Web Güvenliğinde SSL/TLS Kriptografik Protokolü: Açıklıklar, Saldırılar ve Güvenlik Önlemleri

Bu tez Bilgi Güvenliği Mühendisliği'nde
Tezli Yüksek Lisans Programının bir koşulu olarak

Ahmet ÇAKMAK
tarafından

Fen Bilimleri Enstitüsü'ne
sunulmuştur.

İSTANBUL
ŞEHİR
ÜNİVERSİTESİ

Bu tezi okuduk, kapsam ve nitelik açısından Bilgi Güvenliği Mühendisliği alanında Yüksek Lisans derecesi için tümüyle uygun olduğu görüşüne vardık.

ONAYLAYANLAR:

Prof. Dr. Ensar GÜL .....
(Tez Danışmanı)

Doç. Dr. Mehmet Sabır KİRAZ .....
(Tez Eş-danışmanı)

Prof. Dr. İbrahim SOĞUKPINAR .....

Dr. Öğr. Üyesi Mehmet BAYSAN .....

Bu tez İstanbul Şehir Üniversitesi, Fen Bilimleri Enstitüsü tarafından belirlenen tüm koşullara uygundur.

ONAY TARİHİ:

MÜHÜR/İMZA:

Yazarlık Beyanı

Ben, Ahmet ÇAKMAK, başlığı, "Web Güvenliğinde SSL/TLS Kriptografik Protokolü: Açıklıklar, Saldırılar ve Güvenlik Önlemleri" olan tezin ve içinde sunulan bilgilerin şahsıma ait olduğunu beyan ederim. Ayrıca:

- Bu çalışmanın bütünü veya esası bu üniversitede Yüksek Lisans derecesi elde etmek üzere çalıştığım süre içinde gerçekleştirilmiştir.
- Daha önce bu tezin herhangi bir kısmı başka bir derece veya yeterlik almak üzere bu üniversiteye veya başka bir kuruma sunulduysa bu açık biçimde ifade edilmiştir.
- Başkalarının yayımlanmış çalışmalarına başvurduğum durumlarda bu çalışmalara açık biçimde atıfta bulundum.
- Başkalarının çalışmalarından alıntıladığımda kaynağı her zaman belirttim. Tezin bu alıntılar dışında kalan kısmı tümüyle benim kendi çalışmamdır.
- Esaslı yardım aldığım bütün kaynaklara teşekkür ettim.
- Tezde başkalarıyla birlikte gerçekleştirilen çalışmalar varsa onların katkısını ve kendi yaptıklarımı tam olarak açıkladım.

İmza: 

Tarih: 30.11.2018

"Human ingenuity cannot concoct a cipher which human ingenuity cannot resolve"

Edgar Allan Poe

Web Güvenliğinde SSL/TLS Kriptografik Protokolü: Açıklıklar, Saldırılar ve Güvenlik Önlemleri

Ahmet Çakmak

Öz

Haberleşme teknolojilerinin gelişmesi ile birlikte bilgi güvenliği bilimi önemini artırmaktadır. Internet üzerinden güvenli iletişim için SSL/TLS protokolleri ve açık anahtar altyapısı (PKI) kullanılır. Bu yöntemlerin kullanıcıları için pek çok kolaylık ve fayda sağlamasının yanı sıra tehlikeleri de bulunmaktadır. Haberleşme protokolleri ve güvenli haberleşme için geliştirilen SSL/TLS protokollerinde son 20-25 yıla bakıldığında gerçek anlamda güvenlikten ve gizlilikten bahsetmek pek mümkün değildir. SSL/TLS protokolleri ve açık anahtar altyapısı, zaman içerisinde tehlikeli açıklıklar barındırmış ve hatalarından ders çıkararak daha korunaklı hale gelmişlerdir. Bu tezde, Internet güvenliğinin zaman içerisindeki açıklıklarından bahsedilmiş, güncel tüm kritik saldırılar detaylı olarak anlatılmıştır. Bununla birlikte ataklar incelenirken hem kripto ve açık anahtar altyapısı hem de siber güvenlik yaklaşımları göz önünde bulundurulmuştur. İncelenen saldırıların kapalı ve açık ağlardaki sunucular üzerinde güvenlik analizleri ve tavsiyeleri için bir altyapı geliştirilmiştir.

Anahtar Sözcükler: Kriptografik Protokoller, SSL/TLS, Web Güvenliği, Güvenlik Analizi, Ağ güvenliği, kimlik doğrulama, protokol analizi, https, güvenli soket katmanı, taşıma katmanı güvenliği, x.509, Açık anahtar altyapısı

SSL/TLS Cryptographic Protocol in Web Security: Vulnerabilities, Attacks, and Security Countermeasures

Ahmet Çakmak

Abstract

With the increasing developments about communication technology, information security becomes more important. To ensure security through Internet, SSL/TLS protocols are used alongside with public key infrastructure (PKI). Despite their numerious advantages and benefits to users, these procotols also have dangerous flaws. When inspected within 20 years of lifetime, it is not possible to call SSL/TLS protocols secure. These protocols and PKI have had a pack of major flaws and became more robust by fixing flaws. In this thesis, major weaknesses of these protocols are carefully examined and inspected. While examined these weaknesses, different approaches like cybersecurity and cryptography have been made. An infrastructure for analyzing these attacks for online/offline networks has developed while this survey has being made.

Keywords: Cryptographic Protocols, SSL/TLS, Web Security, web security, authentication, protocol analysis, https, transport layer security, ssl, tls, x.509, public key infrastructure

Teşekkür

Tez çalışmam boyunca her türlü desteği ve motivasyonu sağlayan hocalarım Sn. Doç. Dr. Mehmet Sabır Kiraz ve Sn. Prof. Dr. Ensar Gül'e teşekkürü bir borç bilirim. Bu uzun süreçte sağladıkları katkılardan dolayı MCS ekibine de teşekkür ederim.

İçindekiler

Şekil Listesi

Tablo Listesi

Kısaltmalar

SSL	Güvenli Soket Katmanı (Secure Socket Layer)
TLS	Taşıma Katmanı Güvenliği (Transport Layer Security)
RTT	Tur Devir Süresi (Round Trip Time)
$HTTP/HTTPS$	Güvenli/Köprü Metni Aktarım Protokolü (Secure-/Hypertext Transfer Protocol)
RAM	Rastgele Erişimli Bellek (Random Access Memory)
$BEAST$	SSL/TLS'e Yönelik Tarayıcı Açıklığı (Browser Exploit Against/SSL TLS)
$CRIME$	Sıkıştırma Sıklığı ile Bilgi Sızması Açıklığı (Compression Ratio Info-Leak Exploitation)
$FREAK$	İthal RSA Anahtarlarını Hesaplama (Factoring RSA Export Keys)
$DROWN$	Zayıf ve Geçmişte Kalmış Şifrelemelerle RSA Şifresini Çözmek (Decrypting RSA using Obselete and Weakened eNcryption)
$BREACH$	Köprü Metni Uyarlamalı Sıkıştırmasıyla Tarayıcı Keşif ve Exfiltrasyonu (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext)
$OSCP$	Çevrimiçi Sertifika Durumu Protokolü (Online Certificate Status Protocol)
CSR	Sertifika İmzalama İzni (Certificate Signing Request)
MAC	Mesaj Yetkilendirme Kodu (Message Authentication Code)
DOS	Servis Reddi (Denial Of Service)
CBC	Şifreli Blok Şifreleme (Cipher Block Chaining)
$IETF$	İnternet Mühendisliği Görev Grubu(Internet Engineering Task Force)

Bölüm 1

Giriş

1.1 Internet Güvenliği ve Taşıma Katmanı Güvenliği Protokolü

Teknolojinin gelişimi ile birlikte artan haberleşme imkanları, haberleşme güvenliği konusunu önemli hale getirmiştir. Gerek günlük yaşantımızda, gerekse büyük çapta haberleşmelerde Internet üzerinden haberleşme kullanılmaktadır. Internet teknolojisi küçük bir insan topluluğu tarafından kullanılan bir uygulama olmaktan çıkıp, dünya çapında erişim sağlanan bir yapı haline dönüşmüştür. Elektronik cihazların gelişimi ve ucuzlaması ile ağ bağlantıları yapabilen küçük cihazlar ve sensörler ortaya çıktı. Veri haberleşmesi yalnızca bilgisayarlar arasında kalmadı ve her insanın birden çok cihazı ile aktif olarak iletişim kurduğu bir sistem ortaya çıktı.

Kişisel bilgisayarların yanı sıra sensörlü, gömülü ve uzaktan erişimli pek çok elektronik cihaz veri alışverişinde bulunur hale geldi. Internet üzerinden yapılan iletişimler, aktarılan bilgilerin güvenliği ile ilgili sorunlar ortaya çıkardı. Bankacılık işlemlerinde gizli bilgilerin korunması ve Internet kullanımında ortaya çıkan şahsi verilerin mahremiyeti gibi endişeler oluştu. Yapılan elektronik alışverişler ve bankacılık işlemleri maddi varlıkların korunmasını, kişisel Internet kullanımı da kişisel verilerin mahremiyetinin sağlanması gerekliliğini ortaya çıkardı. Internet üzerinden yapılan hareketler yasal yükümlülükleri ve yaptırımları da beraberinde getirdi. Ancak tüm bu gelişmeler Internet'in ilk geliştiricilerinin tahmin etmedikleri özellikler idi.

İlk Internet yaklaşımının güvenliği içermemesinden ötürü [87], mevcut sistemi güvenli hale getirmek, beraberinde büyük zorluklar da getirdi. Internet büyüyüp yaygınlaştıkça

da güvenlik gereksinimlerinin sağlanması için kapsamlı protokollere ihtiyaç duyuldu. Bununla birlikte güvenliğin insanlar için ne kadar önemli olduğu konusunda son kullanıcıların bilinçlendirilmesi gerekliliği doğdu. Bilgi güvenliği bilimi, kullanıcılarının hem ticari kaygılarını hem de mahremiyetlerini korumaları için farkındalık sağlamaya çalıştı. Çünkü, haberleşme güvenliğinin tek başına bir özellik değil, zaman içinde değişen ve olgunlaşan bir kavram olduğu farkedildi. Bu değişim en çok da cihazların birer hizmet sunan yapılar haline gelmesi ve "Bulut Bilişim" teknolojisinin ortaya çıkışı ile hissedildi. Kullanıcılarına altyapı, platform, yazılım, güvenlik vb. gibi hizmetler sunan bulut teknolojisi [72], yalnızca güvenli haberleşme değil, güvenli depolama ve güvenli işlem yapma gibi gereksinimleri ortaya çıkardı. "Nesnelerin İnterneti" kavramının ortaya yayılması ile de otonom cihazlar, yapay zeka sahibi robotlar ve sınırlı kaynak ile güvenli iletişim sağlamaya çalışan sensörler gibi yeni aktörler ağ güvenliğinde yerini almış oldu [101].

Internet ile güvenli haberleşme yapılması için sağlanması gereken en hassas gereksinim kimlik doğrulamasıdır. Çünkü Internet Protokolü'nde (IP), tasarım olarak bir kimlik doğrulaması bulunmamaktadır ve iletişim kuran taraflar kolaylıkla kandırılabilirler. Bu yüzden hassas verilerin kullanıldığı haberleşmelerde taraflar arasında kimlik doğrulaması sağlanmadan iletişime geçilmemelidir. Örneğin, bankacılık işlemlerinin gerçekleşebilmesi için kullanıcının, sadece kendisinin bildiği veya ulaşabildiği değerler ile kendini doğrulaması, bankanın da kullanıcıya gerçekten banka ile konuştuğunu ispatlaması gerekir. Bunu da sağlamak için açık anahtar altyapısının temeli olan asimetrik kriptoya ihtiyaç vardır. Bununla beraber, veri şifrelerken asimetrik kriptonun yavaş olmasından ötürü, simetrik şifreleme kullanılmaktadır. Bu yöntemlerin kullanılması, güvenliği sağlamak için gerekli olsa da, yeterli olmamaktadır. Kimlik doğrulama ve şifreleme gibi pek çok alanda zaman içerisinde birtakım açıklıklar ortaya çıkmıştır. Bu açıklıkların büyük çoğunluğu, güvenli iletişimi sağlamak için atılan adımlarla ortaya çıkmıştır. Çünkü güvenlik, yaşayan bir organizma gibidir ve olgunluğa ulaşması için zorlu bir süreçten geçmesi gerekir.

Internet güvenliği bir çok bilim dalının dahil olduğu çoklu disiplinli bir yapıdır. Bu yüzden güvenlik ile ilgili araştırma yapabilmek için hem kriptografik temeller ve ağ protokolleri gibi teknik altyapılara, hem de haberleşme ve kullanıcı deneyimi gibi sosyal yaklaşımlara hakim olabilmek gerekir. Çünkü, bu yapılar içerisinde aşağıdaki gibi pek çok tehdit bulunmaktadır:

- **Kriptografik Tasarım Açıklıkları:** Bilgi güvenliğinin sağlanması güvenli protokollerin inşasına, güvenli protokollerin inşası da kriptografik algoritmaların güvenliğine dayanmaktadır. Ancak, kötü niyetli bir tasarımcı tarafından, kriptografik tasarımlarda kullanılan gizli değerler aslında başka bir kişi ya da kurum tarafından kolayca elde edilebilen değerler olarak tasarlanabilir. Bu açıklıklara kleptografi adı verilir [102].

- **Donanım Tasarımında Arka Kapı:** Ağ üzerinde kullanılan cihazlar, kullanan kişilerden habersiz olarak bilgi sızdırabilecek şekilde tasarlanabilir. Donanımın güvenli görünmesine rağmen, üzerinde bir arka kapı bulunabilir.
- **Yazılım Geliştirmede Arka Kapı:** Gömülü yazılımlar veya diğer uygulamalar üzerlerinde bilgi sızdırabilecek bir şekilde tasarlanabilir. Buna yazılımsal arka kapı adı verilir.
- **Güvenli İletişim Katmanı Açıklıkları:** Uygulama katmanlarında (TCP/IP) güvenli iletişimi sağlamak ve tehditlere karşı önlem alabilmek için geliştirilen standartlar bulunmaktadır (Örn: SSL/TLS) [25]. Bu protokollerin güvenlik sağlamayı hedefliyor olmalarına karşın, bilerek veya bilmeyerek yapılan hatalar ile güvenlik ve mahremiyet açıklıkları ortaya çıkabilmektedir.
- **Montaj ve Posta Aşamasında Sahtekarlık:** Verilerin düzenlenmesi veya hareket halindeki durumlarında bilgi sızdırma, değiştirme gibi işlemlere müsade edecek açık kapılar bulunabilir.
- **Konfigürasyon Aşamasında Olası Tehlikeler:** Programların veya verilerin düzenlenmesinde ve kurulumunda ortaya çıkan problemler de veri gizliliğini, bütünlüğünü vs. ihlal edebilir.
- **Cihaz Güncellemede Güvenlik Açıklıkları:** Başlangıçta güvenli olan bir cihaz, üzerindeki yazılımın güncellenmesi ile arka kapı bulundurur hale gelebilir.

Yukarıda bahsedilen maddeler genel olarak haberleşme güvenliğinin sorunlarını ifade etmektedir. Görüldüğü gibi pek çok şekilde güvenlik ihlali mümkündür ve her birisi için farklı çalışmalar ortaya çıkarılabilir. Bu tezde bizim yoğunlaştığımız kısım, güvenli iletişim katmanında oluşan tehlikelerden ve bunların teknik analizlerinden oluşmaktadır.

1.2 Tezin Hedefi ve Motivasyonu

Tezin konusu, her gün milyarlarca insanın farkında olarak veya olmadan kullandığı SSL/TLS protokollerinin ve buna bağlı olan açık anahtar altyapısının zaman içerisinde yaşadığı sorunları detaylı olarak incelenmesidir. Bu incelemeler ile ortaya çıkan analizler ve sonuçlar aktarılmış, bu konularda çalışma yapan araştırmacılar için kapsamlı bir kaynak ortaya çıkarılmıştır.

1.2.1 Hedef

SSL/TLS mimarisinin güvenlik, gizlilik, veri bütünlüğü, kimlik doğrulama vb. pek çok konuda kullanıcılara güvenli iletişim imkanı verdiği düşünülebilir. Fakat ne yazık ki pratik yapıya geçildiğinde bir çok problemin mevcut olduğu görülmektedir. SSL/TLS'in geçmişine bakıldığında pek çok kritik açıklıklar ortaya çıkmıştır. Bu açıklıklar SSL/TLS'in kurgulanan algoritmasında olabileceği gibi, SSL/TLS'in kullandığı kriptografik algoritmalarda ve SSL/TLS mekanizmasını gerçeklemek için geliştirilen uygulamalarda da olabilmektedir. Bu çalışmada amacımız; SSL/TLS mimarisi ve altyapısı ile ilgili temel bilgilerden bahsettikten sonra, SSL/TLS sistemine yapılan saldırıları inceleyip, analiz etmek ve uzun vadede güvenilir bir protokole sahip olabilmemiz için nelere dikkat edilmesi gerektiğini göstermektir.

1.2.2 Motivasyon

Bu çalışmayı hazırlarken bizi SSL/TLS saldırılarını inceleyip dökümante etmeye nelerin motive ettiğinden de bahsetmek isteriz. Çalışma süresince de görülebileceği gibi; haberleşme güvenliği protokolleri pek çok sayıda saldırıya maruz kalmıştır ve bu saldırılar gizlilik, bütünlük ve mahremiyet gibi alanlarda tehditler oluşturmuştur. Bununla birlikte uluslararası mecrada ve devlet bazında belirlenen regülasyonlarda karar mekanizmalarını etkiler hale gelmiştir.

Hem kişisel, hem de toplumsal olarak bilgilerimizin güvenliğinin tehlikede olduğu gerçeğine vurgu yapmak ve bu konularda farkındalık sağlayabilmek adına katkıda bulunmak, bizi motive eden maddelerdir.

1.2.3 Tezin İçeriği ve Başlıkları

- **Bölüm-2 Ön Bilgi ve Matemetiksel Yapılar:** Bu bölümde, SSL/TLS protokollerine yapılan saldırıları inceleyebilmek için gerekli olan birtakım konular yer almaktadır. Bu konular aşağıdaki gibidir:
  - Açık anahtar altyapısı ve sertifikaların önemi, kullanımı
  - Protokollerin ve kriptografik fonksiyonların anlaşılabilmesi için gerekli olan matematiksel temeller
  - Anahtar anlaşma protokollerinin gereksinimleri
- **Bölüm-3 Mevcut Anahtar Anlaşma Protokolleri ve Güvenlik Analizleri:** Bu bölüm, temel anahtar anlaşma problemlerinden ve sorunlarından yola çıkılarak günümüzdeki protokollere kadar yaşanan süreci işlemektedir. Protokollerin yaşadığı

sorunlardan bahsedilmiş ve hangi aşamalarla mevcut protokollere gelindiğinden bahsedilmiştir.

- **Bölüm-4 SSL/TLS Protokolü:** Bu bölümde, ayrıntılı bir şekilde SSL/TLS protokolünde haberleşme için kullanılan mesajlar incelenmiştir. Bu tez yazılırken en güncel versiyon olan TLSv1.2 versiyonunun mesaj yapısı üzerinden gidilmiş, eski sürümlerin ise farklılıklarından bahsedilmiştir.

- **Bölüm-5 Açık Anahtar Altyapısına Yapılan Saldırılar:** Açık anahtar altyapısı ve sertifikaların yıllar içerisinde yaşadıkları sorunlar bu başlıkta yer almaktadır. Sertifika otoritelerinin ve büyük şirketlerin Internet güvenliği sürecini etkileyecek ölçüde sorunlar yaşamasından ötürü bu konulara değinilmiştir.

- **Bölüm-6 TLS Protokolüne Yapılan Saldırılar:** Bu bölümde kronolojik olarak SSL/TLS protokollerine yapılan saldırılar, teknik ve ayrıntılı olarak incelenmiştir. Saldırıların ön koşulları, senaryoları ve önüne geçilebilmesi için gerekli önlemler başlıklar halinde anlatılmıştır.

- **Bölüm-7 Diğer Saldırılar:** Teknik olarak SSL/TLS protokollerine saldırı özelliği taşımayan fakat, ortaya çıkması ile protokollerin güvenliğine etki eden açıklıklardan bu bölümde bahsedilmiştir.

- **Bölüm-8 Sonuç:** Sonuç bölümünde, saldırıların genel olarak nasıl meydana geldiğinden, gelecekte saldırıların azalması için neler yapılması gerektiğinden ve güvenli Internet için atılabilecek adımlardan bahsedilmiştir.

Bölüm 2

Ön Bilgi ve Gerekli Matematiksel Altyapılar

2.1 Açık Anahtar Altyapısı

Asimetrik şifreleme temelli olan açık anahtar altyapısı, açık ve gizli anahtarlar kullanılarak şifreleme ve imzalama sistemlerini gerçeklemeyi mümkün kılmaktadır. Adından da anlaşıldığı gibi herkes tarafından bilinen açık anahtar kullanılarak veri şifrelenir. Bu veriyi de yalnızca gizli anahtara sahip olan kişi açıp okuyabilir. İmzalama yapılırken de, gizli anahtar kullanılarak ilgili veriye-dökümana imza atılır ve herhangi bir kişi açık anahtarı kullanarak imzayı doğrulayabilir. Şifreleme için kullanılan algoritmalardan bazıları imzalama için kullanılabileceği gibi, imza ve şifre için farklı algoritmalar kullanmak da mümkündür.

Güvenli iletişim kurulurken, saldırganın açık olan tüm değerlere sahip olduğu varsayılır. Bununla birlikte saldırganın kullanılan algoritmaları da bilmektedir. Bu süreçte sistemin gizliliği anahtarın gizli kalmasına dayanmaktadır (Kerckhoff Prensibi için bkz. [16]).

1976 yılına dek, açık anahtarlı bir algoritma yöntemi önerilememişti. Diffie ve Hellman tarafından ortaya çıkarılan ve devrimsel kabul edilen bir yöntemle açık anahtar altyapısının temelleri atıldı [27]. Bulunan fonksiyonun özelliği, tek yönlü fonksiyon olmasıydı. Tek yönlü fonksiyonlar hesaplanması kolay fakat ters görüntünün bulunması yüksek işlem gücü gerektiren fonksiyonlardır. Tek yönlü fonksiyonların açık anahtarlı sistemlere uygun olması için sahip olması gereken bir diğer özelliği ise, ters görüntünün gizli anahtara sahip bir kişi tarafından kolay şekilde hesaplanabilmesidir. Diffie ve Hellman'dan kısa bir süre sonra Rivest, Shamir ve Adleman tarafından RSA algoritması geliştirildi

[83]. RSA yöntemi, günümüzde halen güvenli kabul edilmekte ve en yaygın kullanılan algoritma olarak yaşamını sürdürmektedir.

