında *"test"* metni geçen sertifikalar kullanabilirsiniz. Ayrıca akıllı kart ile yapacağınız işlemlerde birkaç saniye gecikme olacaktır.

Deneme lisansı ile çalışabilmeniz için indirdiğiniz paket içersinde bulunan pfx dosyasını kullanabilirsiniz. Pfx dosya adında bulunan rakamlar pfx dosyasının şifresidir. Bir pfx yükleyici ile karta test sertifikası ve test özel anahtarınızı aktarabilirsiniz. Eğer pfx yükleyicinin yoksa örnek kodlar içersinden *LoadPfx* sınıfını(sadece Java) kullanabilirsiniz.

2. Lisans Dosyasının Güvenliği

Lisans dosyasının satış yapılan müşteri dışında başkası tarafından kullanılmaması için lisans parolasının kesinlikle başkası ile paylaşılmaması gerekmektedir. Lisans parolası kod üzerinden verildiğinden kodunuzun açılıp lisans parolanızın ele geçmemesi için kodlarınızı karıştırmanızı (obfuscate etmenizi) öneriyoruz. Lisans parolası her kurum/müşteri için sabit olacaktır, ileride lisans dosyanız değiştiğinde aynı parola ile üretileceğinden kodunuzda değişiklik yapmanız gerekmeyecektir.

3. Bakım Sözleşme Bitiş Tarihi

Lisans dosyanızda bakım sözleşme bitiş tarihi ile belirtilen bir alan bulunmaktadır. Bu tarihten daha önce yayınlanan kütüphane sürümlerini kullanmaya devam edebilirsiniz. Bu tarihten sonra yayınlanan kütüphane sürümlerini kullanabilmeniz için bakım sözleşmesi yapmanız gerekecektir.

Ek C. Parola Tabanlı Şifreleme

Bir parola üzerinden verilerinizi şifreleyip aynı parola ile şifrelerinizi çözebilirsiniz. Lisans dosyasının korunması için veya sertifika doğrulama politika dosyanızı korumak için kullanabilirsiniz. Parola tabanlı şifreleme için indirdiğiniz paket içinde yer alan örnek kodların içinde *PasswordBaseCipher* sınıfından yararlanabilirsiniz.

Ek D. Log Tutma

MA3 API JAVA kütüphaneleri log işlemi için slf4j önyüzünü kullanmaktadır. Bu önyüz kendiliğinden bir loglama kütüphanesi sunmamakta fakat farklı alternatiflerin kullanılmasını desteklemektedir.

Log alabilmek için öncelikle slf4j'in desteklediği bir log kütüphanesi kullanılmalıdır. Bunlar jul (java.util.logging) veya log4j olabilmekle birlikte slf4j'in kendi kütüphanesi de kullanılabilir. Eğer jul kullanılacaksa jul için olan bağlayıcı sınıf (slf4j-jdk14-<sürüm>.jar) kullanılmalıdır. Eğer log4j kullanılacaksa log4j için olan bağlayıcı sınıfın (slf4j-log4j12-<sürüm>.jar) yanında log4j kütüphanesi de eklenmelidir. İstenirse bağlayıcı olmadan slf4j'in kendi kütüphanesi de kullanılabilir. Log ayarları ise slf4j önyüzü üzerinden değil, kullanılan kütüphanenin özellik dosyası üzerinden yapılabilir.

Log4j kullanıldığı takdirde `PropertyConfigurator.configure("dosya_ismi")` komutu ile log konfigürasyon dosyası verilebilir.

Java

```
#PropertConfigurator.configure("log4j.properties");
# Set root logger level to DEBUG and its appender to console,rolling,lf5rolling
log4j.rootLogger=debug,rolling
# BEGIN APPENDER: CONSOLE APPENDER (console)
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%p %d{DATE} %c{2} (%F:%M:%L) - %m%n
# END APPENDER: CONSOLE APPENDER (console)
# BEGIN APPENDER: ROLLING FILE APPENDER (rolling)
log4j.appender.rolling=org.apache.log4j.RollingFileAppender
log4j.appender.rolling.File=ESYA_API.log
log4j.appender.rolling.MaxFileSize=50MB
log4j.appender.rolling.MaxBackupIndex=20
log4j.appender.rolling.layout=org.apache.log4j.PatternLayout
log4j.appender.rolling.layout.ConversionPattern=%p %d{DATE} %c{2} (%F:%M:%L) - %m%n
# END APPENDER: ROLLING FILE APPENDER (rolling)
```

Jul için, kullanılan jre'nin içindeki lib klasöründe logging.properties dosyası değiştirilerek log konfigürasyonu yapılabilir.

```
handlers= java.util.logging.ConsoleHandler

.level= FINE

java.util.logging.FileHandler.pattern = %h/java%u.log

java.util.logging.FileHandler.limit = 50000

java.util.logging.FileHandler.count = 1

java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter

java.util.logging.ConsoleHandler.level = FINE

java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

com.xyz.foo.level = FINE
```

MA3 API C# kütüphanesi ise log4net kullanmaktadır. Konfigürasyon dosyası `XmlConfigurator.Configure(new FileInfo("dosya_ismi"))` şeklinde gösterilebilir.

C#

```
<log4net>
<!-- A1 is set to be a ConsoleAppender -->
<appender name="A1" type="log4net.Appender.ConsoleAppender">
<!-- A1 uses PatternLayout -->
<layout type="log4net.Layout.PatternLayout">
<conversionPattern value="%-4timestamp [%thread] %-5level %logger %ndc -
%message%newline" />
</layout>
</appender>
<appender name="FileAppender" type="log4net.Appender.FileAppender">
<file value="ESYA_API.log" />
<appendToFile value="true" />
<lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
<layout type="log4net.Layout.PatternLayout">
<conversionPattern value="%date [%thread] %level %logger %ndc(%F:%L) -
%message%newline" />
</layout>
</appender>
<!-- Set root logger level to DEBUG and its only appender to A1 -->
<root>
<level value="ALL" />
<appender-ref ref="FileAppender" />
</root>
</log4net>
```

Ek E. SÖZLÜK

PKI (AAA)	Açık Anahtar Altyapısını (AAA) ifade eder. Kriptolojinin temel çalışma alanlarından biridir.
MA3	Milli Açık Anahtar Altyapısı'nın kısalması
ESYA	Elektronik Sertifika Yönetim Altyapısı'nın kısaltması
X.509	Bir PKI sertifika ve SİL standardı (Bkz. RFC 5280)
ÇİSDUP	Çevrimiçi Sertifika Durum Protokolü (OCSP). Sertifika iptal durumunun çevrimiçi olarak sorgulanabilmesini sağlayan RFC 2560'ta tanımlı sorgulama protokolünü ifade eder.
SİL	Sertifika İptal Listesi (CRL)
ESHS	Elektronik Sertifika Hizmet Sağlayıcı
Yürürlükteki dizin	Programınızın koştuğu dizin (working directory)
Bağıl adres	Belirli bir adres referans alınarak tanımlanan adres (relative path)