Açık anahtarlı sistemlerin gelişmesi ile, anahtarların, imzaların ve güvenli sistemlerin oluşturduğu bir açık anahtar altyapısı mimarisi ortaya çıkmıştır. Açık anahtar altyapısı sayesinde insanlar, öncesinde paylaşılmış bir anahtar olmaksızın güvenli iletişim sağlayabilir hale gelmişlerdir. Ancak açık anahtar bir çok soruyu ve sorunu da beraberinde getirmiştir:

- Birbirini tanımayan iki insan nasıl güvenli haberleşebilir?
- Açık anahtarlar nasıl üretilir ve yaşam döngüsü ne şekilde yönetilir?
- Tarafların iletişimi boyunca gizlilik, bütünlük, doğrulama, inkar edilemezlik gibi kritik gereksinimler nasıl sağlanabilir?
- Açık anahtar paylaşımı nasıl yapılmalıdır?

Yukarıdaki sorunların tespiti ve çözümü, aynı zamanda bu çözümlerin dünya çapında kullanılması konusu açık anahtar altyapısının temellerini oluşturmuştur. Açık anahtar altyapısının temelinde sertifikalar bulunur. Sertifikalar ise tarafların birbirlerinin kimliklerini doğrulamak amacıyla ortaya çıkmışlardır.

2.1.1 Sertifika

Açık anahtar sertifikaları, güvensiz ortamda veri iletişimi yapılırken kişinin kimlik doğrulamasının ve ilgili verinin bütünlüğünü sağlayan veri bloğudur [67]. Sertifikaların amacı açık anahtarları başka kişiler tarafından kimlik onaylaması yapılabilecek hale getirmektir. Sertifikaların güvenliği sertifikayı sağlayan yetkili mercinin güvenli olması varsayımına dayanmaktadır.

Sertifika dijital bir belgedir. İçeriği, veri bölümü ve imza bölümü olmak üzere iki kısımdan oluşur.

2.1.1.1 Sertifika İçeriği

Açık anahtar altyapısı sertifikalarının içerdiği alanlar Tablo 2.1'deki gibidir [49]:

- **Versiyon:** Üç adet sertifika versiyonu bulunmaktadır. İlk versiyonda temel alanlar bulunurken ikinci versiyon ile ilave alanlar eklenmiş ve üçüncü versiyonda eklentiler kısmı dahil edilmiştir. Günümüzde sertifikaların tümüne yakını üçüncü versiyondur.

Tablo 2.1: Sertifika İçeriği

Versiyon
Seri Numarası
İmza Algoritması
Sertifikayı Veren Kuruluş
Geçerlilik Süresi
Konu
Açık Anahtar

- **Seri Numarası:** Seri numaraları, sertifika otoritesi tarafından sertifikaya eklenen ve her bir sertifikaya eşsiz olan numaralardır.
- **İmza Algoritması:** Bu alanda sertifikanın imzası için kullanılan algoritmanın bilgisi bulunur.
- **Sertifikayı Veren Kuruluş:** Bu alan içerisinde sertifikayı veren kuruluş ile ilgili bilgi bulunur. Sertifikayı veren kuruluşun bulunduğu ülke, çalışma alanı gibi bilgiler yer alır.
- **Geçerlilik Süresi:** Sertifikanın geçerli olduğu tarihin bulunduğu kısımdır. Ne zamandan itibaren ve ne zamana kadar geçerli olacağının tarihleri bulunur.
- **Konu:** Konu alanı, açık anahtarın içerdiği değeri tutar (örneğin web sitesinin adres bilgisi).
- **Açık Anahtar:** Bu alan açık anahtarı, açık anahtarda kullanılan algoritmayı ve parametreleri içerir.

2.1.1.2 Sertifika Eklentileri

Sertifikaların üçüncü versiyonu ile eklenen eklentiler sertifika yapısına esneklik kazandırma amacı ile ortaya çıkmıştır. Eklentilerin kimlik değerleri ve kritiklik seviyeleri bulunur. Kritik eklentiler, sertifikanın kabul edilebilmesi için sertifikayı kontrol eden tarafın incelemesi gereken eklentilerdir. Sertifikanın özelliklerine dair kısıtlar, kullanım alanları ve yetkileri gibi bilgiler de eklentilerde bulunur.

2.1.2 Sertifika Zincirleri

Kullanıcıların kendilerini ispatlamak için sertifikaya ihtiyaç duydukları gibi, sertifikalar da kendilerini ispatlamak için kök sertifikalara ihtiyaç duyar. Bir sertifikanın güvenliğini başka bir sertifikaya, onun da kendini bir başka sertifikaya bağlamasına ve böylece kök

sertifikaya kadar gelinmesine sertifika zinciri adı verilir. Kök sertifikası ise Google, Microsoft gibi şirketlerin güvendikleri ve işletim sistemlerinin kurulumunda, web tarayıcılarının kurulumlarında bilgisayara yüklenen sertifikalardır. Kök sertifikalar, ara sertifikaları imzalar ve güvenliklerini tescil eder. Ara sertifikalar da başka ara sertifikaları ya da son kullanıcıya hitap eden sertifikaları imzalar ve güvenliklerini tescil eder [106].

2.1.3 Sertifika Otoriteleri

Sertifika Otoritesi sertifikayı onaylayan ve ilgili kişinin açık anahtarının kimlik doğrulamasını yapan bir yapıdır ve Internet üzerinde güven modelinin oluşmasındaki en kritik aktördür [67, 81]. Sertifika onaylama işlemi kişinin açık anahtarının Sertifika Otoritesi'nin gizli anahtarı ile imzalanması şeklinde gerçekleşir.

2.1.4 Sertifika Yaşam Döngüsü

Sertifikaların kullanımı, üretilmesi, silinmesi gibi pek çok aşaması vardır ve bu aşamaların herbiri zaman içerisinde ortaya çıkan açıklıklardan ötürü dikkatle ele alınan süreçlerdir [81]. Sertifikaların yaşam süreci sertifika sahibi olmak isteyen bir kullanıcının sertifika imzalama isteği (Certificate Signing Request - CSR) oluşturması ve bu belgeyi istediği bir Sertifika Otoritesi'ne göndermesi ile başlar. CSR nin temel amacı ilgili açık anahtarı tanıtmak ve buna bağlı gizli anahtarın varlığını ispatlamaktır (ispatlama, imza ile gerçekleştirilir). Sonrasında Sertifika Otoritesi aşağıdaki yöntemlerden birisi ile başvuran kişiyi onaylar:

- **Alan Adı ile Onaylama:** Alan adı ile onaylama (Domain Validation) yapılan sertifikalar kişinin o alan adı üzerinde yetkisi olduğunun ispatıdır. Bu onaylamanın gerçekleşmesi, alan adına sahip e-posta adresine doğrulama postası atılması ile gerçekleştirilebilir. Postayı alan kişi ilgili bağlantıya tıklamak gibi bir işlemle de alan adına sahip olduğunu ispatlayabillir. Bu onaylama telefona mesaj gönderilmesi ve mesajdaki kodun cevap olarak dönülmesi gibi basit bir işlemle de yapılabilir.
- **Organizasyon Onaylama:** Organizasyon onaylaması (Organization validation) ile yapılan sertifika onaylamaları kimlik bilgisi ve kimlik doğrulama gerektirir.
- **Geliştirilmiş Onaylama:** Geliştirilmiş onaylama (Extended Validation), organizasyon onaylama sürecinde olduğu gibi kimlik bilgisi ve doğrulama gerektirir fakat daha katı gereksinimlere sahiptir. Organizasyon onaylama ile ortaya çıkan sorunları kaldırmak amacı ile ortaya çıkmıştır ve sürecin takip edilmesi ve her aşamanın raporlanması gibi önlemlerle tutarlı bir sürece sahiptir.

Başarılı bir onaylama sonrasında Sertifika Otoritesi, sertifikayı yürürlüğe sokar. Böylece sertifika için başvuran kullanıcı sertifikayı kullanmaya başlar. Sertifika, geçerlilik süresi bitene kadar veya yürürlükten kaldırılana kadar kullanılabilir. Eğer sertifikaya ait olan gizli anahtar ifşa olursa sertifika iptal süreci başlar. Bu süreç de sertifika alma süreci gibi başvuru ve onaylama ile olur.

2.1.5 Sertifika İptali

Sertifikalar, ilgili gizli anahtar ifşa olunca veya artık ihtiyaç duyulmadığında iptal edilir. Böyle bir durumda kötüye kullanılma riski ortaya çıkar. Özellikle gizli anahtarı ifşa olan sertifikaların iptal edilmesi ve iptal edildiği bilgisinin açık olarak duyurulması gerekir. Internet üzerindeki kullanıcıların iptal edilen sertifikalardan haberdar olabilmesi için Sertifika İptal Listesi (Sertificate Revocation List - CRL) ve çevrimiçi sertifika durumu protokolü oluşturulmuştur.

2.1.5.1 Sertifika İptal Listesi (SİL)

SİL kullanım süresi bitmeden iptal olmuş tüm sertifikaların bulunduğu bir listedir [81]. Bu listelerin kontrolü sertifika otoriteleri tarafından sağlanır. Sertifikalar kullanılırken, o sertifikanın iptal olması durumunda hangi SİL'de bulunacağı bilgisi de bulunmalıdır. Böylece tüm listelerin kontrol edilmesi gerekmez. SİL ile alakalı en büyük sorun ise bu listelerin çok büyük olması ve anlık sorgulamaların mümkün olmamasıdır. SİL dökümanları belirli aralıklarla güncellendiği için, son güncellemeden bu yana iptal olan sertifikalar listede yer almaz.

2.1.5.2 Çevrimiçi Sertifika Durumu Protokolü (ÇSDP)

Çevrimiçi Sertifika Durumu Protokolü (Online Certificate Status Protocol) güvenen yapıların (relying party) bir sertifikanın iptal olup olmadığı bilgisini sunar. Bir sertifikanın iptal edilme bilgisini verecek olan ilgili ÇSDP'nin erişim bilgisi, o sertifikanın "Authority Information Access" eklentisi alanında bulunur.

ÇSDP'ler SİL'lerden farklı olarak soru ve cevap şeklinde çalışır ve bu sayede gerçek zamanlı olarak sertifika iptali bilgisi elde edilebilir. Ancak yine de ÇSDP'ler ile iletişim kurmak da iletişim maliyetini performans sorunları sebebiyle artırmaktadır.

2.2 Matematiksel Temeller

Kriptografik sistemlerin güvenlikleri çoğu zaman çözülmesi zor olan matematiksel problemlere dayandırılır. En çok kullanılan iki problem ise çarpanlarına ayırma (integer factorization) ve ayrık logaritma (discrete logarithm) problemleridir.

Kriptografik sistemlerde saldırganın çok güçlü olduğu varsayılır. Bir hesaplama işleminin (en fazla ihmal edilebilir girdi haricinde) polinom sürede çözülebilir olması, o işlemin kolay olduğu anlamına gelir. Yani, eğer bir algoritma bir problemin ihmal edilemeyen kadar kısmını polinom sürede çözüyorsa, bu probleme dayanan kriptografik sistem de güvensiz kabul edilir.

Zor problemlerdeki amaç, bir fonksiyonun hesaplanmasının kısa sürede yapılabilmesi fakat aynı fonksiyonun tersinin çok uzun sürelerde hesaplanmasıdır. Örneğin; iki asal sayının çarpımı, sayılar ne kadar büyük olursa olsun, polinom sürede hesaplanabilir. Büyük bir sayının asal çarpanlarına ayrılması ise polinom sürede hesaplanamayacak karmaşıklıktadır.

2.2.1 Ayrık Logaritma Problemi (ALP)

Ayrık logaritma problemi (Discrete Logarithm Problem), pek çok kriptografik mekanizmanın güvenliğinin dayandığı bir problemdir [1]. Kısaca, ayrık logaritma problemi, verilen $g, y \in \mathbb{G}$ değerleri için $g^x = y$ eşitliğini sağlayan x değerinin bulunması problemidir [2] [61]. Normalde reel sayılarda logaritma problemi zor bir problem değildir. Bunun nedenlerinden birisi, üs alma işleminin monoton özelliğini sağlamasıdır ($\forall x > y$ ve $b > 1, b^x > b^y$). Bundan ötürü logaritma işlemi (örn. Taylor Serisi) hesaplanması kolay bir işlemdir. Ancak ayrık logaritma problemi monoton bir özelliği sağlamamasından dolayı zor bir problem olduğu kabul edilir, ancak buna rağmen bu problemin karmaşıklığını azaltmak için pek çok analiz çalışması yapılmıştır (örn. Kabakuvvet arama (exthausive search) algoritması, Bebek ve dev adımı (Baby-step giant-step) algoritması, Pollard ρ algoritması, Pohlig-Hellman algoritması, Index-calculus algoritması [74]). Bu analizlerin ve saldırıların gelişimine bağlı olarak anahtar boyları uzamaktadır [42].

2.2.2 Hesapsal & Kararsal Diffie-Hellman Problemleri (HDH & KDH)

Diffie-Hellman (DH) problemi ayrık logaritma problemi ile yakından ilişkilidir. DH problemi zorluğuna dayanarak Diffie-Hellman anahtar anlaşması ve ElGamal açık anahtarlı

[1] Aksi belirtilmedikçe, bu tezdeki $\mathbb{G}$ grubu, mertebesi n ve üreteci g olan sonlu devirli bir grubu temsil etmektedir. Genel olarak pratikte $\mathbb{G}$, mertebesi $p-1$ olan $\mathbb{Z}_p^*$ çarpımsal grubu olarak kullanılmaktadır.

[2] Aksi belirtilmedikçe bundan böyle bütün işlemler $\mathbb{G}$ grubunda yapılacaktır.

şifreleme gibi kriptografik mekanizmalar gerçekleştirilebilmektedir. Diffie-Hellman problemi de iki alt probleme ayrılmaktadır: Hesapsal DH (HDH) ve Kararsal DH (KDH).

- Hesapsal Diffie-Hellman Problemi, verilen bir p asal sayısı, $\mathbb{Z}_p^*$ devirli grubuna ait g üreteci ve g^a, g^b değerleri için g^{ab} değerinin hesaplanabilmesidir.
- Kararsal Diffie-Hellman Problemi, verilen bir p asal sayısı, $\mathbb{Z}_p^*$ devirli grubuna ait g üreteci ve g^a, g^b, g^c değerlerinin (a, b değerlerine ihtiyaç duymadan) $g^c = g^{ab}$ formunda olup olmadığının hesaplanabilmesidir.

HDH problemi, KDH probleminden daha zordur. Bu kıyaslama aşağıdaki gibi yapılabilir:

- $\mathbb{Z}_p^*$ içerisinde HDH efektif bir şekilde çözülebilir olsun.
- Verilen g^a ve g^b değerleri ile g^{ab} değeri hesaplanabilir olur.
- Böylece g^{ab} değerinin g^c değeri olup olmadığı da anlaşılabilir.
- Sonuç olarak hesapsal Diffie-Hellman probleminin çözümü, kararsal Diffie-Hellman probleminin de çözümü demektir. Ancak bu durumun tersi mümkün değildir. Bu yüzden polinom süresi olarak HDH, KDH'den daha zor bir problemdir (HDH$\geq_P$KDH).

HDH ile DLP arasındaki bağlantı ise, aşağıdaki gibi bir yaklaşımla anlaşılabilir:

- $\mathbb{Z}_p^*$ içerisinde ALP efektif bir şekilde çözülebilir olsun.
- Verilen $g, p, g^a \mod p, g^b \mod p$ değerleri için a değerinin g, p, g^a değerleri kullanılarak hesaplanması mümkün olur.
- Sonrasında ise, $(g^b)^a$ işlemi yapılarak g^{ab} sonucu elde edilebilir.
- Sonuç olarak ayrık logaritma probleminin çözümü, hesapsal Diffie-Hellman probleminin de çözümü demektir. Ancak bu durumun tersi mümkün değildir. Bu yüzden polinom süresi olarak ALP, HDH'den daha zor bir problemdir (ALP$\geq_P$HDH).

Yukarıdaki iki kıyaslamanın sonucu olarak; ALP$\geq_P$HDH$\geq_P$KDH sonucu ortaya çıkmış olur.

2.2.3 Çarpanlarına Ayırma Problemi

Çarpanlarına ayırma problemi, büyük bir sayının çarpanlarına ayırılmasının zorluğuna dayanır [61]. Verilen bir N sayısı için p_i farklı asal sayılar ve $e_i \geq 1$ eşitsizliği için $N = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$ şeklinde asal çarpanlarının bulunması problemidir.

İki adet sayının çarpımının hesaplanması kolay iken, verilen bir sayının asal çarpanlarını bulmak zordur. Eğer çarpanlardan birisi bilinirse, diğer çarpanı bulmak kolay hale gelir. RSA şifreleme algoritmasının temelini bu problem oluşturur. Bölüm 3.7'de RSA algoritması anlatılmıştır.

2.2.4 Eliptik Eğri Ayrık Logaritma Problemi

Eliptik eğrilerin kriptografide kullanılması ilk olarak 1985'te ortaya atıldı [75]. Eliptik eğriler üzerindeki noktalar kullanılarak işlemler gerçekleştirilir [61]. Bu noktalar ile gerçekleştirilen işlemler tuzak kapılı (trapdoor) tek yönlü fonksiyon haline gelir. Örneğin bir eliptik eğri grubunda P başlangıç noktası seçelim. Bir kullanıcı gizli k değerini seçer ve $Q = k \cdot P$ değerini hesaplar. Burada Q kişinin açık anahtarı, k'de kişinin gizli anahtarı olur. Burada, eliptik eğri ayrık logaritma problemi kullanılır. Q ve P değerleri bilinirken k'yı bulmak zor bir problemdir.

Eliptik eğriler düşük anahtar boylarında dahi yüksek güvenlik sağlarlar. 1024 bitlik RSA anahtarının sağladığı güvenliği, eliptik eğriler 160 bit ile sağlayabilir. Ancak bu alan henüz implementasyonu zor olan ve geliştirilmesi için araştırmaların devam ettiği bir alandır.

2.2.5 Tek Yönlü Fonksiyonlar

Tek yönlü fonksiyonlar bir yöne doğru çalıştırılması kolay fakat diğer yöne doğru çalıştırılması zor matematiksel fonksiyonlardır [61]. Tuzak kapı (trapdoor) tek yönlü fonksiyonlar ise, tersi zor olan fonksiyonun ekstra bir bilgi bilinmesi halinde kolay hale gelmesidir. Eğer bu ekstra bilgi bilinmiyorsa tersine işlem zorluğunu korur.

Açık anahtarlı kriptografik sistemler, temelinde tuzak kapılı tek yönlü fonksiyonlara dayanır. İşlemin tersi zor iken kolay hale gelmesini sağlayan bilgi de gizli anahtardır. Gizli anahtara sahip olmayan kişi fonksiyonu yalnızca ileri yönde çalıştırabilir. İleri yöndeki işlem şifreleme ve imza doğrulama için kullanılırken, geri yöndeki işlem gizli anahtar sahibi tarafından şifre çözme ve imzalama için kullanılır.

2.2.5.1 Özet Fonksiyonları

Özet fonksiyonları, günümüzde pek çok alanda kullanım gören ve genellikle veri bütünlüğü kontrolü sağlayan fonksiyonlardır. Ayrıca klasik RSA algoritmasına gerçekleştirilen saldırılardan korunmak için kullanılan yöntemlerden birisidir. Özet fonksiyonu, herhangi bir uzunluktaki bir girdiyi sabit uzunlukta bir çıktıya çeviren, tek yönlü kriptografik bir fonksiyondur[16]. Herhangi bir girdinin özeti kolaylıkla alınabilirken, özet bilgisinden orijinal veriye ulaşılması zor bir problemdir. Özet fonksiyonların sağlaması gereken bazı özellikler bulunmaktadır. Bunlardan en önemli olanı çakışmaya karşı dayanıklı olmasıdır. Çakışmaya dayanıklılık; iki farklı girdinin aynı özet değerini vermemesini sağlamak anlamına gelir. Yani birbirinden farklı M_1 ve M_2 mesajları için $\text{Hash}(M_1) = \text{Hash}(M_2)$ eşitliğinin sağlanamaması gerekmektedir.

Eğer verilen M_1 mesajı ve $\text{Hash}(M_1) = H$ olacak şekilde H özet değerine sahip bir saldırgan, aynı özet değerine sahip başka bir M_2 mesajı oluşturabilirse ($\text{Hash}(M_1) = \text{Hash}(M_2)$) buna ikinci öngörüntü saldırısı denir. Eğer verilen bir H özet değerine sahip bir saldırgan $H = \text{Hash}(M)$ olacak şekilde M mesajını oluşturabilirse buna öngörüntü saldırısı denir.

Özet fonksiyonları genelde aşağıdaki amaçlar doğrultusunda kullanılır;

- **Veri bütünlüğü sağlama:** Bir verinin değişikliğe uğrayıp uğramadığını anlamak için kullanılır. Alınan özet değeri, daha önceden alınan özet ile uyuşuyorsa veri zaman içinde değişikliğe uğramamış demektir.
- **Parola kaydı tutma:** Parolalar veri tabanlarında açık metin halinde tutulmamalıdır. Bunun yerine özet değerleri tutulur ve kullanıcılar parolalarını girdiklerinde parolanın özeti ile sistemde tutulan özet kıyaslanarak doğrulama gerçekleştirilir ve bu şekilde parolaların gizliliği korunmuş olur.
- **Döküman imzalama:** Döküman imzalama verinin tamamının işlemden geçmesi demektir ve imza süreci uzun hale gelir. Bunun yerine daha hızlı olan özet algoritması ile işlem yapılır. Dökümanın önce özeti alınır, sonra özete imza atılır. İmza doğrulama yapan kişi de dökümanın özetini alarak imzalanan veri ile kıyaslama yapar.

Uzun süre boyunca MD5[82] ve SHA-1[34] özet algoritmaları yaygın bir biçimde kullanılmıştır. Ancak bu yöntemler günümüzde güvensiz hale gelmiştir. 2004 yılında Xiaoyun Wang ve takımı tarafından MD5 algoritmasında çakışmalar olduğu ortaya çıkarılmıştır [98]. Aynı ekip, SHA-1 algoritmasının güvenliğinin de 2^{69} bite düştüğünü gösterdi [99]. Günümüzde ise halen güvenli kabul edilen SHA-2 ve SHA-3 aileleri kullanılmaktadır. En

düşüğü 2^{112} bitlik güvenliğe sahip olan bu algoritmalar, işlem gücü olarak günümüzde erişilmesi zor değerlerdir [41].

Klasik RSA algoritmasında oluşan imza sorununu çözebilmek için önce metnin sonra özetin imzalanması fikri geliştirildi (hash-then-sign). RSA Laboratories şirketi tarafından geliştirilen PKCS #1 standartı ile [53], özet değeri, önüne bir ön ek eklenerek imzalanmıştır.

2.2.6 Açık Anahtarlı Algoritmalar

2.2.6.1 ElGamal Açık Anahtar Şifreleme Algoritması

Taher ElGamal tarafından 1985 yılında geliştirilmiş ElGamal algoritması, RSA açık anahtar şifrelemeye alternatif bir yöntemdir [36]. ElGamal algoritmasının güvenliği KDH'ye dayanır (bkz. 2.2.2). ElGamal algoritması, Diffie-Hellman anahtar anlaşması protokolünün açık anahtarlı bir sisteme dönüştürülmesidir. Bu algoritmanın şifreli metnin açık metnin iki katı uzunluğunda olması dezavantajı vardır. Bununla birlikte aynı metnin her şifrelendiğinde farklı bir şifreli metin oluşturması ile semantik güvenlik (yani birbirinden ayırt edilemeyen şifreli metinler) sağlaması avantajı vardır. ElGamal algoritması anahtar oluşturma, şifreleme ve şifre çözme olarak 3 kısımda incelenebilir:

1. Anahtar oluşturma:
   (a) Ayşe mertebesi q ve üreteci g olan devirli bir $\mathbb{G}$ grubu oluşturur.
   (b) Sonrasında $1 \leq x \leq q-1$ olacak şekilde rastgele bir x sayısı seçer.
   (c) Ayşe $h := g^x$ işlemini yapar.
   (d) Ayşe $\mathbb{G}, q, g$ ve h değerlerini yayınlar. Bu değerler onun açık anahtarını ifade etmektedir. x değeri de Ayşe'nin gizli anahtarı olur.
2. Şifreleme
   (a) Bora, Ayşe'ye göndemek istediği m mesajını Ayşe'nin açık anahtarını $(\mathbb{G}, q, g, h)$ kullanarak yapar.
   (b) Bora $1 \leq r \leq q-1$ olacak şekilde rastgele bir r sayısı seçer ve $c_1 := g^r$ değerini hesaplar.
   (c) Sonrasında $c_2 := m \cdot h^r$ değerini hesaplar.
   (d) Bora şifreli metin olarak $(c_1, c_2) = (g^r, m \cdot h^r) = (g^r, m \cdot g^{xr})$ ikilisini Ayşe'ye gönderir.
3. Şifre Çözme

(a) Ayşe Bora'dan gelen (c_1, c_2) şifreli mesajı çözmek için x gizli anahtarını kullanır.

(b) Ayşe $s := {c_1}^x$ değerini ve $m := c_2 \cdot s^{-1}$ değerini hesaplar. Bu işlemin doğruluğu aşağıdaki şekilde sağlanabilir:

$$c_2 \cdot s^{-1} = m \cdot h^r \cdot (g^{xr})^{-1} = m \cdot g^{xr} \cdot g^{-xr} = m$$

2.2.6.2 Dijital İmza Algoritması

Dijital imza algoritması (DSA), Amerika Birleşik Devletleri'nin dijital imzalar için geliştirdiği standardın ismidir. En güncel hali ile Ulusal Teknoloji ve Standartları Enstitüsü tarafından 2013 yılında yayınlanmıştır [38]. Anahtar oluşturma ve imzalama olarak iki aşamadan oluşur.

DSA algoritmasının ilk kısmı, aşağıdaki gibi açık ve gizli anahtar üretimi ile olur:

- Dijital imza sahibi olmak isteyen kişi (sunucu olsun) bir q asal sayısı belirler.
- $p - 1 \mod q \equiv 0$ olacak şekilde p asal sayısı belirler.
- $1 < g < p$, $g^q \mod p \equiv 1$ ve $g = h^{(p-1)/q} \mod p$ olacak şekilde bir g üreteci oluşturur.
- $0 < x < q$ koşulunu sağlayan bir x sayısı seçer.
- $y = g^x \mod p$ işlemini yapar.
- Sunucunun açık anahtarı, $p||q||g||y$ olur.
- Sunucunun gizli anahtarı, $p||q||q||x$ olur.

DSA algoritmasının ikinci kısmı imza oluşturma ve imza doğrulama aşamalarıdır. İmzayı oluşturmak için sunucu aşağıdaki adımları izler:

- Gönderilmek istenen mesajın özeti $h = Hash(M)$ olacak şekilde alır.
- $0 < k < q$ olacak şekilde rastgele bir k sayısı seçer.
- $r := g^k \mod p \mod q$ şeklinde r sayısını oluşturur. Eğer $r = 0$ ise yeni bir k değeri seçerek işlemi tekrarlar.
- $k \cdot i \mod q \equiv 1$ denkliğini sağlayan i değerini hesaplar. i değeri, q modunda k sayısının çarpma işlemine göre tersidir.

- $s := i \cdot (h + r \cdot x) \mod q$ şeklinde s sayısını oluşturur. Eğer $s = 0$ ise yeni bir k değeri seçerek işlemi tekrarlar.
- İmzalı paket, r, s şeklindedir.

Aldığı mesajı doğrulamak isteyen (istemci olsun) taraf için ise imza doğrulama aşağıdaki gibidir:

- Kendisine ulaşan mesajın $h = Hash(M)$ olacak şekilde özetini alır.
- $s \cdot w \mod q \equiv 1$ olacak şekilde w değerini hesaplar. i değeri, q modunda k sayısının çarpma işlemine göre tersidir.
- $u_1 := h \cdot w \mod q$ değerini hesaplar.
- $u_2 := r \cdot w \mod q$ değerini hesaplar.
- $v := (((g^{u_1}) * (y^{u_2})) \mod p) \mod q$ değerini hesaplar.
- Eğer $v = r$ ise, dijital imza geçerlidir.

2.3 Anahtar Anlaşma Protokolleri

Güvensiz ortamda güvenli iletişimin sağlanması, anahtar anlaşmasının doğru yapılmasından geçer. Eğer taraflar yalnızca kendilerinin bildiği gizli bir değerde anlaşabilirler ise olası saldırıların pek çoğu ortadan kalkar. Ancak, anahtar anlaşma protokolleri yıllar boyunca hep eksiklikler barındırmıştır. Dikkatlice yazılmış ve yaygın kullanıma sahip protokoller bile basit yaklaşımlarla (kriptografi olmadan) alt edilebilmiştir. Bu protokolerden aşağıdaki gibi özellikleri sağlaması beklenmektedir:

- Protokolün canlı ortamda kontrol edilmesi zordur. Bu yüzden geliştirilirken eş zamanlı çalışabilen ve implementasyonu kolay yapılabilen bir protokol geliştirilmelidir.
- Aktif saldırganlar farkedilmesi zor aktörlerdir. Protokolün kullanıcılarına hizmet verirken diğer yandan kötü kullanımlardan etkilenmemesi gerekir.
- Protokollerin gizlilik, bütünlük, kimlik doğrulama, yetkilendirme gibi gerekleri yerine getirmesi gerekir.

2.3.1 Anahtar Anlaşma Protokolünde Güvenlik Gereksinimleri

Bir protokolün güvenliğinin analiz edilmesi için birtakım gereksinimleri karşılaması beklenmektedir [96]. Bu bilgiler kısaca aşağıdaki gibi tanımlanabilir;

- **Bilinen anahtar güvenliği (Known key security):** Protokol ile oluşan oturum anahtarı eşsiz olmalıdır. Eğer bu anahtar ifşa olursa, bu durum diğer oturumların güvenliğini tehdit etmemelidir.
- **İleriye yönelik gizlilik (Forward secrecy):** İleriye yönelik gizlilik; bir kişinin gizli anahtarının ifşa olması durumunda o kişiye ait geçmiş oturum bilgilerinin halen güvende olmasını sağlamak demektir [74]. Bir kişinin gizli değerinin zaman içerisinde ifşa olma riski göz önüne alındığında, o kişiye ait verilerin gizliliğinin devam edebilmesi gerekir. Bu durumda güvenli iletişim kurmak isteyen tarafların gizli anahtarları ile birlikte oturum anahtarlarına rastgelelik eklemesi gerekir.

  Ayşe ile Bora temel Diffie-Hellman algoritması ile anahtar anlaşması yapmış olsun. gizli a, b değerleri ile oluşturulan oturum anahtarı $K = g^{ab}$ dir. Ayşe ile Bora'nın başka bir zamanda tekrar anahtar anlaşması yapması durumunda oluşacak oturum anahtarı yine $K = g^{ab}$ olur. Eğer her bir oturumda farklı oturum anahtarı oluşturulursa ve oturum anahtarı yalnızca a ve b değerlerine bağlı kalmazsa, böylece gizli a, b değerlerinin ifşa olması durumunda bu oturum anahtarları gizli kalabilirse, yönelik gizlilik sağlanmış olur.
- **Anahtar ifşası ile yerine geçmeye dayanıklılık (Key compromise impersonation resilience):** Eğer Ayşe'nin gizli anahtarı ifşa olursa, başka bir kişi Ayşe'nin yerine geçebilir. Ancak, herhangi birisi Ayşe'nin karşısına geçip Ayşe'nin iletişim kurmak istediği bir taraf gibi davranamamalıdır.
- **Bilinmeyen anahtar paylaşımına dayanıklılık (Unknown key share resilience):** Eğer Ayşe ve Bora oturum anahtarı oluşturmak istiyorsa, araya giren Ece'nin, oluşan oturum anahtarını bilmemesine rağmen ve Bora'nın Ayşe ile konuştuğunu bilmesine rağmen, Ayşe'nin kendisi (Ece) ile konuştuğuna kandıramaması gerekmektedir.
- **Anahtar kontrolü (Key Control):** Taraflar, oturum anahtarının değerinin istedikleri bir değer olmasını zorlayamamalıdırlar.

2.3.1.1 Kimlik Doğrulama

Bir protokolün sağlaması gereken güvenlik gereksinimlerinden birisi de, kimlik doğrulama yapabilmesidir. Çünkü güvenli iletişim kurmak isteyen tarafların birbirlerine kim

oldukları bilgisini ispatlamaları gerekir. Ağ üzerinde iki taraf arasında bir çok ara nokta (switch, router vs.) bulunmaktadır ve ağ üzerinde paketler manipülasyona açık halde dolaşır. İki tarafın da doğrulama yapmaması, ortadaki adam saldırısına karşı zayıflık ortaya çıkarır. Yaygın olarak kullanılan web ortamında istemcinin doğrulama yapması istenmez. Ancak E-Devlet gibi sayısal imzanın kullanıldığı ortamlarda istenebilir. Eğer sunucu istemcinin kendisini doğrulamasını isteyecek olursa, protokolün bu isteği karşılayabilmesi gerekir.

Kimlik doğrulama yapılarak kurulan bağlantılara kimlik doğrulamalı anahtar anlaşma protokolü denir [15]. Kimlik doğrulama iki kategoride incelenebilir. Birincisi parola kullanımı ile kimlik doğrulama yapılması, ikinci de asimetrik şifreleme kullanılarak ortak bir gizli değer oluşturma ile yapılabilir. Internet ortamında kimlik doğrulama ikinci yöntemle yapılır. Çünkü parola paylaşılabilmesi için de güvenli bir ortam oluşturulmalıdır ve bu ortam yalnızca asimetrik şifreleme ile mümkün olmaktadır.

Kimlik doğrulamalı anahtar anlaşma protokollerinin aşağıdaki iki gereksinimden birisini karşılaması gerekir [22]:

- **Aşikar olmayan anahtar doğrulama (implicit):** Bora, Ayşe'nin oluşturulan anahtara sahip olabilecek tek kişi olabileceğine ikna olur.
- **Aşikar olan anahtar doğrulama (explicit):** Bora, Ayşe'nin oluşturulan anahtara sahip olabilecek tek kişi olabileceğine ikna olur ve Ayşe, anahtara sahip olduğunu ispatlar.

Kimlik doğrulamalı anahtar anlaşma protokollerinin aşağıdaki iki gereksinimi de karşılaması gerekir [100]:

- **Karşılıklı kimlik doğrulama (Mutual authentication):** Tarafların ikisi de kimliklerini karşı tarafa doğrulatmak zorundadır.
- **Güvenli iletişim (Secure communication):** Taraflar aralarındaki iletişimi güvene alabilmek için bir oturum anahtarında anlaşmaları gerekir.

Bu özellikler anahtar anlaşma protokollerinin değerlendirilmesinde kullanılan önemli unsurlardır. Bu özelliklerin ortaya çıkışı, protokollerin açıklıklarının çıkması, açıklıkların giderilmesi ve sonrasında tekrar açıklık çıkması şeklinde döngüsel olmuştur. Bölüm 3.1'de anlatılan protokol neredeyse hiç bir gereksinimi sağlamamaktadır. Sonrasında eklenen özellikler ve yeni protokoller ile güvenli protokol gelişimi süreci olgunlaştırılmıştır.

2.3.2 Anahtar Anlaşma Protokolüne Yapılan Saldırılar

Anahtar anlaşma protokolüne saldırı yapabilmenin birkaç yolu vardır. Bu bölümde başlıklar halinde atakların karakterleri, aktif-pasif atak türleri ve genelgeçer atak yöntemlerinden bahsedilecektir.

2.3.2.1 Saldırı Karakteristiği

Bir saldırının uygulanabilir olup olmadığı aşağıdaki sorulara verdiği cevaplarla belirlenebilir [86]:

- Saldırıyı gerçekleştirmek için ne kadar bilgi ve beceri gerekmektedir?
- Ne kadar gizli bilginin ifşa olmuş olması gerekir?
- Gerçekleştirebilmek için ne kadar çaba harcanmalıdır?

Saldırıların gerçek hayatta uygulanabilirliği ve tehditlerinin ulaşabileceği boyutlar değişkenlik gösterir. Protokol geliştirilirken en kötü senaryo ele alınmalı, saldırganların kapasitelerinin ve motivasyonlarının yüksek olduğu düşünülmelidir.

2.3.2.2 Aktif ve Pasif Saldırılar

Anahtar anlaşma protokolüne yapılan saldırılar aktif ve pasif saldırılar olarak iki kısımda incelenebilir;

- Pasif saldırı, saldırganın yalnızca izleme becerisinin olduğu saldırılardır. Taraflardan birisinin cihazında veya veri akışı üzerine konuşlanan bir saldırgan, iletilen paketleri dinleyebilir ve kaydedebilir. Basit bir saldırı yöntemidir ve aradaki bağlantıyı şifreli hale getirme ile önüne geçilebilir. TLS protokolünde de temel amaçlardan birisi budur.
- Aktif saldırı ise, saldırganın paketleri bozma, değiştirme, engelleme gibi yetkinliklerinin olduğu saldırı tipidir. Hem anahtar anlaşma protokollerine hem de TLS protokolüne yapılan saldırıların büyük çoğunluğu bu tür saldırılardır.

2.3.2.3 Farklı Saldırı Türleri

Bir saldırganın protokol üzerinde saldırı yapmasının bir çok yolu vardır. Bilinen saldırı türleri kısaca aşağıdaki gibi sınıflandırılabilir [15]:

- **Dinleme (Eavesdropping):** Dinleme, taraflar arasında aktarılan paketlerin üçüncü şahsın eline geçmesi anlamına gelir. Dinleme işlemi genellikle haberi olmadan ve fark etmeleri mümkün olmayan şekilde gerçekleşir. Internet ortamında akan verilerin dinlenmesi engellenememektedir. Ancak buna rağmen veriler şifrelenerek korunabilir. Bölüm 3.7'de olduğu gibi veri gönderilecek kişinin açık anahtarı ile şifrelenir ve böylelikle yalnızca o kişinin erişebilmesi sağlanır.

- **Değiştirme (Modification):** Bu yöntemde saldırgan gönderilmek istenen paketi durdurur ve istediği şekilde değiştirir. Değiştirmenin önüne geçebilmenin yolu veri bütünlüğünü sağlayabilecek kriptografik yöntemler kullanmaktır.

- **Yeniden gönderme (Replay):** Taraflar arasındaki paketleri kaydeden saldırgan, bu paketlerden bazılarını daha sonra tarafların birisinin yerine geçerek tekrar göndermesi durumudur. Mesaj şifreli dahi olsa geçerli bir mesaj olduğu için kabul görebilir. Bu tip saldırıların engellenebilmesi için paketlerin tazeliğinin kontrolü yapılmalıdır. Yani paketler içerisine zaman bilgisi eklenir ve olası bir saldırganın mesajı daha sonra göndermesi durumu fark edilip engellenebilir.

- **Yansıma (Reflection):** Yansıma saldırısı, iddia-karşılık (challenge-response) türü bir kimlik doğrulama yapılırken aynı protokolü karşı tarafa oynama saldırılarıdır. Amaç; iddia eden kişinin ispatlanmasını istediği bilgiyi bir başka bağlantı açıp, iddia eden kişiye yaptırmaktır. Yansıma saldırılarından kaçınmak için iddia eden tarafın kendisini ispatlamadan önce karşı tarafı ispatlamasını beklemesi gerekir.

- **Hizmet reddi saldırısı (Denial of Service Attack):** Hizmet reddi saldırısı, saldırganın sunucuya tam olarak bağlanmayıp bir çok geçersiz bağlanma isteği göndermesi ile sistemi boğması, bu şekilde sistemin hizmet veremez hale gelmesi demektir. Hizmet reddi saldırıları, sunucuların işlem kapasitelerini doldurmaya yönelik olabileceği gibi, sistem konfigürasyonuna göre oluşturabileceği bağlantı sayısını doldurmak şeklinde de yapılabilir. Bu tür saldırılar için kesin bir çözüm bulunamamaktadır. Sunucunun her bir bağlantı ile üzerinde oluşan yükü en aza indirerek verimli çalışması ile saldırının etkisi azaltılabilir.

- **Sertifika Manipülasyonu (Certificate Manipulation):** Sertifika manipülasyonu, saldırganın protokole saldırmak için bir kişinin sertifikasını değiştirip, o hali ile kendi sertifikası olarak onay alması işlemidir. Sertifika otoritesi, başvuran kişinin açık anahtarına denk gelen gizli anahtar bilgisine sahip olup olmadığını kontrol etmez ise saldırgan istediği değer için sertifika alabilir. Bu açıklık, Bölüm 3.4.2'de anlatılan saldırının gerçekleşmesine yol açar. Açıklığın önüne geçmek için başvuranın gizli anahtarının varlığı onaylanmalıdır.

Bölüm 3

Mevcut Anahtar Anlaşma Protokolleri ve Güvenlik Analizleri

3.1 Temel Diffie-Hellman Anahtar Anlaşma Protokolü

Diffie-Hellman Protokolü, güvenli olmayan ağ üzerinde (örn: Internet) iki kullanıcının ortak bir gizli anahtar üreterek, güvenli bağlantı oluşturmasını sağlayan bir protokoldür. Bu protokolün amacı, iki tarafın simetrik anahtar paylaşabilmesi ve böylece bağlantıyı dinleyen üçüncü kişilerin simetrik anahtarı ele geçirememesini sağlamaktır. Whitfield Diffie ve Martin Hellman tarafından ilk olarak 1976 da bulunan bu yöntem[27], SSL ve sonrasında TLS protokollerinde yaygın olarak kullanılmıştır. Yıllar içerisinde bu protokole yapılan saldırılar ve araştırmaların etkisiyle, hem algoritması değişmiş, hem de farklı gerçeklenmeleri ortaya çıkmıştır.

Temel Diffie-Hellman anahtar anlaşma protokolü Şekil 3.1'deki gibidir.

Ayşe		**Bora**
$a \in_r \mathbb{Z}_p^*$		$b \in_r \mathbb{Z}_p^*$
$A := g^a \mod p$		$B := g^b \mod p$
	$\xrightarrow{A}$	
	$\xleftarrow{B}$	
$K := B^a \mod p$		$K := A^b \mod p$

Şekil 3.1: Temel Diffie-Hellman Anahtar Anlaşma Protokolü

1. Ayşe ve Bora, bir p asal sayısı üzerinde anlaşırlar ve ortak parametre olarak $\mathbb{Z}_p^*$ içerisinde mertebesi q asal sayısı olan bir g sayısı belirler.

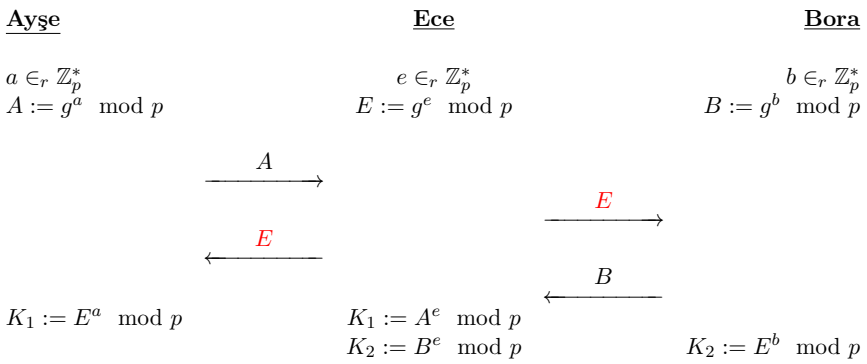
2. Ayşe yalnızca kendisinin bildiği bir a değeri oluşturur ve Bora'ya $A := g^a \mod p$ değerini hesaplayıp gönderir.
3. Bora yalnızca kendisinin bildiği bir b değeri oluşturur ve Ayşe'ye $B := g^b \mod p$ değerini hesaplayıp gönderir.
4. Ayşe, Bora'dan gelen B değerini kullanarak $K := B^a \mod p := (g^b)^a \mod p$ değerini hesaplar.
5. Bora, Ayşe'den gelen A değerini kullanarak $K := A^b \mod p := (g^a)^b \mod p$ değerini hesaplar.

Küçük sayılar kullanılarak gerçekleştirilen bir Diffie-Hellman anahtar anlaşma örneği aşağıdaki gibidir:

1. Ayşe ve Bora $p = 23$ ve $g = 5$ değerlerinde anlaşır.
2. Ayşe gizli bir $a = 7$ değerini seçer ve $A := 17 \equiv 5^7 \mod 23$ değerini hesaplayıp Bora'ya gönderir.
3. Bora gizli bir $b = 4$ değerini seçer ve $B := 4 \equiv 5^4 \mod 23$ değerini hesaplayıp Ayşe'ye gönderir.
4. Ayşe, Bora'dan gelen 4 değerini kullanarak $K := 8 \equiv 4^7 \mod 23$ değerini hesaplar.
5. Bora, Ayşe'den gelen 17 değerini kullanarak $K := 8 \equiv 17^4 \mod 23$ değerini hesaplar.
6. Elde edilen değerler birbirine eşittir ve $S = 8$ değeri Ayşe ve Bora'nın ortak gizli anahtarı oluşturmasında kullanılarak şifreli iletişim kurulur.

Yukarıdaki örneklerde dikkat edilmesi gereken nokta; Ayşe ve Bora'nın gizli olan a ve b değerlerini açıktan göndermeden aynı S değerine ulaşabilmiş olmalarıdır. Bu senaryoda açıktan paylaşılan bilgiler kullanılarak gizli anahtar elde edilemez. $S = g^{ab}$ değerini bulmak için seçilen bir $X = g^c$ değerinin S'e eşit olup olmadığının kontrolü yapılarak çözülmeye çalışılabilir. Bu yüzden temel Diffie-Hellman protokolünün güvenliği KDH problemi seviyesindedir.

Oluşan anahtarın gizliliğinin zor probleme dayanması, senaryonun güvenli olduğu ve Ayşe ile Bora arasında güvenli bağlantı oluştuğu anlamına gelmez. Bağlantıyı dinleyen ve Ayşe ile Bora'nın birbirlerine gönderdikleri değerlere müdahale edebilen bir üçüncü şahıs (Ece) olsun. Bu senaryoda Ece; Ayşe'ye karşı Bora, Bora'ya karşı Ayşe rolüne girebilir [71]. Bu yüzden temel Diffie-Hellman anahtar anlaşma protokolü ortadaki adam saldırısına karşı



Şekil 3.2: Temel Diffie-Hellman Anahtar Anlaşma Protokolüne Ortadaki Adam Saldırısı

dayanıksızdır. Aradaki mesajları engelleyip yönlendirebilen üçüncü bir şahıs Şekil 3.2'deki saldırıyı gerçekleştirebilir.

1. Bora ile konuşmak isteyen Ayşe, bir adet gizli (a) bir adet de açık anahtar (A) oluşturur ve bu açık değeri Bora'ya gönderir.
2. Ece, Ayşe'den Bora'ya giden mesajı araya girerek engeller. Onun yerine kendisi bir adet gizli (e) bir adet de açık anahtar (E) oluşturmuştur ve kendi açık anahtarını Bora'ya gönderir.
3. Bora cevap olarak B değerini Ece'ye gönderir.
4. Ece, Bora'nın açık anahtarı yerine kendi açık anahtarı E'yi Ayşe'ye gönderir.
5. Ayşe Ece'den gelen E mesajını kullanarak K_1 ortak anahtarını oluşturur.
6. Ece Ayşe'dan gelen A mesajını kullanarak K_1 ortak anahtarını oluşturur.
7. Ece Bora'dan gelen B mesajını kullanarak K_2 ortak anahtarını oluşturur.
8. Bora Ece'den gelen E mesajını kullanarak K_2 ortak anahtarını oluşturur.
9. Ece, Ayşe ile Bora arasındaki trafiği açıp dinleyip tekrar şifreleyerek karşı tarafa iletebilir. Çünkü her iki taraf ile şifre paylaşan kişi Ece olmuştur.

Yukarıdaki örnekte de görüldüğü gibi, Ece, iki tarafla iki farklı anahtar anlaşması yaparak Ayşe ve Bora'nın güvenli bir bağlantı oluşturamamasına neden olmuştur. Bu saldırının temel nedeni, Ayşe ve Bora'nın birbirleri ile konuşup konuşmadıklarını kontrol edebilecekleri bir mekanizma bulunmamasıdır. Eğer taraflar karşısındaki kişiyi doğrulayabilselerdi, araya giren bir kişi olduğu tespit edilebilirdi. Bu ihtiyaçtan ötürü sertifikalı Diffie-Hellman anahtar anlaşma protokolü ortaya çıkmıştır (sertifikalar için bkz. Bölüm 2.1).

3.2 Sertifikalı Diffie-Hellman Anahtar Anlaşma Protokolü

Bu yöntem, Ayşe ile Bora'nın karşı tarafa, kimliklerini ispat etmelerini sağlar. Bunu yapmak için öncelikle Ayşe ve Bora, bir sertifika otoritesinden sertifika alır ve iletişime başlarken birbirlerine bu sertifikaları gönderirler. Elde edilen bu sertifikalar muhatap olunan kişinin ispatını sağlar. Eğer üçüncü bir şahıs (Ece) araya girerek bağlantıyı değiştirirse, sertifikalar bozulmuş olur ve bağlantı iptal edilerek üçüncü şahsın iki tarafla farklı birer bağlantı kurması engellenmiş olur [26]. Sertifikalı Diffie-Hellman anahtar anlaşma protokolü Şekil 3.3'deki gibidir.

1. Ayşe ve Bora, güvenli haberleşme yapabilmek için öncelikle bir anahtar çifti oluşturur (Ayşe = (A,a) ve Bora = (B,b)). Bu işlem bir defa yapılır ve bundan sonra kuracakları iletişimlerde sertifika geçerli olduğu sürece A ve B değerleri kullanılır.

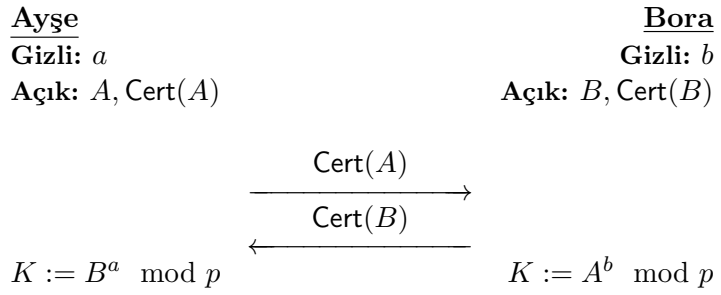
2. Daha sonra, A ve B değerlerini güvendikleri bir sertifika otoritesine imzalatırlar. Bu işlem sertifika otoritesinin gizli anahtarı ile yapılır ve bu işlemin doğruluğu sertifika otoritesinin açık anahtarı ile test edilir.

3. Ayşe, Bora ile iletişime geçmek için Bora'ya kendi sertifikasını $\mathsf{Cert}(\mathrm{A})$ gönderir. Bu sertifika aynı zamanda Ayşe'nin açık anahtarı olan A değerini de içermektedir.

4. Bora da Ayşe'ye açık anahtarının bulunduğu kendi sertifikasını gönderir $\mathsf{Cert}(\mathrm{B})$.

5. Ayşe Bora'nın sertifikasını, Bora da Ayşe'nin sertifikasını ilgili sertifika otoritesine ait açık anahtar ile doğrularlar. Bu işlemin başarılı olması Ayşe ile Bora'nın arasında Ece'nin bulunmadığını ispatlar.

6. Sonrasında Ayşe $K := B^a \mod p$ değerini, Bora'da $K := A^b \mod p$ değerini hesaplar ve taraflar arasında ortak gizli değerde anlaşılmış olur.



Şekil 3.3: Sertifikalı Diffie-Hellman Anahtar Anlaşma Protokolü (Cert(A),Cert(B) sırayla Ayşe ve Bora'nın sertifikalarıdır.)

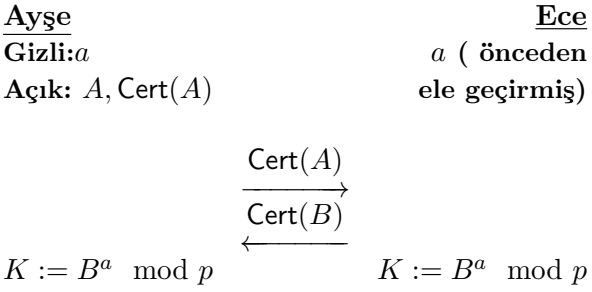
Sertifikalı Diffie-Hellman anahtar anlaşma protokolü teorik olarak üçüncü şahısların güvenli bağlantıyı dinlemesini engellemiş olsa da, uzun vadede ortaya çıkan diğer sorunlara

cevap verememiştir. Çünkü, güvenli iletişimde en kritik nokta önceden anlaşılmış bir anahtarı olmayan ve birbirlerini tanımayan tarafların bulunduğu ağ ortamında kimlikleri doğrulanarak ve gizli bir şekilde anahtar anlaşmasının yapılmasını sağlamaktır. Bununla birlikte ileriye yönelik gizliğin sağlanması, bilinen anahtar saldırısı ve anahtar kontrolüne dayanıklılık gibi gereklilikler de ortaya çıkmıştır.

3.2.1 Anahtar İfşası ile Yerine Geçme (KCI) Saldırısı

Sertifikalı Diffie-Hellman anahtar anlaşma protokolünde, kişilerin kendilerini birer sertifika ile ispatlıyor olmaları, her iletişimlerinde aynı açık ve gizli anahtar çiftini kullanmalarını gerekli hale getirmektedir. Bunun anlamı; herhangi bir anda anahtar anlaşması yapan Ayşe ve Bora, daha sonra yapacakları ve daha önce yaptıkları anahtar anlaşmalarında elde ettiklerinden farklı bir ortak anahtar elde etmeyeceklerdir. Daha genel bir ifade ile bu iki taraftan birisinin sertifikası yenilenmedikçe, oluşan bağlantı her zaman "$g^{ab} \mod p$" değerine eşit olacaktır. Eğer taraflardan birisinin gizli anahtarı (a ya da b) üçüncü bir şahıs tarafından ele geçirilirse ve bu şahıs Ayşe ile Bora arasındaki bağlantıları kaydetmişse, bir çok oturuma ait bilgileri ele geçirebilir. Bununla birlikte bir kişinin gizli anahtarının ele geçirilmesi durumunda saldırganın o kişinin yerine geçmesi mümkün olabilir; fakat bir kişi gizli anahtarını kaybettiğinde saldırganın gizli anahtarını ele geçirdiği kişinin konuşmak istediği kişi kılığına girmesi farklı bir yaklaşım gerektirir. Bu tip bir saldırıya yerine geçme saldırısı adı verilir [48].

Avusturyalı bir çalışma ekibi tarafından bulunan ve gizli anahtarın ifşası ile yerine geçme saldırısının mümkün olduğunu ortaya koyan KCI saldırısı, basit birkaç adım ile saldırının pratikte de mümkün olduğunu gösterdi [48]. Anahtar ifşası ile yerine geçme saldırısı Şekil 3.4'deki gibidir.



Şekil 3.4: Anahtar İfşası ile Yerine Geçme Saldırısı (Cert(A),Cert(B) sırayla Ayşe ve Bora'nın sertifikalarıdır.)

1. Ayşe Bora'ya açık anahtarının bulunduğu sertifikayı gönderir.
2. Ece Ayşe'nin mesajını keser ve kendisi ile konuşmasını sağlar. Bunu yaparken Ayşe'nin farketmemesi için de Bora'nın sertifikasını cevap olarak gönderir.

3. Ayşe Bora'nın sertifikasını aldığı için muhatabını Bora sanır ve sertifikada bulunan B değerini kullanarak $K := B^a$ ortak anahtarı oluşturur.

4. Ece, Bora'nın gizli anahtarı olan b değerini bilmediği için $K := A^b \mod p$ işlemini yapamaz fakat, Ayşe'nin gizli anahtarı olan a değerini bildiği için $K := B^a$ yaparak ortak anahtarı oluşturur.

5. Ayşe, Bora ile konuştuğunu düşünerek Ece ile konuşmaya başlar.

Sertifikalı yöntemde ortaya çıkan bu eksikliği gidermek için, kısa ömürlü anahtarlar kullanılmasını sağlayan yöntemler geliştirilmiştir.

3.3 Kısa Ömürlü (Ephemeral) Diffie-Hellman Anahtar Anlaşma Protokolü

Sertifikalı Diffie-Hellman algoritmasının yerine geçme saldırısına karşı dayanıklı olabilmesi ve ileriye yönelik gizlilik sağlanabilmesi için kısa ömürlü (Ephemeral) Diffie-Hellman yöntemi ortaya çıkmıştır. Kısa ömürlü Diffie-Hellman anahtar anlaşma protokolü temel olarak, hem ileriye yönelik hem de geriye yönelik gizliliğin ihlal edilmemesini sağlar. Bu yöntemde her bir oturum için oluşturulan gizli değerlere rastgele değerler de eklenerek her bir oturum için farklı anahtarlar elde edilir. Bu yöntem taraflar arasındaki işlem yükünü artırıyor olsa da, uzun vadede gizlilik ve güvenlik sağladığı için başarılı bir yöntemdir. Kısa ömürlü Diffie-Hellman anahtar anlaşma protokolü Şekil 3.5'deki gibidir.

Ayşe		**Bora**
Gizli: a		**Gizli:** b
Açık: $A, \mathsf{Cert}(A)$		**Açık:** $B, \mathsf{Cert}(B)$
$x \in_r \mathbb{Z}_p^*$		$y \in_r \mathbb{Z}_p^*$
$X := g^x \mod p$		$Y := g^y \mod p$
	$\mathsf{Cert}(A), X \longrightarrow$	
	$\mathsf{Cert}(B), Y \longleftarrow$	
$K := Y^a B^x = g^{(ya+bx)}$		$K := X^b A^y = g^{(xb+ay)}$

Şekil 3.5: Kısa Ömürlü Diffie-Hellman Anahtar Anlaşma Protokolü

1. Ayşe, rastgele bir x değeri oluşturur ve $X = g^x$ değerini hesaplar. X değerini sertifikası ile birlikte Bora'ya gönderir. Burada, x değeri Ayşe'nin kısa ömürlü gizli anahtarı, X değeri kısa ömürlü açık anahtarıdır.

2. Bora, rastgele bir y değeri oluşturur ve $Y = g^y$ değerini hesaplar. Y değerini sertifikası ile birlikte Ayşe'ye gönderir. Burada, y değeri Bora'nın kısa ömürlü gizli anahtarı, Y değeri kısa ömürlü açık anahtarıdır.

3. Ayşe, Bora'dan gelen B ve Y değerlerini kullanarak $K = Y^a \cdot B^x = g^{(ya+bx)}$ değerini hesaplar.

4. Bora, Ayşe'den gelen A ve X değerlerini kullanarak $K = X^b \cdot A^y = g^{(xb+ay)}$ değerini hesaplar.

5. Taraflar paylaşmış oldukları bu ortak giz değerini kullanarak oturum anahtarını belirler ve güvenli iletişime geçerler.

Kısa ömürlü değerler ile yapılan bu anahtar anlaşması, her bir bağlantıda farklı x ve y değerlerinin kullanılması ile farklı oturum anahtarları sağlar. Bu şekilde uzun vadede gizlilik sağlanır. Ayrıca rastgele olan bu x ve y değerleri sayesinde, Ayşe veya Bora'nın gizli anahtarları ele geçirilmiş olsa dahi, oturum bilgileri ele geçirilemeyecektir. Bu yöntemde de, sertifikalı anahtar anlaşmasında olduğu gibi önce sertifikalar gönderilerek kişilerin doğruluğu sağlanır.

3.4 MQV Anahtar Anlaşma Protokolü

MQV, Diffie-Hellman temelli bir anahtar anlaşma protokolüdür. 1995 yılında Menezes, Qu ve Vanstone tarafından geliştirilmiştir [73]. İlerleyen yıllarda Law ve Solinas tarafından yapılan modifiyelerle, yetkilendirme sağlayan etkin bir algoritma olarak kabul görmeye başlamıştır [64]. Anahtar anlaşma protokollerinden beklenen birtakım güvenlik kıstasları vardır. Bunların temelinde, protokolün hem aktif hem de pasif saldırılara karşı dayanıklı olması beklentisi vardır. Bununla birlikte, güvenli bir protokolün aşağıdaki güvenlik özelliklerine de sahip olması beklenmektedir [64]:

1. Taraflar arasındaki her oturumda farklı bir anahtar oluşturulmalıdır. Böylece oturum anahtarı üçüncü şahıslar tarafından ele geçirilse bile yalnızca o oturum bilgileri ifşa olur.

2. Eğer taraflardan birisinin gizli anahtarı ifşa olursa, eskiye yönelik oturumların gizliliği tehlikeye girmemelidir.

3. Anahtar ifşası ile yerine geçme saldırısına karşı dayanıklı olunmalıdır (bkz. Bölüm 3.2.1). Eğer bir kişinin gizli anahtarı ele geçirilirse, bu bilgi sayesinde o kişinin iletişim kurmak istediği tarafın yerine geçilememelidir.

4. Tarafların birbirleri ile oluşturdukları güvenli bağlantıyı doğrulayabilmeleri gerekmektedir. Bilgi paylaşımı yapılmadan önce istenilen muhatap ile bağlantı kurulduğu kesinleştirilmelidir.

5. Taraflar, oluşturulacak anahtarı önceden belirlenmiş bir değer yapmaya zorlayamamalıdır.

6. Gerekli durumlarda sunucu da istemcinin kimliğini doğrulama ihtiyacı duyabilir. Bu özelliği sağlayan bir algoritma olmalıdır.

MQV protokolü, yukarıda bahsedilen özellikleri sağlayabilmek amacıyla geliştirilmiş bir protokoldür. Uygulanan adımlar olarak Diffie-Hellman algoritmasına dayanır. Temel adımlara ek olarak, kimlik doğrulama ve kısa ömürlü değerler kullanarak her seferinde farklı anahtarlar oluşturma gibi yaklaşımları ile internet güvenliği konusunda büyük etkileri olmuştur. MQV anahtar anlaşma protokolü Şekil 3.6'da özetlenmiştir.

Ayşe		**Bora**
Gizli: a		**Gizli:** b
Açık: $A, \mathsf{Cert}(A)$		**Açık:** $B, \mathsf{Cert}(B)$
$x \in_r \mathbb{Z}_p^*$		$y \in_r \mathbb{Z}_p^*$
$X := g^x \mod p$		$Y := g^y \mod p$
	$\xrightarrow{Cert_A, X}$	
	$\xleftarrow{Cert_B, Y}$	
$\bar{X} := X \mod 2^l + 2^l$		$\bar{X} := X \mod 2^l + 2^l$
$\bar{Y} := Y \mod 2^l + 2^l$		$\bar{Y} := Y \mod 2^l + 2^l$
$S_A := (x + \bar{X}a) \mod p$		$S_B := (y + \bar{Y}b) \mod p$
$t_A := YB^{\bar{Y}} \mod p$		$t_B := XA^{\bar{X}} \mod p$
$K := (t_A)^{S_A}$		$K := (t_B)^{S_B}$

Şekil 3.6: MQV Anahtar Anlaşma Protokolü ($l = |p/2|$)

l değeri, p değerinin bit uzunluğunun yarısıdır. $\bar{Y}$ değeri, Y değerinin en sağdaki l bit uzunluğundaki değerinin alınması ile oluşur. Bu işlemin yapılmasının sebebi, $t_A = YB^{\bar{Y}} \mod p$ ve $t_B = XA^{\bar{X}} \mod p$ işlemlerinin maliyetini düşürmektir. $S_A = S_B$ eşitliğinin nasıl sağlandığı Şekil 3.7'de görülebilir.

$$\begin{aligned} YB^{\bar{Y}} &\equiv g^y \cdot (g^b)^{\bar{Y}} \mod p \\ &\equiv g^{(y+b\bar{Y})} \mod p \\ &\equiv g^{S_B} \mod p \end{aligned} \qquad \begin{aligned} XA^{\bar{X}} &\equiv g^x \cdot (g^a)^{\bar{X}} \mod p \\ &\equiv g^{(x+a\bar{X})} \mod p \\ &\equiv g^{S_A} \mod p \end{aligned}$$

$$\begin{aligned} K_A &:= (YB^{\bar{Y}})^{S_A} \\ &:= g^{S_B \cdot S_A} \end{aligned} \qquad \begin{aligned} K_B &:= (XA^{\bar{X}})^{S_B} \\ &:= g^{S_A \cdot S_B} \end{aligned}$$

Şekil 3.7: MQV Ortak Anahtar İspatı

3.4.1 MQV Protokolünün Eksiklikleri

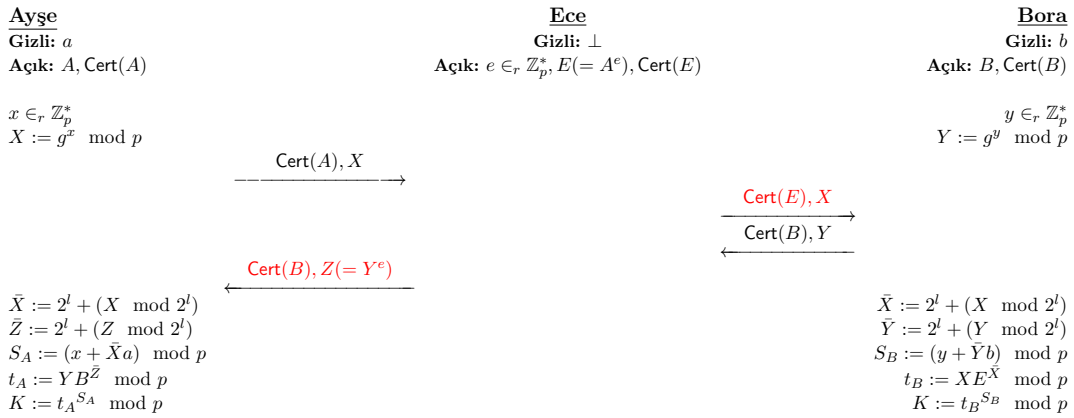
MQV protokolünde iddia edildiğine göre [73], bilinen anahtar güvenliği, ileriye yönelik gizlilik, anahtar ifşası ile yerine geçme ve anahtar kontrolü konularında gereksinimler sağlanmıştır. Ancak, MQV protokolü iddia ettiği özelliklerden birkaçını ve farklı saldırılara karşı savunmasını sağlayamamaktadır. Bu açıklıkların bazıları aşağıdaki gibidir:

- **Bilinmeyen Anahtar Paylaşımı Saldırısı (UKS Attack):** MQV protokolü, UKS saldırısına karşı güvensizdir. MQV protokolünün UKS saldırısına karşı koyabilmesi için tarafların gizli anahtarlara sahip olduklarını ispat etmesi(Proof of Possession) yöntemi düşünülmüştür. Ancak; Kaliski tarafından gösterildiği üzere [55], bu ispat olsa dahi MQV protokolü UKS saldırısına karşı güvensiz olmaktadır.
- **Mükemmel İleriye Yönelik Gizlilik (Perfect Forward Secrecy):** MQV protokolü mükemmel ileriye yönelik gizlilik sağlamamaktadır. Bu sorun yalnızca MQV protokolüne ait bir sorun değildir. Aşikar olmayan kimlik doğrulama yapan ve 2 mesajlı mekanizmaya sahip (HMQV dahil) protokollerin genel sorunudur.
- **Anahtar İfşası ile Yerine Geçme Saldırısı (KCI Attack):** MQV protokolü, standart KCI saldırısına dayanıklıdır. Ancak; taraflardan birisinin gizli anahtarını ve diğerinin ephemeral değerini bilen güçlü bir saldırgan KCI saldırısını gerçekleştirebilir. [59].

Yukarıda bahsedilen eksiklikler ve birtakım diğer eksikliklerden en önemlisi "Bilinmeyen Anahtar Paylaşımı" saldırısıdır.

3.4.2 Bilinmeyen Anahtar Paylaşımı (UKS) Saldırısı

UKS saldırısı, Ayşe ve Bora arasında anahtar paylaşımı yapılması sonrasında Ayşe'nin Bora ile konuştuğunu bilmesi fakat Bora'nın Ece ile konuştuğunu sanmasına neden olan bir saldırıdır. Burada saldırgan Ece, arada paylaşılan K anahtarını bilmemektedir ve buna rağmen Bora'yı kendisi ile konuştuğuna kandırmaktadır. Bu saldırı kaynak değiştirme saldırısı (source substitution attack) olarak da bilinir. Saldırının başarılı olması durumunda Ayşe'nin Bora'ya gönderdiği gizli mesajı, Bora Ayşe'den değil Ece'den geliyor şeklinde yorumladığı için karmaşaya yol açar. Örneğin; Ayşe Bora'ya göre yetkili birisi ise ve mesaj içeriğinde emir bulunuyorsa, Bora bu mesajı Ece'den geldi diye düşünmesinden ötürü umursamayabilir. Bilinmeyen anahtar paylaşımı saldırısı Şekil 3.8'deki gibi özetlenmiştir:



Şekil 3.8: Bilinmeyen Anahtar Paylaşımı Saldırısı ($l = |p/2|$)

3.5 HMQV Anahtar Anlaşma Protokolü

HMQV protokolü; MQV protokolünde ortaya çıkan eksiklikleri gidermek amacı ile ortaya çıkmıştır. Ancak, HMQV protokolünün 2 mesajlı bir protokol olmasından ötürü mükemmel ileriye yönelik gizliliği tam olarak sağlayamamaktadır. HMQV protokolünde temel fark, kısa ömürlü değerlerin diğer değerlerle birlikte özetinin alınarak gönderilmesi işlemidir. Bu işlem sayesinde MQV'nin maruz kaldığı grup saldırısı tehlikesinin yanı sıra, UKS ve diğer yerine geçme saldırılarına karşı da dayanıklı hale gelmiştir. Bununla birlikte HMQV, sağladığı artıları performans kaybı yaşamadan ortaya koyabilmektedir.

MQV algoritmasının göz önüne almadığı bir önemli açıklık da, kısa ömürlü gizli x, y değerlerin ifşasına karşı dayanıklılıktır. Oturuma özel oluşturulan kısa ömürlü değerler sistemin performansını yükseltmek adına anlık olarak üretilmeyip, işlem yoğunluğu düşük zamanlarda oluşturularak kaydedilmektedir. Bu, düşük güçteki cihazlar için olduğu kadar büyük sistemlerde ihtiyaç duyulan bir özelliktir. Bu değerler, uzun süreli gizli anahtarlara göre daha kolay erişilebilir haldedirler. Çünkü uzun süreli gizli anahtar genellikle güvenli depolama modülü gibi cihazlarda korunaklı olarak tutulmaktadır. Buna karşın kısa ömürlü değerler çoğunlukla hafıza (RAM) üzerinde tutulurlar ve zararlı yazılımların etkisine daha kolay maruz kalırlar. HMQV protokolü, bu değerlerin ifşası durumunda dahi oturum güvenliğini sağlamaktadır [59].

HMQV ile MQV arasındaki teknik detay farkı ise $\bar{X}$ ve $\bar{Y}$ değerlerinin hesaplanmasındaki farklılıktadır. MQV'de $\bar{X} = X \mod 2^l + 2^l$ ve $\bar{Y} = Y \mod 2^l + 2^l$ şeklinde hesaplanan değer, HMQV de $\bar{X} = H(X||B)$ ve $\bar{Y} = H(Y||A)$ olarak hesaplanmasıdır. HMQV anahtar anlaşma protokolü Şekil 3.9'de verilmiştir.

Ayşe		**Bora**
Gizli: a		**Gizli:** b
Açık: $A, \mathsf{Cert}(A)$		**Açık:** $B, \mathsf{Cert}(B)$
$x \in_r \mathbb{Z}_p^*$		$y \in_r \mathbb{Z}_p^*$
$X := g^x \mod p$		$Y := g^y \mod p$
	$\xrightarrow{Cert_A, X}$	
	$\xleftarrow{Cert_B, Y}$	
$\bar{X} := H(X\|\|B)$		$\bar{X} := H(X\|\|B)$
$\bar{Y} := H(Y\|\|A)$		$\bar{Y} := H(Y\|\|A)$
$s_A := x + \bar{X} \cdot a \mod p$		$s_B = y + \bar{Y} \cdot b \mod p$
$K := H((Y \cdot B^{\bar{Y}})^{s_A})$		$K := H((X \cdot A^{\bar{X}})^{s_B})$

Şekil 3.9: HMQV Anahtar Anlaşma Protokolü ($H(\cdot)$ güvenli bir özet fonksiyondur.)

3.6 FHMQV Anahtar Anlaşma Protokolü

FHMQV (Fully Hashed Menezes Qu Vanstone) protokolü, MQV ve HMQV protokollerinin geliştirilmiş halidir [85]. HMQV protokolünde oturuma dair gizli bilgilerin ifşası durumunda oluşabilecek yerine geçme ve ortadaki adam saldırıları mümkün hale gelmektedir. FHMQV protokolü temel farklılık olarak tüm mesajların özet değerlerini de mesajla birlikte gönderir ve bu şekilde herhangi bir aşamada müdahale aktif saldırı olduğunda farkedilmesine imkan tanır. FHMQV anahtar anlaşma protokolü Şekil 3.10'de verilmiştir.

Ayşe		**Bora**
Gizli: a		**Gizli:** b
Açık: $A, \mathsf{Cert}(A)$		**Açık:** $B, \mathsf{Cert}(B)$
$x \in_r \mathbb{Z}_p^*$		$y \in_r \mathbb{Z}_p^*$
$X := g^x \mod p$		$Y := g^y \mod p$
	$\xrightarrow{Cert_A, X}$	
	$\xleftarrow{Cert_B, Y}$	
$\bar{X} := H(X\|\|Y\|\|A\|\|B)$		$\bar{X} := H(X\|\|Y\|\|A\|\|B)$
$\bar{Y} := H(Y\|\|X\|\|A\|\|B)$		$\bar{Y} := H(Y\|\|X\|\|A\|\|B)$
$s_A := x + \bar{X} \cdot a \mod p$		$s_B = y + \bar{Y} \cdot b \mod p$
$K := H((Y \cdot B^{\bar{Y}})^{s_A}\|\|A\|\|B\|\|X\|\|Y)$		$K := H((X \cdot A^{\bar{X}})^{s_B}\|\|A\|\|B\|\|X\|\|Y)$

Şekil 3.10: FHMQV Anahtar Anlaşma Protokolü ($H(\cdot)$ güvenli bir özet fonksiyondur.)

3.7 RSA Tabanlı Anahtar Anlaşma Protokolü

RSA algoritması üç temel bileşenden oluşur. Açık bir N değeri, açık üs değeri e ve gizli üs değeri d. N değeri iki adet p ve q asal sayılarının çarpımından oluşan bir değerdir

($N = p \cdot q$). Yapılan tüm işlemler modüler uzayda yapılmaktadır. Bunun anlamı her işlem sonrasında N den büyük bir değerin N ye bölünüp, kalanının alınması demektir.

Seçilen herhangi bir M değeri için ($M < N$ değerini sağlayan), aşağıdaki işlem her zaman doğru çıkmaktadır:

$$M \equiv M^{d \cdot e} \mod p \equiv M^{e \cdot d}$$

Burada N ve e değerleri açık, d değeri ise gizlidir. Anahtar oluşturma safhasında gizli değeri üretmek için aşağıdaki denklik kullanılarak bilinen e, p ve q değerlerine karşılık gelen d değeri hesaplanır:

$$d \cdot e \equiv 1 \mod (p-1)(q-1)$$

Bu hesaplamanın doğruluğu, Çinli Kalanlar Teoremi'ne ve Fermat'ın Küçük Teoremi'ne dayanmaktadır [16]. Bu hesaplamayı yapabilmek için p ve q değerlerini bilmek gerekir. Yani; gizli d değerinin güvenliği, gizli p ve q değerlerine dayanmaktadır. p ve q değerleri de açık bir değer olan N değerinin çarpanları olduğu için, RSA algoritmasının zorluğu çarpanlara ayırma problemine dayanmaktadır. RSA algoritması ile temel olarak şifreleme aşağıdaki gibidir:

1. Bora, Ayşe'ye gizli bir mesaj göndermek ister. Bunun için Ayşe'nin açık anahtarına (N, e değerlerine) ihtiyaç duyar.
2. Bora, göndermek istediği M mesajını $E \equiv M^e \mod p$ işlemi ile şifreler ve Ayşe'ye gönderir.
3. Ayşe, Bora'dan aldığı şifreli E mesajını açmak için $M_D \equiv E^d \mod p$ işlemini yapar.
4. $M_D \equiv (M^e)^d \equiv M$ olduğundan ötürü şifreli mesaj açıldığında, şifrelenmeden önceki mesajla aynı olmaktadır.

RSA algoritması ile temel olarak imzalama işlemi aşağıdaki gibidir:

1. Ayşe M mesajını imzalayıp Bora'ya göndermek istiyor. Böylece Bora, mesajın Ayşe'den geldiğinden emin olabilecek.
2. Ayşe $S \equiv M^d \mod p$ değerini hesaplar ve S değerini Bora'ya gönderir.

3. Bora Ayşe'den aldığı S mesajını doğrulamak için Ayşenin açık anahtarı olan e değerini kullanarak $M_V \equiv S^e \mod p$ değerini hesaplar.

4. $M_V \equiv M^{d \cdot e} = M$ olduğundan ötürü Bora S mesajının Ayşe'den geldiğinden emin olur.

Klasik RSA algoritması, güvenlik açısından sorunlar barındırmaktadır. Bir saldırganın M_1 ve M_2 mesajlarına ve $S(M_1)$ ve $S(M_2)$ imzalı mesajlarına erişebildiğini varsayalım. RSA algoritmasının yapısından ötürü, $S(M_1) \cdot S(M_2) \mod p$ değeri $S(M_1 \cdot M_2)$ metninin imzası anlamına gelmektedir. Böylece iki metine imza atan bir kişi o iki metnin birleşimine de imza atmış gibi görülür. Buna, RSA'nın çarpım özelliği denir [20]. Bir diğer sorun da; dolgulama yapılmadan bir mesajın şifrelenmesi işlemi, matematiksel olarak imzalama işleminin tam tersidir. Bir kişinin hem imzalama hem de şifreleme için aynı açık-gizli anahtar çiftini kullanması durumunda, saldırgan ilgili kişiye o kişinin açık anahtari ile şifrelenmiş bir metni imzalaması için verir. Kurbanın verilen metni imzalaması, şifreli metni çözüp saldırgana vermesi anlamına gelir. Klasik RSA algoritmasında bulunan açıklıkların giderilmesi için birtakım yaklaşımlar geliştirilmiştir. Bunların başında rastsallaştırma ve tuzlama özellikleri gelir.

3.8 Rastsallaştırma-Tuzlama

Rastsallaştırma, aynı girdi ile farklı çıktıların elde edilebilmesini sağlayan bir algoritmik yaklaşımdır. Bu sayede sözlük saldırılarının önüne geçilmiş olur [88]. Şifreli verilerin içerebileceği değerlerin kısıtlı olduğu durumlarda saldırganın şifreli veriler üzerinden bilgi edinmesi rastsallaştırma ile engellenmiş olur.

Ancak, PKCS #1 formatında özet alıp imzalama işleminin deterministik olmasından ötürü rastgeleleştirmek için tuzlama yöntemi ortaya çıktı. Özet alınmadan önce eklenen rastgele bir değer olan tuz, bir kerelik kullanımından ötürü nonce (tek seferlik sayı) olarak da isimlendirilmektedir.

Bölüm 4

SSL/TLS Protokolü

TLS (önceki ismiyle SSL) prookolleri, güvensiz ortamda iletişim güvenliği sağlayabilmek için geliştirilmiş protokollerdir. TLS (Transport Layer Security - Taşıma Katmanı Güvenliği) protokolü, ismini taşıma katmanı üzerinde bulunmasından ötürü almıştır. Günümüzde çoğunlukla web tarayıcısı ve web siteleri arasında kurulan bağlantılarda kullanılır. Temel anlamda TLS kullanımındaki amaç taraflar arasında güvenli haberleşmenin sağlanmasıdır. Bu çalışmada, mevcut versiyon olarak TLSv1.2 incelenmiştir. TLSv1.3 versiyonu son hali ile gerçeklenmemiştir. TLS protokolü, kendi içinde bir çok protokol ve kriptografik yöntem barındırmaktadır. Genel olarak bu başlıklar aşağıda belirtilmiştir.

4.1 Kayıt Protokolü

TLS protokolü, veri iletişiminin uygulama katmanlarından alt katmanlara iletilmesi için kayıt protokolünü kullanmaktadır. Kayıt Protokolü iki temel kısımdan oluşur. İlk kısım, kayıt içerisinde bulunan veri ve kayıt hakkında bilgilerin bulunduğu başlık kısmıdır. İkinci kısımda ise iletilmek istenen veriler vardır [25]. Başlığın ilk baytında, içerik tipi bulunur. İçerik tipine göre paketin içindeki verilerin türü anlaşılır. İçerik tipi yapısı ve değerleri aşağıdaki gibidir:

```
enum {
        change_cipher_spec (20),
        alert (21),
        handshake (22),
        application_data (23)
} ContentType;
```

20 = 0x14 = ChangeCipherSpecs Mesajı
21 = 0x15 = Uyarı Mesajı

22 = 0x16 = El sıkışma Mesajı
23 = 0x17 = Uygulama Verileri

Sonraki iki bayt, versiyon baytlarıdır. Paketin desteklediği en alt ve en üst SSL/TLS versiyonu bilgileri yer alır. Yapısı aşağıdaki gibidir:

```
struct {
        uint8 major;
        uint8 minor;
} ProtocolVersion;
```

Sonraki iki bayt kayıt uzunluğu bilgisini içerir. Kayıt uzunluğu kadar veri bilgisi de takip eden baytlarda bulunur. TLS protokolü kayıt yapısı aşağıdaki gibidir:

```
struct {
        ContentType type;
        ProtocolVersion version;
        uint16 length;
        opaque fragment[TLSPlaintext.length];
        }TLSPlaintext;
```

4.2 Uyarı Protokolü

TLS'in alt protokollerinden birisi olan uyarı protokolü, iletişim sırasında tarafların birbirlerine olası uyarıları ve bu uyarıların seviyesini belirtebilmesini sağlar. Kritik seviyede gönderilen mesajlar bağlantının anında sonlandırılması demektir. Uyarı mesajları mevcut iletişimde ortak anahtar olması halinde şifrelenerek ve sıkıştırılarak gönderilir. Uyarı mesajarının yapısı aşağıdaki gibidir:

```
enum { warning(1), fatal(2), (255) } AlertLevel;

enum {
        close_notify(0),
        unexpected_message(10),
        bad_record_mac(20),
        decryption_failed_RESERVED(21),
        record_overflow(22),
        decompression_failure(30),
        handshake_failure(40),
        no_certificate_RESERVED(41),
        bad_certificate(42),
        unsupported_certificate(43),
        certificate_revoked(44),
```

```
        certificate_expired(45),
        certificate_unknown(46),
        illegal_parameter(47),
        unknown_ca(48),
        access_denied(49),
        decode_error(50),
        decrypt_error(51),
         export_restriction_RESERVED(60),
        protocol_version(70),
        insufficient_security(71),
        internal_error(80),
        user_canceled(90),
        no_renegotiation(100),
        unsupported_extension(110),
(255)
} AlertDescription;

struct {
        AlertLevel level;
        AlertDescription description;
} Alert;
```

4.3 El Sıkışma Protokolü

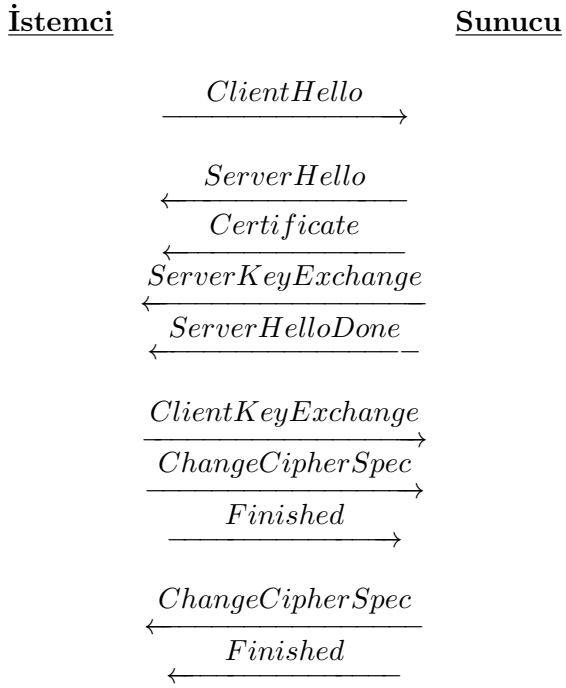
El sıkışma(Handshake) protokol adımı, SSL/TLS protokolündeki en hayati adımdır. Tarafların arasında hedeflenen güvenli bağlantı kurulumu bu protokol ile sağlanır. Kurulacak bağlantı tipine göre 6 ile 10 arası mesaj alışverişi yapılır. El sıkışma işlemi, kullanıma göre sertifika istenen ya da sertifika istenmeyen bir bağlantı olabilir. Her TLS bağlantısı el sıkışma ile başlar. Eğer istemci, sunucu ile daha önce bir oturum kurmamış ise, tam el sıkışma adı verilen senaryo gerçekleşir. TLS el sıkışma adımları aşağıdaki gibidir:

- Tarafların hello mesajları ile kullanılacak algoritmaları ve rastgele değerleri paylaşmaları sağlanır. Bununla birlikte bir oturumu tekrar ayağa kaldırmak için karşı taraf kontrol edilebilir.
- Tarafların ön ana giz (pre-master secret) oluşturabilmesi için gerekli kriptografik parametreler paylaşabilmesi sağlanır.
- Paylaşılan kriptografik bilgilerle ve sertifikalarla tarafların birbirlerini doğrulamaları sağlanır.
- *Hello* mesajları ile iletilen rastgele değer ve ön ana giz değeri kullanılarak ana giz (master secret) oluşturulur.

- Tarafların oluşturdukları bağlantının istenen güvenlik parametrelerine sahip olup olmadığı ve bağlantının bir saldırgan tarafından bozulup bozulmadığı kontrol edilir.

El sıkışma mesajının yapısı aşağıdaki gibidir:

```
struct {
        HandshakeType msg\_type;
        uint24 length;
        HandshakeMessage message;
} Handshake;
```



Şekil 4.1: El Sıkışma Senaryosu

4.3.1 Hello Mesajları

Hello mesajları sunucu ile istemcinin güvenlik isterlerini paylaştıkları mesajlardır. Yeni bir oturum başlarken kullanılacak şifreleme mekanizması, özet almak için kullanılacak algoritmalar ve sıkıştırma algoritması değerleri bulunur. Hello mesajları sayesinde bu değerler belirlenir.

4.3.1.1 Hello Request

Hello Request (Merhaba İsteği) mesajı sunucu tarafından istemciye gönderilen ve istemciye *ClientHello* mesajını gönderebileceğini bildiren mesajdır. Bu mesaj içerisinde herhangi bir içerik bulunmayıp yalnızca aşağıdaki gibi bir yapıdadır:

```
struct{ } HelloRequest;
```

4.3.1.2 Client Hello

ClientHello (İstemci Merhaba) mesajı, istemcinin sunucu ile iletişime başlamak için göndermesi gereken ilk mesajdır. *ClientHello* mesajı içerisinde aşağıdaki değerler bulunur [25]:

```
struct {
        ProtocolVersion client_version;
        Random random;
        SessionID session_id;
        CipherSuite cipher_suites<2..2^16-2>;
        CompressionMethod compression_methods<1..2^8-1>;
        select (extensions_present) {
                case false:
                struct {};
                case true:
                Extension extensions<0..2^16-1>;
        };
} ClientHello;
```

- **ProtocolVersion:** İstemcinin oturum için kullanacağı SSL/TLS versiyonlarından hangisini kullandığı bilgisi bulunur.
- **Random:** Protokolde daha sonraki aşamalarda kullanılacak olan rastgele bir değer bulunur. SSL/TLS bağlantısı için gerekli anahtarlar oluşturulurken *ServerHello* mesajındaki rastgele değer ile birlikte kullanılır.
- **SessionID:** İstemcinin oturum için kullanmak istediği ID (kimlik) değeri bulunur. Taraflar arasında eğer ilk kez el sıkışma yapılıyorsa bu alan boş olur.
- **CipherSuite:** Cipher Suite, şifre paketi anlamına gelmektedir. İstemci sunucuya desteklediği şifre paketlerini gönderir. Şifre paketlerinin içinde tüm protokol süresince gerekli olacak kriptografik algoritmaların isimleri bulunur. Sunucudan bu paketlerden birisini seçmesi beklenir. Eğer sunucu istemcinin gönderdiği paketlerin hiçbirisini desteklemiyorsa bağlantı sonlanır.
- **CompressionMethod:** Bu kısımda istemcinin desteklediği sıkıştırma yöntemlerinin listesi bulunur. İstemci bu bölümü boş olarak da gönderebilir. Fakat, sucucu liste içerisinde desteklediği bir sıkıştırma yöntemi bulamazsa veya sıkıştırma yapmama opsiyonunu kabul etmiyorsa bağlantı sonlanır.

- **Extension:** İstemci tarafından oturum için istenilen ekstra özellikler olması durumunda bu bilgiler Extension kısmında gönderilir.

İstemci *ClientHello* mesajını gönderdikten sonra sunucudan *ServerHello* mesajının gelmesini bekler.

4.3.1.3 Server Hello

Sunucu istemcinin *ClientHello* mesajında gönderdiği bilgilere göre bağlantı parametrelerini seçer ve istemciyi bilgilendirmek için *ServerHello* (Sunucu Merhaba) mesajını gönderir. *ServerHello* mesajı içerisinde aşağıdaki değerler bulunur [25]:

```
struct {
ProtocolVersion server_version;
Random random;
SessionID session_id;
CipherSuite cipher_suite;
CompressionMethod compression_method;
select (extensions_present) {
case false:
struct {};
case true:
Extension extensions<0..2^16-1>;
};
} ServerHello;
```

- **ProtocolVersion:** Sunucunun, istemci tarafından gönderilen SSL/TLS versiyonlarından hangisini seçtiği bilgisi bulunur.
- **Random:** Protokolde daha sonraki aşamalarda kullanılacak olan rastgele bir değer bulunur. SSL/TLS bağlantısı için gerekli anahtarlar oluşturulurken *ClientHello* mesajındaki random değer ile birlikte kullanılır.
- **SessionID:** Bu alandaki değer oturumun kimliğidir. Eğer buradaki değer sıfırdan farklı ise, sunucu bu değerin kayıtlı değerlerden biri olup olmadığına bakar (sunucular oturum yenileme yapılabilmesi için oturum kimliği değerini bir süre saklar). Eğer gönderilen değer kayıtlı ise sunucu bu değeri geri döner ve oturum yenileme yapılır (bkz. Bölüm 4.4). Kayıtlı olmayan değerlerde yeni bir el sıkışma olacağı anlamı çıkar. Yeni el sıkışma durumundaki SessionID değerinin sunucu tarafından kaydedilip kaydedilmeyeceği de sunucunun cevabı ile anlaşılır. Eğer sunucu gelen değerin aynısını dönerse oturum kimliğinin kaydedileceği anlamına gelir. Eğer boş bir değer dönerse kaydedilmeyeceği anlamına gelir.

- **CipherSuite:** Sunucu, istemci tarafından gönderilen şifre paketlerinden birisini seçer ve istemciye gönderir.
- **CompressionMethods:** Sunucu, istemci tarafından gönderilen şifreleme yöntemlerinden birisini seçer ve istemciye gönderir.
- **Extension:** İstemci tarafından istenilen olası eklentilerden sunucunun desteklediklerinin listesi bulunur.

4.3.2 Server Certificate

Sunucu istemciye, istemcinin sunucuyu doğrulayabilmesi için, sertifikasını gönderir (Bu adım, sunucunun doğrulanması gerektiği durumlarda gerçekleşir). Bu mesaj, sunucunun sertifikasının zincirini istemciye iletmiş olur. Server Certificate mesajında aşağıdaki değerler bulunur [24]:

```
enum {
rsa_sign(1), dss_sign(2), rsa_fixed_dh(3), dss_fixed_dh(4),
(255)
} ClientCertificateType;

opaque DistinguishedName<1..2^16-1>;

struct {
ClientCertificateType certificate_types<1..2^8-1>;
DistinguishedName certificate_authorities<3..2^16-1>;
} CertificateRequest;
```

- **CertificateTypes:** Bu kısımda sunucunun sertifika türü bilgisi bulunur.
- **CertificateAuthorities:** Bu kısım sertifikayı sunucuya veren otorite bilgisini içerir.

4.3.3 Server Key Exchange

Merhaba (Hello) mesajlarında seçilen anahtar paylaşımına göre, ana giz (master secret) oluşturulabilmesi için, sunucu ilave bilgiler gönderir. Bu ek bilgiler ön ana giz (premaster secret) değerinin hesaplanması için gerekli bilgilerdir. Her şifre paketi (cipher suite) için gönderilmesi gereken bir paket değildir.

4.3.4 Server Hello Done

ServerHelloDone mesajı, sunucunun ClientHello sonrasında gereken mesajları gönderdiğini ve anahtar anlaşma yapmaya hazır olduğunu belirten bir sinyaldir [25]:

```
struct { } ServerHelloDone;
```

Bu mesaja cevap olarak istemcinin yapması gereken şey sunucunun sertifikasını doğrulamak ve sunucunun gönderdiği mesajların uygunluğunu kontrol etmektir. Sıra dışı bir durum olmaması halinde taraflar hello mesajlarında karar kıldıkları anahtar anlaşma yöntemi ile el sıkışmaya başlayabilirler.

4.3.5 Client Certificate

Bu mesaj, istemcinin *ServerHelloDone* mesajını aldıktan sonra göndereceği mesajdır. Bu mesaj yalnızca sunucu tarafından sertifika talep edildiyse gönderilir. Günümüzde Internet üzerinde sitelerin büyük çoğunluğu istemciden sertifika istememektedir. Ancak e-devlet gibi resmi işler için erişim sağlanırken sertifika istenilebilir. *ClientCertificate* mesaj yapısı *ServerCertificate* mesaj yapısı ile aynıdır [24].

4.3.6 Client Key Exchange

İstemci bu mesajı göndererek ön ana giz değerinin oluşturulmasını sağlar. *ClientHello* ve *ServerHello* mesajları ile belirlenen şifre paketine göre hangi el sıkışma protokolü kullanılacaksa (RSA veya DH (Diffie−Hellman)) ona göre mesaj gönderilir. Eğer DH seçilmişse ve istemci *ClientCertificate* mesajını göndermişse, bu mesajda DH değeri gönderilmez. *ClientKeyExchange* mesajının yapısı, seçilmiş olan metoda göre değişebilir [25]:

```
struct {
        select (KeyExchangeAlgorithm) {
        case rsa:
                EncryptedPreMasterSecret;
        case dhe_dss:
        case dhe_rsa:
        case dh_dss:
        case dh_rsa:
        case dh_anon:
                ClientDiffieHellmanPublic;
        } exchange_keys;
} ClientKeyExchange;
```

4.3.6.1 RSA İle Şifreli Ön Ana Giz

Eğer anahtar anlaşma ve kimllik doğrulama için RSA seçilmişse, istemci 48 baytlık ön ana giz değerini oluşturur, bu değeri sunucunun açık anahtarı ile şifreler ve şifreli ön ana giz değerini sunucuya gönderir.

4.3.6.2 Diffie-Hellman ile Açık Değer Gönderme

Diffie−Hellman ile anahtar anlaşması yapılacaksa tarafların birbirlerinin açık değerlerini bilmeleri gerekir. Sunucu açık anahtarını istemciye $Certificate$ mesajında gönderdiği için tekrar göndermez. Eğer sunucu istemciden sertifika istememişse, anahtar anlaşması yapabilmek için istemciye ait açık anahtara ihtiyaç duyar (Bu açık anahtar, Bölüm 3'de anlatılan el sıkışmala prookollerinde Ayşe kullanıcısının gönderdiği "Cert(A)" değeridir).

4.3.7 Change Cipher Spec

ChangeCipherSpec mesajı kayıt protokolüne ait olmayıp, başlı başına bir protokoldür. Bu mesaj yalnızca bir bayt değer (1 değeri) içerir ve karşı tarafa sinyal vermeye yarar. Bu sinyal, bundan sonraki mesajların önceki mesajlarla uzlaşılmış olan simetrik anahtar ile şifrelenip gönderileceğini bildiren sinyaldir. Mesaj yapısı aşağıdaki gibidir:

```
struct {
        enum { change_cipher_spec(1), (255) } type;
} ChangeCipherSpec;
```

4.3.8 Finished

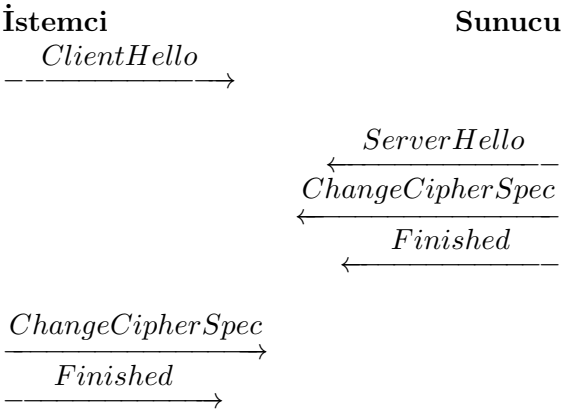
Finished mesajları, *ChangeCipherSpec* mesajlarından hemen sonra gönderilir ve hem anahtar anlaşmanın hem de kimlik doğrulamanın başarılı olduğunu ifade eder. Bu mesajlar, ortak anahtarla şifrelenmiş ilk mesajlardır. *Finished* mesajından sonra bir onaylama gerekmeksizin taraflar birbirlerine mesaj göndermeye başlayabilir. Mesaj yapısı aşağıdaki gibidir:

```
struct {
        opaque verify_data[verify_data_length];
} Finished;

verify_data
        PRF(master_secret, finished_label, Hash(handshake_messages))
        [0..verify_data_length-1];
```

4.4 Oturum Yenileme

El sıkışma senaryosu birçok mesajın iletildiği, zahmetli bir süreçtir. Bununla birlikte kriptografik fonksiyonlar ve özellikle sunucu ve istemcinin sertifika onaylama işlemleri büyük işlem gücü gerektirmektedir. Bu işlemlerin tekrar tekrar yapılmasına gerek duyulmayan durumlar için, kısaltılmış el sıkışma ortaya çıkmıştır. Oturum yenileme (Session Resumption) işleminin gerçekleştirilmesi, sunucunun bağlantı bitiminde oturum bilgilerini bir süre daha saklaması ile mümkün olmaktadır. Oturumu yenilemek isteyen sunucu $ClientHello$ mesajına cevap verirken, oturum ilk kurulduğunda oturuma ait atadığı ID değerini $ServerHello$ mesajında istemciye gönderir. Bu değer sayesinde taraflar, bir önceki el sıkışma senaryosunda kullanılan anahtar paylaşımı değerlerini kullanarak hızlı bir biçimde oturum elde etmiş olurlar. Oturumu yenilemek isteyen taraf istemci ise, $ClientHello$ mesajında oturum ID değerini sunucuya gönderir. Sunucu $ServerHello$ mesajında aynı ID değerini dönerse, oturum yenileme yapmayı onaylamıştır. Sunucu ve istemci önceden anlaşılmış ana giz (master secret) değerini kullanarak yeni anahtar değerini oluşturur. Sonuç olarak ağ yükü azaltılarak el sıkışma gerçekleştirilir. Oturum yenilemeye ait figür aşağıdaki gibidir:



Şekil 4.2: Oturum Yenileme

4.5 Anahtar Paylaşımı

Bir önceki bölümde (bkz. Bölüm 2) matematiksel olarak bahsedilmiş olan anahtar değişim algoritmaları protokol seviyesinde incelenecektir. TLS protokolü, farklı istemcilerin ihtiyaçlarına yanıt vermek amacıyla çok sayıda anahtar paylaşımı- değişimi algoritmasını desteklemektedir. Anahtar paylaşımı yapılmadan önce şifre takımı belirlenir ve sonrasında sunucu ile istemci hangi algoritmaları kullanacağını öğrenmiş olur. Bağlantıların tamamına yakınında kullanılmakta olan 4 temel anahtar protokol mevcuttur. Bunlar aşağıdaki gibidir:

- **RSA:** Internet haberleşmesinde en yaygın kullanılan protokoldür. Basit tasarımı ve kodlaması kolay yapısından dolayı yazılımcılar tarafından ilgi görmektedir. Buna karşın, pasif saldırganın saldırılarına karşı dayanıksızdır. İleriye yönelik gizlilik özelliği barındırmadığı için, yıllar içerisinde popülerliği düşmeye devam daha yeni ve ileriye yönelik gizlilik sağlayan algoritmalara bırakmaktadır.
- **DHE_RSA:** Ephemeral (geçici) ifadesi içeren Diffie-Hellman anahtar değişim protokolü RSA dan farklı olarak ileriye yönelik gizlilik sağlamaktadır. Kimlik doğrulama özelliği olmaması ve RSA ile kıyaslandığında yavaş olması dezavantajlarıdır.
- **ECDHE_RSA:** DHE protokolünün eliptik eğrilerle gerçeklenmesi ve kimlik doğrulama kısmında RSA kullanılması anlamına gelmektedir.
- **ECDHE_ECDSA:** Bir önceki madde ile aynı yapıya sahiptir. Doğrulama kısmında RSA değil ECDSA kullanılır.

4.6 Uyarı Protokolü

Uyarı protokolü,tarafların birbirlerine olası sorunları bildirebilmeleri amacıyla oluşturulmuştur. İçeriğinde uyarının seviyesi ve tanımı bilgileri bulunur. Yapısı aşağıdaki gibidir:

struct {
AlertLevel level;
AlertDescription description;
} Alert;

4.7 Haberleşme Güvenliği Protokolü - SSL/TLS

SSL/TLS protokolleri, zamanla gelişme uğrayan ve temel olarak kimlik doğrulama, asimetrik şifreleme ile ortak giz oluşturulması işlevlerini sağlama becerilerine sahiptir. Sağladıkları özellikler bunlarla sınırlı değildir. Teknolojinin adaptasyonu ve kullanım senaryoları arttıkça yeni hedefler ortaya çıkmıştır. Genel hatlarıyla aşağıdaki maddeler ön plandadır:

- **Kriptografik Problemlere Dayanma :** Güvenli protokolün temelinde güvenli kriptografik yapıtaşlarının kullanılması yatar. Eğer bu sağlanamazsa protokolün güvenli olması durumunda bile ciddi açıklıklar söz konusu olur.

- **Birlikte Çalışılabilirlik:** Farklı ortamlarda birbirine uzak tasarımlara sahip pek çok ağ cihazının haberleşebilmesi için ortak bir paydaya ihtiyaç vardır. İyi bir protokolün bu ihtiyaca karşılık verebilmesi beklenir.
- **Bağımsız Tasarım:** Protokoller tasarlanırken mevcut algoritmalara bel bağlanılmamalıdır. Zaman geçtikçe yeni algoritmalar ve yöntemlerle uyumlu bir yapı dizayn edilmelidir.
- **Verimlilik:** Tasarlanan sistemin kullanıma elverişli ve kolay adapte edilebilir olması gerekir. Çünkü kullanılabilir ve verimli olmayan bir güvenlik tercih edilmeyeceğinden ötürü güvenli kabul edilemez.

4.7.1 SSL Versiyonları- v1,v2,v3

SSLv1 protokolü 1994 yılında basına kapalı bir şekilde Netscape tarafından geliştirildi. birçok güvenlik açıklığına sahip olduğu farkedildi ve bu açıklıklar protokolü büyük oranda değiştirdi. Bu yüzden 1 yıl sonra SSLv2 ortaya çıktı. Ancak bu versiyon da kimlik doğrulama ve veri bütünlüğü gibi kritik güvenlik açıklıklarına sahipti ve bu yüzden bir yıl sonrasında SSLv3 versiyonu geliştirildi.

4.7.2 TLSv1.0

TLSv1.0 versiyonu, SSLv3 ün standartlaştırılıp daha profesyonel bir çatıda toplanması amacıyla çıkarılmıştır. İsminde bulunan soket kelimesi kaldırılmıştır. İki protokol arasındaki farklar kısaca aşağıdaki gibidir:

- PRF(Rastgele Sayı Üreteci) geliştirilerek rastgele sayı kullanılacak adımlarda güvenlik artışı sağlandı.
- SSLv3 da kullanılan HMAC versiyonu yerine resmi bir versiyon tercih edildi.
- POODLE saldırısına karşı dolgulama yapısı güçlendirildi.
- SSL versiyonlarında o tarihe kadar güvensiz olduğu ortaya çıkan bazı şifre takımları kullanımdan kaldırıldı.

4.7.3 TLSv1.1

2006 yılıda ortaya çıkan bu versiyon bir önceki version olan TLSv1.0 ile aşağıdaki farklılıklara sahiptir.

- BEAST saldırısına karşı önlem olarak AES CBC kipinde her kayıt için farklı bir başlangıç vektörü kullanılması sağlandı.
- Uyarı mesajlarında geliştirmeler yapıldı. Böylece dolgulama özelliğinin saldırılar için kullanılmasının önüne geçildi.

4.7.4 TLSv1.2

TLSv1.2, 2008'de yayınlandı. TLSv1.1 ile arasında kısa bir süre olmasına rağmen teknik anlamda önemli farklara sahiptir:

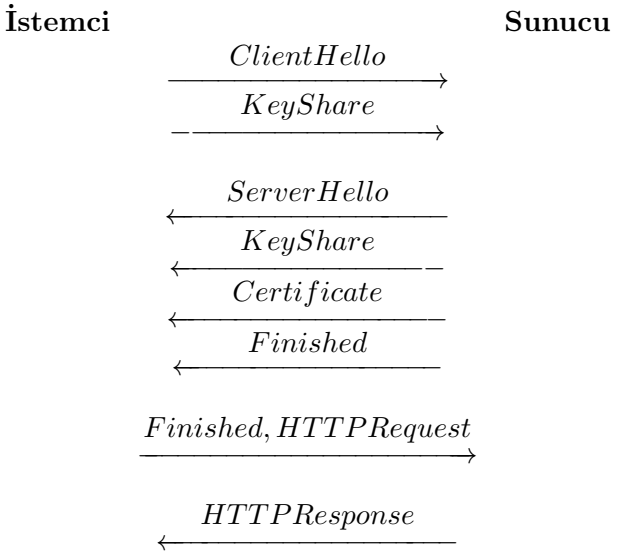
- Şifrelemede kimlik denetimi özelliği sağlayan ve authenticated encryption adı verilen özellik eklendi.
- Şifre takımları güncellendi ve güvensiz olanlar çıkarıldı.
- Özet algoritmalarında SHA256 tercih edilmeye başlandı. MD5 ve SHA1 algoritmaları çıkarıldı
- TLS bağlantılarında eklentileri özelliği geliştirildi.
- DES, RC4 gibi güvensiz protokoller yerine AES içeren şifre paketleri eklendi.
- $NULL$ değeri içeren şifre takımlarının yalnızca başlangıç durumunda TLS bağlantısının kontrolü için kullanılabileceğini ve güvensiz olduğu belirtildi.

4.7.5 TLSv1.3

TLSv1.3, 2018 yılı Temmuz ayı itibariyle son draft olarak mevcuttur. Fakat halen gerçeklenmesi ve Internet'te kullanımı başlamamıştır [25]. Bu versiyon tasarım itibariyle geçmişteki protokollere göre çok daha hafif ve hızlıdır. Ancak TLSv1.2 nin güvenli ve yüksek kullanıma sahip olması bu versionun adaptasyonu için gerekli motivasyona engel olmaktadır.

Bir önceki versiyonu ile aralarındaki birtakım farklar aşağıdadır:

- RSA kullanılmamaya başlandı. Tercih edilen tüm anahtar anlaşma protokolleri ileriye yönelik gizlilik barındırır hale geldi.
- $ChangeCipherSpec$ mesajı, $ClientHello$ ile birlikte gönderilip RTT (Round Trip Time) süresinde azaltma sağlandı.



Şekil 4.3: TLSv1.3 El Sıkışma Senaryosu

- Eski versiyonlarda saldırılara maruz kalan pek çok yöntem (AES CBC kipi, MD5, SHA1, DES vb.) kaldırıldı.
- TCP bağlantısı süresince sunucudan gelecek uyarıların çeşitliliği artırıldı. Bu sayede saldırı yüzeyi artırılıp tahmin saldırılarına karşı güçlendirme yapıldı.
- $NULL$ değeri içeren şifre takımları tamamen kaldırıldı.

TLSv1.3 protokolünde bulunan, $ChangeCipherSpec$ mesajı, $ClientHello$ ile birlikte gönderilmesi özelliği aşağıdaki gibidir:

4.8 Ağ Güvenliği Zaman Süreci

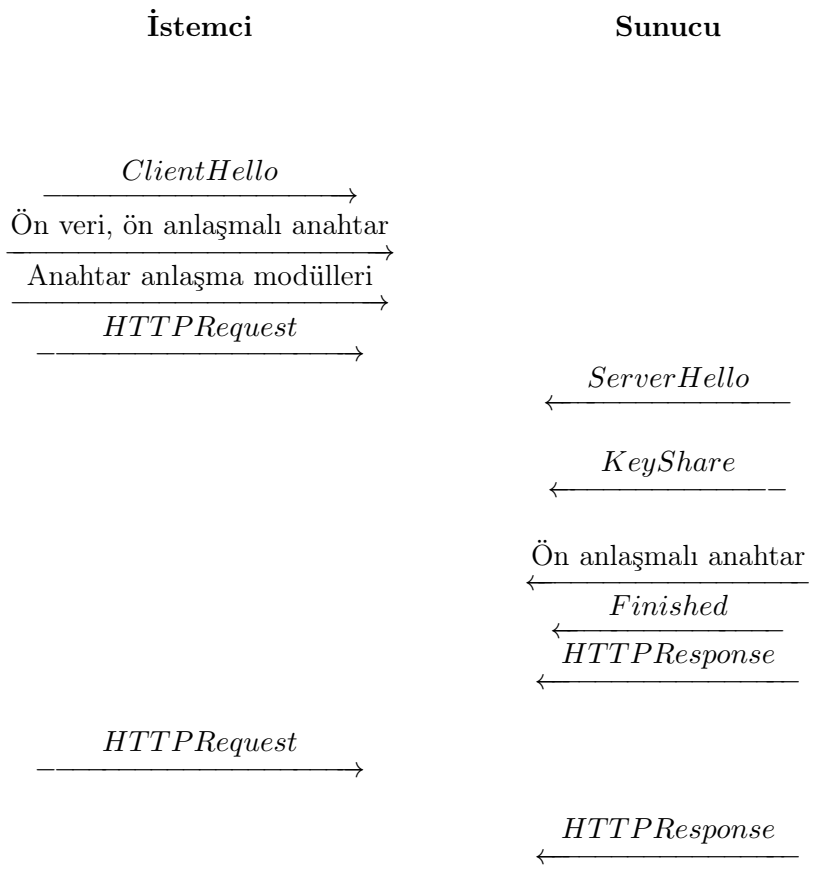
SSL/TLS protokolleri ve açık anahtar altyapısı, 20 yılı aşkın süredir Internet güvenliği konusunda rol almaktadır. Aşağıdaki zaman çizelgesinde, web güvenliğinin serüvenine dair önemli maddeler yer almaktadır:

1994	SSLv1 geliştirildi fakat ciddi güvenlik açıklıkları bulundurduğu için yayınlanmadı.
1994-Kasım	Netscape tarafından SSLv2 protokolü geliştirildi. 1995 yılında bu protokol Netscape Navigator üzerinde kullanılmaya başlandı
1995-Kasım	SSLv2 de ortaya çıkan ciddi açıklıklar sonucunda SSLv3 geliştirildi. Bu protokol zamanın şartlarına göre güvenliydi ve Internet'in altın çağı başlamış oldu.

1999-Ocak	IETF (Internet Mühendisliği Görev Gücü) tarafından 3 yıl süren bir çalışma ile SSL'in standartlaştırılması gerçekleştirildi. Ortaya çıkan protokole TLSv1.0 adı verildi. Bu protokol SSLv3 ile neredeyse aynı idi.
2001-Ocak	İlk sertifika sahteciliği yapıldı. Microsoft olduğunu iddia eden birisi VeriSign firmasından imzalı Microsoft sertifikasına sahip oldu
2006-Nisan	TLSv1.1 geliştirildi. Bu versiyonun sebep olduğu BEAST saldırısı 5 yıl boyunca farkedilemedi.
2007-Haziran	Bazı sertifika otoriteleri tarafından, kimlik doğrulama sürecinin iş yükünü azaltmak amacıyla Extended Validation Certificates geliştirildi.
2008-Mayıs	Debian sistemlerde üretilen rastgele sayı üreteçlerinin tahmin edilebilir olduğu farkedildi. Bu sistemler kullanılarak üretilen tüm anahtarlar güvensiz hale geldi.
2008-Temmuz	Mike Zusman tarafından mevcut bir alan adı için sertifika elde edilebilindiği farkedildi. Bu durum, aynı alan adına ait sertifikaların erişimi için standartların getirilmesine neden oldu.
2008-Ağustos	TLSv1.2 geliştirildi. Bu sürümün en büyük artısı kimlik doğrulamalı şifreleme (authenticated encryption) sağlaması oldu
2008-Aralık	Alec Sotirov ve Marc Stevens tarafından MD5 ın güvensiz olduğu ortaya çıkarıldı.
2009-Ağustos	Bir TCP bağlantısının yeniden anlaşma (renegotiation) ile iki farklı TLS bağlantısı oluşturulabildiği ortaya çıktı. Sunucu bunu farkedemediği için kurbana giden veriler üçüncü şahıslara da ulaşabilir oldu.
2009-Ağustos	Moxie Marlinspike tarafından "sslstrip" ortaya çıkarıldı. İki uç arasında taraflardan birisinin güvenli olduğunu sandığı bir saldırı gerçekleştirildi.
2010-Mayıs	Chrome tarafından HTTP Katı Transfer Güvenliği (HTST) listesi geliştirildi. Bu listedeki sitelerin ilk bağlantıları dahi güvenli hale getirildi.
2010-Eylül	Google HTTP den daha güvenli SPDY protokolünü geliştirdi. Bu protokolün en önemli artısı trafiğin her zaman şifrelenmesi oldu.
2010-Eylül	Google, TLS anahtar anlaşmasını hızlandıran bir yöntem olan False Start yöntemini kullanmaya başladı. Bu yöntemle, istemci henüz kimlik doğrulama tamamlanmadan uygulama verilerini göndermeye başlamış oldu.
2011-Mart	IETF (Internet Mühendisliği Görev Gücü) SSLv2 yi yasaklayan bir döküman yayınladı.
2011-Haziran	Şifreli blok zincirleme (cipher block chaining) yönteminde bulunan bir açıklıktan ötürü BEAST saldırısı ortaya çıktı.

2011-Kasım	Google tarafından ileriye yönelik gizlilik geliştirildi. Bu sayede uzun vadede gizli anahtarların sızması, eski bağlantıların güvenliğini tehdit etmemes, sağlanmış oldu.
2012-Eylül	Duong ve Rizzo tarafından CRIME adı verilen saldırı ortaya çıkarıldı. TLS'in sıkıştırma fonksiyonu üzerine gerçekleştirilen bu saldırı, tarayıcılar tarafından özelliğin hemen kaldırılması ile engellenmeye çalışıldı.
2012-Aralık	TürkTrust tarafından 2 adet CA sertifikası yanlışlıkla son kullanıcıya verildi. Bu hata 15 ay boyunca farkedilmedi.
2013-Şubat	AlFardan ve Paterson tarafından Lucky 13 saldırısı ortaya çıkarıldı. Saldırı, TLS'in CBC modu ile şifreleme yöntemine yapılan dolgu tahmini (padding oracle) ile yapıldı.
2013-Ağustos	HTTP paketinin şifrelenmeden önce sıkıştırılmasından ötürü BREACH saldırısı ortaya çıktı.
2013-Ağustos	TLSv1.3'ün geliştirilmesine başlandı. TLSv1.2 güvenli olarak görülmesine rağmen, ilerleyen yıllarda ihtiyaçları karşılamasının mümkün olmadığı öngörüldü.
2014-Ocak	RSA sertifikaları için 1024-bit anahtar kullanılmamaya başlandı. Asgari anahtar uzunluğu 2048-bit olarak belirlendi.
2014-Mart	Üçlü el sıkışması saldırısı (Triple Handshake Attack) ortaya çıktı. TLS'in tekrar anlaşma (renegotiation) özelliğinden ortaya çıkan bu saldırı, pratikte büyük bir etki yaratmadı.
2014-Nisan	TLS'in en yaygın implementasyonlarından olan OpenSSL'in heartbeat açıklığından ötürü Heartbleed saldırısı ortaya çıktı. Korunmasız sistemlerin hafızasındaki bilgilerin sızmasına yol açan bu saldırı, o güne dek çıkan en etkili saldırı oldu.
2014-Kasım	SSLv3'ün dolgu tahmini saldırılarına karşı dayanıklı olmadığını ortaya çıkaran POODLE saldırısı ortaya çıktı.
2014-Aralık	POODLE saldırısından 1 ay sonra TLSv1.0'ın da dolgu tahmini saldırılarına karşı dayanıklı olmadığı farkedildi. Bu saldırıya POODLE TLS adı verildi.
2015-Mart	İthal kriptografi kullanan sunucuların maruz kaldığı FREAK saldırısı ortaya çıktı.
2015-Mayıs	Diffie-Hellman anahtar değişiminde ithal şifre paketleri kullanılması ile ortaya Logjam saldırısı çıktı.
2016-Ocak	Sertifika otoriteleri tarafından SHA1 kabul edilmemeye başlandı.
2016-Ocak	Kripto Forum Araştırma Grubu (Crypto Forum Research Group) tarafından iki adet eliptik eğri standartlaştırıldı (Curve25519 ve Curve448).

2016-Mart	SSLv2'ye yönelik yeni bir saldırı olan DROWN saldırı ortaya çıktı. Bu saldırıyı etkili yapan ise, SSLv2 desteklemeyen sunucuların da ortak anahtarlardan ötürü savunmasız olabilmesidir.
2016-Temmuz	Google, artık RC4 ve SSLv3'ü desteklemediğini açıkladı.
2016-Ekim	Mozilla tarafından yapılan açıklamaya göre, 2016 Ekim ayı itibariyle ağ trafiğinin yarısından fazlası HTTPS üzerinden yapılmaya başlandı.
2017-Ocak	SHA-1 içeren sertifikalar tarayıcılar tarafından kabul edilmemeye başlandı.
2017-Nisan	Aynı alan adı görünümlü ve sertifikalı oltalama saldırısı ortaya çıktı. Bu saldırı "farkedilmesi neredeyse imkansız" olarak tanıtıldı.



Şekil 4.4: TLSv1.3 0-RTT

Bölüm 5

SSL/TLS Protokolündeki Elektronik Sertifika Altyapısına Yapılan Saldırılar

Açık anahtar altyapısı ile ilgili saldırılar, taşıma katmanı güvenliğinin pek çok aşamasını ilgilendirmektedir. Bu yüzden açık anahtar altyapısı ve X.509 sertifikası [50] ile ilgili güvenlik sorunları bu bölümde incelenmiştir.

5.1 Sertifika Otoritelerinin Güvenliği

Teknolojinin günümüzde gelmiş olduğu aşamada, Internet bankacılığı, e-posta, elektronik alışveriş gibi ağ uygulamaları insanların hayatlarında büyük yer almakta ve vazgeçilmez hale gelmektedir. Bu uygulamalar kimlik doğrulama işlemleri için Sertifika Otoritesi tarafından işlenmiş olan X.509 sertifikalarını kullanırlar. Açık anahtar altyapısı mimarisine göre bu sertifika otoriteleri güvenilir olarak kabul edilmekte ve kullanıcıların işletim sistemlerinde veya tarayıcılarında kayıtlı olarak bulunmaktadır. Sertifika otoritelerine karşı duyulan bu koşulsuz güven, onların hata yapmayacakları ve güvenlik açıklıkları bulundurmayacakları varsayımına dayanır. Ancak bu varsayım, büyük bir tehlike barındırmaktadır. Çünkü, sertifika otoriteleri tamamiyle güvenilir olmayabilir veya güvenlik gereksinimlerini tam olarak karşılayamayabilir. Örneğin, son 10 yılda pek çok ağ adresi için, ağ sahibinin haber olmaksızın sahte sertifikaların işlenmesi olayları yaşanmıştır (bkz. Bölüm 5.5,5.2.1)[52]. Bu sertifikaların işlenmesi, ortadaki adam saldırılarını mümkün kılmakta ve büyük çapta siber saldırılara yol açabilmektedir (örn. Stuxnet). Bu yüzden Sertifika Otoritelerinin yetkilerinin dağıtılması ve mutlak güvenin tek bir noktada olmasının engellenmesi için çalışmalar yapılmaktadır (bkz. Bölüm 5.1.2).

Sertifika Otoriteleri güvenilir olmayan yapılarla pek çok şekilde iletişim halinde olabilir. Eğer bir Sertifika Otoritesi herhangi bir şekilde ele geçirilmiş ise (örn. sisteme sızma, bilgi çalınması, zararlı yazılım yüklenmiş olması vs.) bu Sertifika Otoritesi tarafından işlenen tüm sertifikalar da ele geçirilmiş olur. Sertifikaların verilerin gizliliği, kimlik doğrulama ve yetkileme gibi işlemlerde kullanılmasından ötürü, Sertifika Otoritesi'nin ele geçirilmiş olması, tüm kullanıcılarının verilerinin mahremiyetinin ve iletişim güvenliğinin ihlal edilmesi anlamına gelir. Bu riske karşı önlem olarak pek çok organizasyon çevrimdışı olarak oluşturdukları kendi Sertifika Otorite'sini yönetmeye yönelmiştir. Böylece Sertifika Otoritesi organizasyonun ağı ile iletişim halinde olmamakta ve ele geçirilme tehlikesinin önüne geçilmektedir.

5.1.1 Sertifikaların Zaman İçerisindeki Sorunları

Açık anahtar altyapısı ve sertifika kavramı, yıllar içerisinde pek çok güvenlik problemi ile karşı karşıya kalmıştır (bkz. Bölüm 6). Sertifikaların yaşadıkları sorunlar Internet'in yaşam süreci ile şekillenmiştir. Güvenli Internet kavramı, kullanıcıların Internet üzerinden gizlilik ve güvenlik gerektiren işlemleri yapmaya yönelmesi ile önem kazanan bir kavram olmuştur. Bu yüzden sertifikalar ve Sertifika Otoriteleri'nin rolleri ortaya çıkan eksiklikler ve saldırılar ile şekillenmiştir. Bu süreç sonucunda ortaya birçok Sertifika Otoritesi ortaya çıkmış ve rekabet yaşanması sebebiyle Sertifika Otoriteleri arasında da güvenlik problemleri yaşanmaya başlamıştır. Sertifika Otoritelerinin kendi yönetici sertifikalarına ait gizli anahtarının ifşa olması, CRL lerin yeterince güncel olamaması, sertifikasız sitelere girilirken tarayıcılarda çıkan uyarıları kullanıcıların umursamayıp erişim sağlaması gibi pek çok sorun yaşanmıştır.

5.1.2 Sertifika Otoriteleri'nin Güven Mekanizmasının Dağıtımı

Sertifika Otoriteleri'nin üzerindeki güveni dağıtarak güvenlik açıklarının önüne geçilmesi amaçlanmıştır. Bu amaçla anahtar işaretleme (key pining [37]), çoklu kaynak (crowd sourcing [28]), tarayıcılara iptal listesini gönderme [11] gibi önlemler alınmıştır. Ancak bu çözümler büyük çapta uygulanabilir olamamıştır.

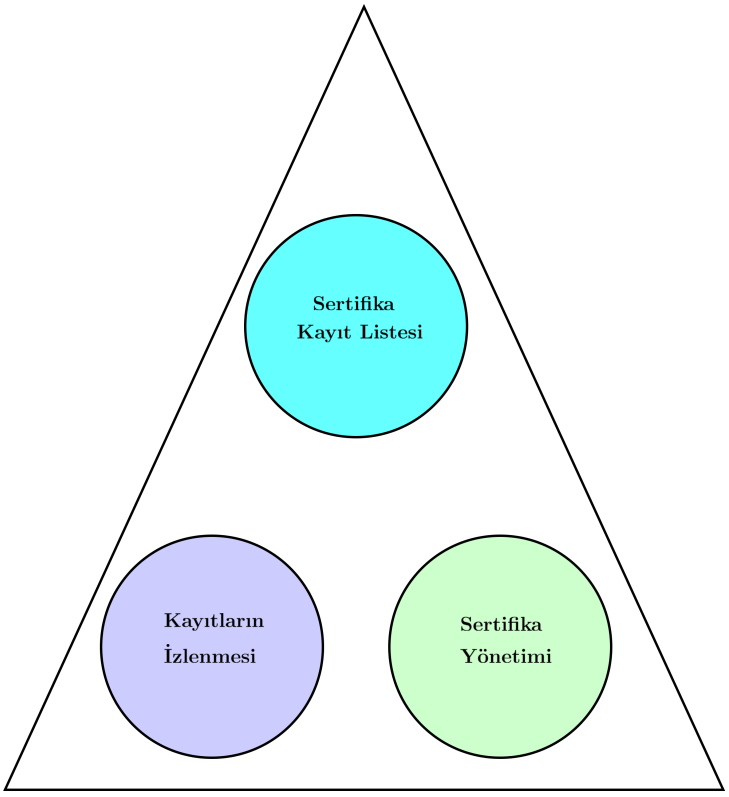
5.1.2.1 Yetkili Anahtar Yöntemi

Yetkili Anahtar (Sovereign Key Cryptography) Yöntemi'nde, istemci sunucu ile TLS bağlantısı kurarken, sertifika talep etmenin yanı sıra, web sitesinin yöneticisi tarafından oluşturulmuş ikincil bir anahtarı (sovereign key) talep eder [35]. Sunucu gönderdiği bu

anahtar değerinin doğrulanabilmesi için alan adına ait veri tabanı kayıt girdilerini de gönderir. İstemci ikincil anahtarı kontrol ederek erişmek istediği siteyi doğrular.

5.1.2.2 Sertifika Şeffaflığı

Sertifika Şeffaflığı (Certificate Transparency), Yetkili Anahtar Yöntemi'ndeki log kontrolüne ilave olarak zaman damgasının imzalanması ile geliştirilmesidir [63]. Bununla birlikte Merkle Hash Trees yöntemi ile logların doğrulanması yapılabilmektedir [77]. Sertifika Şeffaflığı iki farklı kriptografik yöntemle ispatlanabilmektedir. İlki, kayıt içerisinde ilgili sertifikanın bulunup bulunmadığı, ikincisi de, gönderilen kayıtların gerçek kayıtlar olup olmadığı kontrolüdür. Alan adı yöneticileri istemciye sertifikalarının zaman damgalarını gönderirken X.509 sertifika uzantısı, TLS uzantısı veya OCSP damgası olarak gönderebilir. Sertifika Şeffaflığı, bu yolla gelen sertifikalardan sahte olanların farkedilebilmesini sağlar.



Şekil 5.1: Sertifika Güven Üçgeni

5.1.2.3 Sorumlu Anahtar Altyapısı

Sorumlu Anahtar Altyapısı (Accountable Key Infrastructure), alan adının yalnızca sertifikasının değil, kayıt bilgisi, güncelleme bilgisi ve iptal bilgisi gibi değerlerin tutulduğu

dizin kayıtlarının da kontrol edilmesi yöntemidir [57]. Bu yöntemde sunucu, birden fazla Sertifika Otoritesi tarafından desteklenebilir ve sorgulanabilir durumdadır. Bununla birlikte birden fazla yetkinin olması kötü niyetli bir Sertifika Otoritesi'nin farkedilebilmesini mümkün kılar.

Teoride önemli bir güvenlik sağlamasına rağmen, hem sunuculara, hem de Sertifika Otoriteleri'ne büyük oranda işlem yükü katacak olmasından ötürü Sorumlu Anahtar Altyapısı, uygulanabilir bir yöntem değildir.

5.1.2.4 Saldırıya Dirençli Açık Anahtar Altyapısı

Saldırıya Dirençli Açık Anahtar Altyapısı, tüm Sertifika Otoriteleri'nin üzerine binen işlem yüküne hafiflik getirmesidnen Sorumlu Anahtar Altyapısı yönteminin geliştirilmiş hali kabul edilebilir [8]. Sunucu sertifika oluşturma işlemini tek bir Sertifika Otoritesi ile gerçekleştirebilir fakat, diğer tüm Sertifika Otoriteleri sertifika işlemlerini takip etme yetkisine sahiptir. Böylece hatalı veya kötü niyetli sertifika işlemleri diğer Sertifika Otoriteleri tarafından farkedilebilir hale gelir.

Bu yöntemin dezavatajı ise, Sertifika Otoritelerinin üzerine binen izleme işlemleri yüküdür. Tüm Sertifika Otoriteleri, birbirlerini takip edecek olduğu için hem ağ trafiğinde hem de işlenecek verilerde büyüme olacağı için pratik olarak gerçeklenmesi mümkün olmayan bir yöntemdir.

5.1.3 Bilinen Saldırılar ve Yanlış Kullanımlar

Açık anahtar altyapısının güven modelinde, Sertifika Otoriteleri tamamen güvenilir kabul edilmektedir. Ancak bu yaklaşım ciddi anlamda tehlikelidir. Sertifika Otoriteleri'nin güvenliğini sağlamak için sarfedilen çabalara rağmen Sertifika Otoriteleri'nin ele geçirilmesi ve güvensiz duruma düşmesine birçok örnek bulunmaktadır. Örneğin:

- 2011 yılı Mart ayında Comodo Sertifika Otoritesi firmasına sızıldı ve bir miktar sahte sertifika Comodo'nun yetkisi ile işlendi. Aynı yıl içerisinde Hollanda menşeili DigiNotar firması da benzer bir şekilde sahte sertifika işleme saldırısına maruz kaldı. 2012 Aralık ayında Google tarafından, Türk bir Sertifika Otoritesi olan TURKTRUST şirketinin sertifika işlerken yanlışlıkla 'sertifika işleme yetkisi olan' bir sertifika işlediği farkedildi.
- 2015 yılı Nisan ayında Çin merkezli CNNIC firmasına bağlı Mısır tabanlı bir ara Sertifika Otoritesi, yanlışlıkla Google alan adına ait sertifikalar işledi. Bu hata sonucunda CNNIC firmasının kök sertifikası Firefox ve Chrome tarafından bloklandı. Bu

olayın sonrasında, ara Sertifika Otoritesi'nin gizli anahtarların depolanması ve açık anahtar altyapısı konularında bilgi sahibi olmadığı ve doküman kontrolü yapmadığı ortaya çıktı. Aynı yılın Mayıs ayında Symantec firmasının da Google'a ait alan adı için sertifikalar işlediği ortaya çıktı. Symantec firması bu işlemin test amaçlı yapıldığını ve yanlış ellere düşmediğini iddia etmiş olsa da, konu ile ilgili bazı çalışanlarını işten çıkardı.

- 2016 yılı Mayıs ayında Blue Coat firması, Symantec Sertifika Otoritesi üzerinden kendisine Sertifika Otoritesi sertifikası edindi. Ancak Blue Coat firması, SSL/TLS trafiğini dinlemek için uygulamalar geliştiren bir firma olduğu için sorun yaşanmış oldu. Trafiği dinlemek için araya giren bir saldırgan, Symantec tarafından işlenen bir sertifikaya sahip olduğu için, taraflar tarafından güvenilir kabul edilir. Bu tehlikeyi engellemek için Symantec ve Blue Coat firmaları gizli anahtarın Symantec tarafında tutulacağını ve bu ara sertifikanın yalnızca test için kullanılacağını açıkladı. Ancak bu açıklama şüpheleri yatıştırmadı ve Blue Coast'ın araya girme programlarının Burma, İran ve Suriye gibi ülkelerde kullanıldığına dair tahminler ortaya atıldı.
- Sertifika Otoriteleri'nin Devlet tarafından müdahaleler ile de arka kapılara sahip olabileceğine dair olaylar yaşanmıştır. Örneğin, 2013 yılı Aralık ayında Google tarafından farkedilen ve Google'a ait alan adlarına verilen sertifikalar gibi. Bu sertifikaları veren Sertifika Otoritesi ANSSI ile bağlantılı olduğu ortaya çıkmıştır. Hollanda'lı Sertifika Otoritesi olan Diginotar firmasına sızılması ile işlenen sahte sertifikaların da İran içerisinde SSL/TLS trafiğini dinlemek için kullanıldığına dair haberler çıkmıştır.

5.2 Pratik Saldırılar

Bu bölümde, elektronik sertifika altyapısındaki açıklıklar nedeniyle mümkün hale gelmiş olan saldırılardan bahsedilecektir.

5.2.1 Microsoft'un Sertifika Hatası

2001 yılında, Finlandiya'lı bir bilgi işlem uzmanı [94], Comodo firmasına Microsoft yetkilisi gibi mail atarak Microsoft Windows Live sertifikasına sahip oldu. Bu sertifika sayesinde kişi, Microsoft adına imza atıp zararlı uygulama yükleyebilir yada güvenli bağlantılarda farkedilmeden araya girip büyük miktarlarda maddi kazanç sağlayabilirdi. Ancak iyi niyetli birisi olduğu için açıklığı Microsoft'a bildirdi ve sertifikası iptal edildi.

Bu olay ile birlikte sertifika yaşam döngüsünün ve her aşamada kontrolün ne denli önemli olduğu ortaya çıkmış oldu. Büyük firmaların dahi basit hatalar yapabileceği, güvenli iletişimde her adımın dikkatle atılması gerekliliği ortaya çıktı.

5.2.2 OCSP Saldırısı

Moxie Marlinspike tarafından 2009 yılında sertifika durumu kontrolleri ile ilgili bir saldırı gerçekleştirildi [69]. Çevrimiçi Sertifika Durumu Protokolü (OCSP) sayesinde bir istemci bir sertifikanın iptal olup olmadığını kontrol edebilmektedir (bkz. Bölüm 2.1.5.2). OCSP tarafından dönülen mesajın yapısı aşağıdaki gibidir [40]:

```
OCSPResponse ::= SEQUENCE {
responseStatus         OCSPResponseStatus,
responseBytes          OPTIONAL

OCSPResponseStatus ::= ENUMERATED {
successful            (0),  --Response has valid confirmations
malformedRequest      (1),  --Illegal confirmation request
internalError         (2),  --Internal error in issuer
tryLater              (3),  --Try again later
--(4) is not used
sigRequired           (5),  --Must sign the request
unauthorized          (6)   --Request unauthorized
}
```

OCSP cevabı alındığında *responseStatus* değerinin sayısal karşılığı sorgulama cevabını belirlemektedir. Örneğin "0" değeri sorgulamanın başarılı olduğunu, "6" değeri de isteğin yetkisi olmadığını belirtir. Göz ardı edilen açıklık ise, cevabın "3" yani "tekrar deneyin" olması durumunda cevabın imzalanmadan gönderilmesidir. Bundan ötürü "tekrar deneyin" mesajının gerçekten de yetkili bir makamdan gelip gelmediği kontrol edilemez. Araya giren bir saldırgan istemci ya da istemcilerin OSCP isteklerinin tümünü düşürüp, istemciye cevap olarak "tekrar deneyin" mesajı döner ve servise erişimi engellemiş olur. Bu açıklığı kapatabilmek için tüm cevap türlerinin imzalanması ve bu sayede doğrulanması gerekmektedir.

5.2.3 Let's Encrypt Sertifika Otoritesi ve Oltalama Saldırısı

2015 yılında "Linux Foundation" tarafından, "Lets Encrypt" isimli ücretsiz ve otomatize bir sertifika otoritesi kuruldu [2]. Bu kuruluşun amacı, isteyen her web sunucusuna ücretsiz olarak sertifika sağlamak ve böylece Internet güvenliğini artırmaktı. Ancak, 2017 yılı Mart ayında ortaya çıkan bir oltalama saldırısı, bu sertifika otoritesinin kötüye kullanılabileceğini ortaya çıkardı.

2016 yılı boyunca, "Lets Encrypt" tarafından 15 bin civarında sertifika işlendi [19]. Bu sertifikaların %96.7'sinin oltalama saldırısı yapan siteler olduğu ortaya çıktı. Bu siteler yüzünden kurbanlar güvenli iletişim kurdurklarını ve PayPal ile konuştuklarını sanarak saldırganlarla iletişim kurmuş oldu. Sayısı tam olarak bilinmeyen pek çok kullanıcının kredi kartı bilgileri çalınmış oldu.

Diğer sertifika otoritelerine göre hızlı ve otomatik bir şekilde çalışan Lets Encrypt'in gözünden kaçan bu kötüye kullanım, sertifika otoritelerinin hem güvenlik hem de verimlilik açısından sorunlarını çözemediklerini göstermektedir.

5.3 Sertifika Özetlerinin Çakışması

5.3.1 MD5 Özet Çakışması

2005 yılında Lenstra, Wang ve Weger tarafından ortaya çıkarılan açıklıkta, MD5 ile özeti alınan geçerli iki sertifikanın aynı özet değerini vermesi sorunu gündeme getirilmiştir [90]. Çakışan özet değerleri ile bir sunucu ya da istemci başka bir kişinin yerine geçebilmektedir.

Özet çakışmasının önüne geçebilmek için sertifika otoriteleri MD5 kullanan sertifikaları işlememe yolunu seçmişlerdir. Ancak bir kullanıcı MD5 desteğini kaldırmadığı müddetçe bu açıklığa halen sahiptir.

5.3.2 SHA1 Özet Çakışması

2017 yılı şubat ayında, Google'da çalışan araştırmacılar tarafından aynı SHA-1 özet değerine sahip iki farklı döküman oluşturmanın mümkün olduğu gösterildi [91]. Ortaya çıkmasının üzerinden 20 yıl geçen SHA-1 için ilk kez pratik olarak özet çakışması ortaya çıkmış oldu. Yaklaşık 2 yıl süren bir çalışma ile aynı özeti veren iki farklı PDF (yazılabilir döküman formatı) üretildi.

Bu açıklığa önlem olarak SHA-1 algoritmasının kullanılmaması, bunun yerine günümüzde halen güvenli olan SHA-256 ve SHA-3 algoritmalarının tercih edilmesi gerekmektedir.

5.4 Internet Explorer SSL Sertifika Açıklığı

Moxie Marlinspike tarafından 2008 yılında ortaya çıkarılan açıklıkta, Internet Explorer tarayıcısının kritik bir adımı atladığı farkedilmiştir [68]. Bu kritik adım; bir sertifikanın

alt sertifika imzalama hakkının olup olmadığının kontrol edilmesi adımıdır. Yani istemci bir sitenin sertifikasını kontrol ederken, geçerli bir Sertifika Otoritesi olup olmadığına bakar. Eğer Sertifika Otoritesi tarafından imzalanmadıysa, Sertifika Otoritesi tarafından sertifika imzalama yetkisi verilen bir ara sertifika ile imzalanan bir sertifika olması gerekmektedir. Bunun kontrolünü sağlamak için, tarayıcının önüne gelen sertifikayı ve onun üzerindeki sertifikaları kontrol edip, ara sertifikaların imzalama yetkisini kontrol etmesi gerekir. Internet Explorer bu kontrolü yapmadığı için, araya giren bir saldırgan geçersiz bir sertifika ile istediği sertifikayı imzalar ve kontrolden geçmediği için istemci tarafından güvenli sanılarak istediği site ile güvenli iletişim kurduğunu sanmasına yol açar.

Bu açıklık hızlı bir şekilde giderilmiş olup, tarayıcının gerekli kontrolleri yapması sağlanmıştır. Böyle bir saldırı, sertifika doğrulama yapılmasının önemini ortaya çıkarmıştır.

5.5 Sahte Sertifika Otoritesi Sertifikası (RapidSSL)

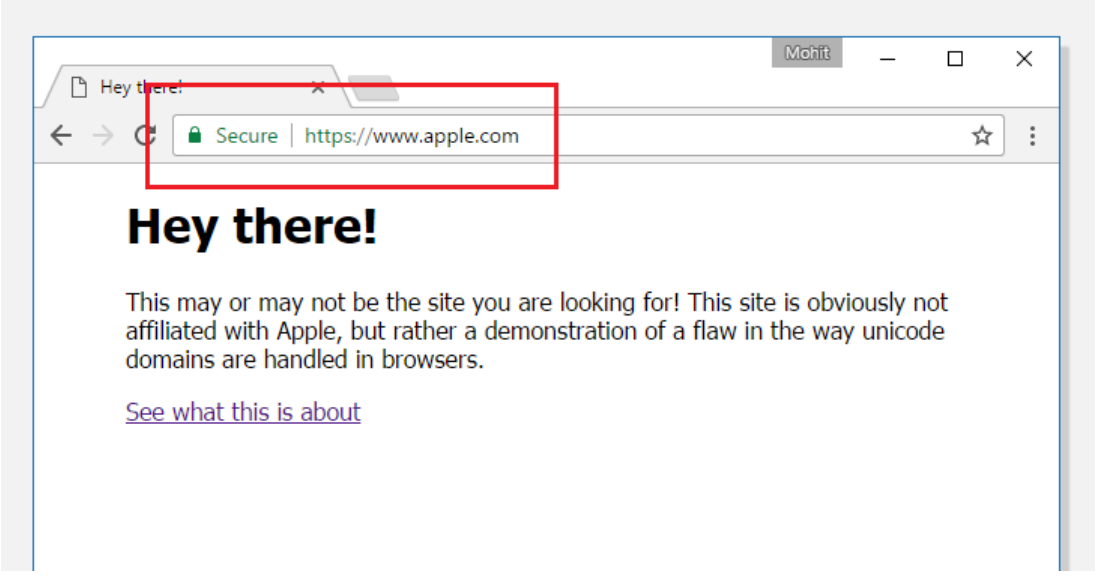
2008 yılında Sotirov ve Stevens tarafından ortaya çılarılan teorik bir saldırıya göre, dünyadaki tüm sertifikaları imzalayabilecek sahte bir sertifika ortaya çıkarmak mümkündür [90]. Saldırının temeli, MD5 özet algoritmasına yönelik saldırılardan kaynaklanmaktadır. MD5 algoritmasına yönelik saldırılar aşağıdaki listede özetlenmiştir [81]:

- **1991:** Ronald Rivest MD4 algoritmasının yerine kullanılması için MD5'i tasarlamıştır [82].
- **1991-1996:** MD5 algoritması geniş çapta kullanılmaya ve popüler olmaya başlamıştır. Bununla birlikte MD5'e yönelik açıklık sinyalleri ortaya çıkmış [21] ve daha güvenli özet algoritmaları tavsiye edilmeye başlanmıştır.
- **2004:** Wang tarafından çakışma saldırısı geliştirilmiştir [98]. Bu saldırı ile MD5 artık güvensiz kabul edilmeye başlansa da, saldırının pratik uygulaması henüz gerçekleşmemiştir.
- **2005:** Lenstra, Wang ve Weger tarafından pratik bir saldırı gerçekleştirilmiştir [65]. Bu saldırıya göre iki farklı sertifika çakışmadan ötürü aynı MD5 özetine sahiptir ve bu yüzden imzaları aynıdır.
- **2006:** Stevens, Lenstra ve Weger tarafından yeni bir teknik ortaya sunulmuştur [90]. Bu pratik yönteme göre iki farklı sertifikanın aynı MD5 özet değerine sahip olabilmesi mümkün hale gelmiştir.

- **2008:** MD5 algoritmasının zayıf kabul edilmesine ve 2006 yılında pratik saldırı yapılmış olmasına rağmen bazı sertifika otoriteleri yeni sertifikalarda MD5 kullanmaktan vazgeçmemiştir. Sotirov ve Stevens tarafından geliştirilen bir saldırı ile, MD5 çakışması ile sahte bir sertifika ortaya çıkarılmış ve bu sertifika ile herhangi bir web sitesi için geçerli bir sertifika oluşturabilmek mümkün hale gelmiştir [89].
- **2012:** Flame adı verilen etkili bir zararlı yazılım Orta Doğu'da ağ cihazlarına bulaşmış halde bulundu [105]. Bu yazılım, MD5 özet çakışmasından faydalanarak Windows güncellemesi sırasında Microsoft'a ait Sertifika Otoritesi sertifikasının yerine geçerek bilgisayarlara istediği yüklemeleri yaptırabilmiştir. Bu yazılımın kim tarafından geliştirildiği ve ne kadar süredir aktif olduğu kesin olarak bilinmemektedir.

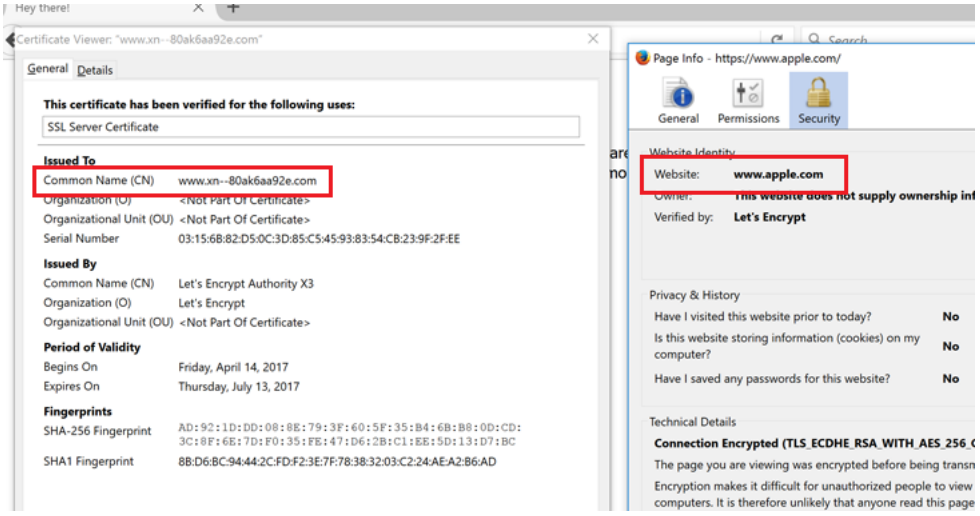
5.6 Unicode Alan Adları ile Oltalama Saldırısı

Nisan 2017'de Xudong Zheng isimli araştırmacı tarafından "farkedilmesi neredeyse imkansız" denilen bir oltalama saldırısı ortaya çıkarıldı [104]. Bu saldırı, Unicode karakterlerinin pek çoğunun ASCII karakterlerinden ayırt edilememesinden kaynaklanmaktaydı. Örneğin; kiril alfabesindeki "a" (U+0430) ve latin alfabesindeki "a" (U+0041) karakterleri Unicode olarak farklı olsa dahi, tarayıcının adres ekranında aynı görünmektedir. Saldırgan bir siteye ait alan adını farklı Unicode karakterleri kullanarak ve aynı görüntüye sahip halde göstererek kendisine yeni bir alan adı oluşturur. Sonra bu alan adına ait bir sertifika alır. Oltalama yapmak istediği kurban ise bu siteye girdiği zaman hem geçerli bir alan adı görür, hem de sertifikalı iletişim olduğu için saldırıyı fark edemez.



Şekil 5.2: Unicode Karakterli Sahte Alan Adı

Şekil 5.2'de görüldüğü gibi www.apple.com adresi görünümli bir web sitesi aslında farklı karaktelerle oluşturulmuş ve sahte bir alan adıdır. Ancak hem tarayıcı ekranında sıradışı bir karakter yazmaması, hem de saldırganın sahte site için sertifika almış olması şüphelenilmeyecek bir görüntü sunmaktadır. Bu web sitesinin sertifikasının ayrıntıları ise Şekil 5.3'deki gibidir. Sertifikaya ait alan adı ve organizasyon adı bilgileri Apple firmasına ait olmayan bilgiler içermektedir.



Şekil 5.3: Sahte Sertifika ile Oltalama

Neredeyse her insanın text olarak gördüğü geçerli bir linke ve yeşil sertifika ikonuna güvendiği göz önüne alındığında, bu saldırı ciddi anlamda tehlikeli hale gelmektedir. Bu ve benzeri oltalama saldırılarına maruz kalmamak için - özellikle de elektronik alışveriş ve bankacılık işlemlerinde - erişilmek istenen web adresi el ile girilmeli, linklere güvenilmemelidir.

Bölüm 6

SSL/TLS Haberleşme Protokolüne Yönelik Saldırılar

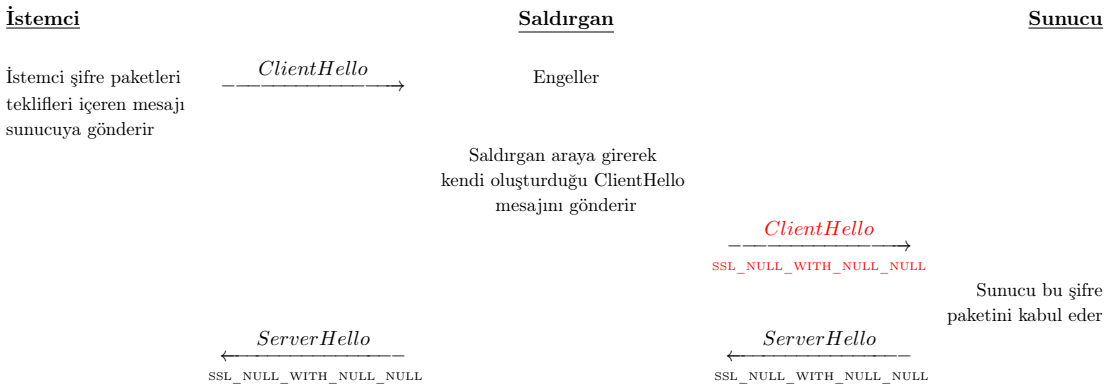
Bu bölümde TLS protokolüne yapılan saldırılar kronolojik olarak yer almaktadır. Bu saldırıların bazıları kriptografik açıklıklardan ve protokol hatalarından kaynaklanmaktadır. Bununla birlikte protokolün gerçeklenmesinde (implementasyonunda) oluşan hatalardan kaynaklı bazı saldırılara da yer verilmiştir. Bölüm sonunda (bkz. 6.15) saldırıların ve önlemlerinin yer aldığı bir tablo ile saldırılar özetlenmiştir.

6.1 Şifre Paketi Düşürme Saldırısı

Şifre paketi düşürme saldırısı 1996 yılında Wagner ve Scneier tarafından ortaya çıkarılan ve SSLv2.0 a yapılan bir saldırıdır [97]. Saldırgan; istemcinin ClientHello mesajında bulunan şifre paketlerini silip, NULL şifrelerle (NULL şifre,şifreleme yapmama anlamına gelir [39]) değiştirip sunucuya gönderir. Sunucunun iletişimi güvenli hale getirmek için yapabileceği birşey kalmaz. Ya bağlantıyı sonlandırır ya da NULL şifrelemeyi kabul eder.

SSLv3.0 ile NULL şifre paketleri kaldırılarak bu sorun giderilmeye çalışılmıştır. Ancak; SSL 3.0 ın SSLv2.0 ı desteklemesinden ötürü bu açıklık devam etmiştir. Sonrasında SSL 3.0 ın NULL şifreleri hiç bir zaman desteklememesi sağlanmış ve gönderilen mesajların hash değerleri de pakete eklenerek doğrulama yapılmıştır.

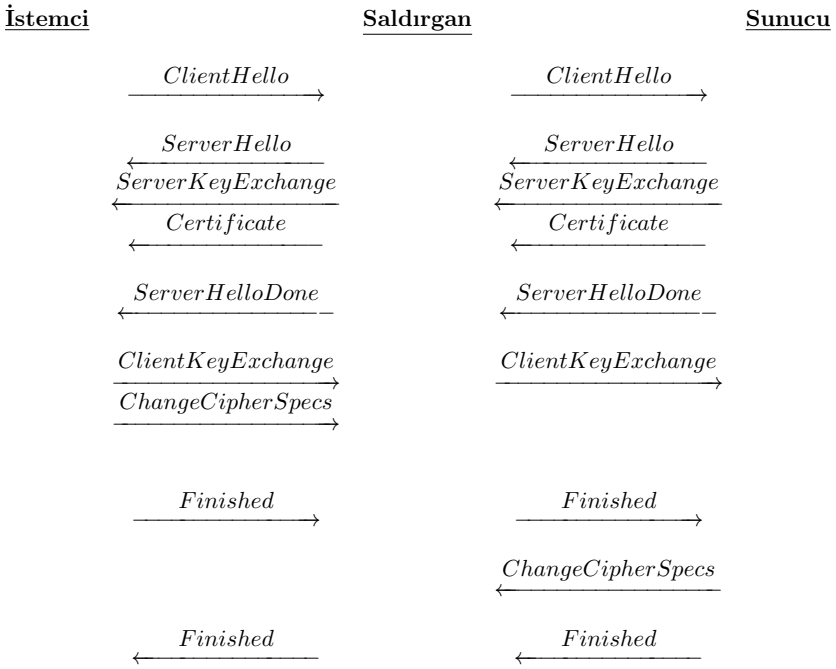
Alınan önlem ile el sıkışma protokolündeki açıklık kapatılmıştır fakat; uyarı protokolü ve şifreli iletişime geçme protokolü mesajları için hash değerleri alınmamıştır. Bu ihmal ilerleyen senelerdeki açıklıklara yol açmıştır.



Şekil 6.1: Şifre Paketi Düşürme Saldırısı

6.2 Şifreli İletişime Geçme Mesajını Düşürme

SSLv2'de bulunan basit ancak tehlikeli bir atak yine Wagner ve Scneier'in söz ettiği bir saldırı yöntemidir [97]. El sıkışma sürecinde taraflar birbirlerine şifreli iletişime geçme paketi *ChangeCipherSpec* göndererek, şifreli iletişime geçeceklerini bildir. Saldırgan; istemcinin bu paketini düşürerek sunucuya ulaşmasını engeller. Devamında gönderilen tamamlanma *Finished* mesajları ile taraflar uygulama verilerini paylaşacaklarını söyler-ler. İstemci tarafından şifreli iletişime geçme paketi gelmediği için şifreli iletişim olmadan açık halde veriler paylaşılır.



Şekil 6.2: Şifreli İletişime Geçme Mesajını Düşürme

Bu sorunun yaşanmaması için, *Finished* mesajının *ChangeCipherSpec* mesajları alın-madan gönderilmemesi gerektiği anlaşılmıştır.

Şekil 6.3: PKCS #1 Blok Şifreleme Formatı

6.3 MAC'in tuzlama uzunluğunu kapsamaması açıklığı

SSLv2, Wagner ve Schneierin bahsettiği üzere, mesaj yetkilendirme kodu (MAC) ile ilgili açıklık bulundurmaktadır [97]. SSLv2 ile yapılan MAC ekleme kısmı yalnızca veriyi ve maskeleme kısmını korumakta olup, maskeleme uzunluğu bilgisini açık metin halinde göndermektedir. Bu durum mesajın bütünlüğünü tehlikeye atmaktadır.

Bu şekilde bir açıklığa yol açılmaması için, gönderilen her bir bitin saldırgan tarafından kullanılabileceği göz önünde bulundurulmalıdır. Her türlü bilginin bütünlüğü ve önemi dikkate alınarak korunmalı, böylece saldırganın saldırı uzayı mümkün olduğunca küçültülmelidir.

6.4 Bleichenbacher Saldırısı

Bleichenbacher Saldırısı, PKCS#1 [53] standardındaki RSA ile şifrelenmiş seçili metinler üzerinde yapılan bir SSL protokolü saldırısıdır. RSA tabanlı şifre paketlerine yönelik olan bu saldırı, 1998 yılında Daniel Bleichenbacher tarafından ortaya çıkarılmıştır [14]. Genel senaryosu, saldırganın gizli veriyi bilmemesine rağmen gizli veriyi manipüle ederek bit bit ortaya çıkarabilmesidir.

6.4.1 Saldırının Ön Koşulları

Saldırının gerçekleşebilmesi için saldırganın sunucuya istediği mesajı şifreletebilmesi ve şifreli mesaja da ulaşabilmesi gerekmektedir. Sonrasında saldırgan tahminlerde bulunarak hedefini daraltır ve daha önce seçmiş olduğu şifreli bir mesajın açık haline erişir. Açık hali elde edilen şifreli paketin istemcinin sunucuya gönderdiği ve ön ana giz değerini içeren *ClientKeyExchange* mesajı olması durumunda, saldırgan oturum anahtarını hesaplayabilir ve tüm oturum verilerine erişebilir.

6.4.2 Saldırı Senaryosu

Bleichenbacher saldırısı, PKCS#1 standardındaki blok şifreleme metoduna gerçekleştirilen bir saldırıdır. PKCS, açık anahtar şifreleme standardı demektir. PKCS#1 standardına göre şifreleme yapılırken Şekil 6.3'deki format kullanılır:

Şifrelemeye hazır veri formatının ilk bayt 00 değeri, mesajın başladığı anlamına gelir. İkinci bayt 02 değeri de bu mesajın şifreleme mesajı olduğu anlamına gelir. Sonrasında dolgulama kısmı gelir. Dolgulama için rastgele ve sıfır olmayan baytlar oluşturulur. Sıfır değerinde bayt oluşturulmamasının sebebi, dolgulamanın nerede bitip esas verinin nerede başladığını belirlemek içindir. Ayrıca, PKCS standardı gereği, dolgulama en az 8 bayt olmalıdır. Bu da demektir ki; ilk iki bayt olan 00 ve 02 baytlarından en az 8 bayt sonra bir 00 baytı bulunmaktadır. Dolgulamanın bitişini belirten bu bayttan sonra gizlenecek veri başlamaktadır. Elimizdeki toplam veri uzunluğu k bayt olsun. Şifrelenecek veriyi bayt bazında olarak inceleyecek olursak;

- $B_1 = 00$
- $B_2 = 02$
- B_3 den B_{10} a kadar sıfır olmayan baytlar
- B_{11} den B_k ya kadar en az bir bayt 00 (verinin içinde de 0 olan baytlar bulunabilir)

RSA şifreleme yaparken, N ve e açık anahtar, p q ve d değerleri de gizli anahtardır. Şifreli metin c ile, Açık metin m ile ifade edilir. (Ayrıntılı bilgi için Bölüm 2'ye bakınız). N değerinin uzunluğu da k bayt olduğu için, N değeri tam sayı haline çevirildiğinde $2^{8(k-1)} \leq N < 2^{8k}$ aralığındadır. Bu bilgi ve yukarıdaki bilgiler saldırı yaparken tahminlerde ve denemelerde kullanılacağı için değerli bilgilerdir.

6.4.2.1 Seçili Şifreli Metin Saldırısı Senaryosu

Bleichenbacher saldırısını anlayabilmek için öncelikle, seçili şifreli metin saldırısı incelenmelidir. Bu saldırı, seçili şifreli metinler kullanılarak açık metini elde etme saldırısıdır [20]. Amaç; öğrenilmek istenen şifreli metini başka değerlerle karıştırmak, sunucuya gönderip şifrelemesini sağlamak ve elde edilen şifreli metinden istenilen veriyi elde etmektir. Bir saldırgan, klasik RSA algoritması ile şifrelenmiş bir mesajı şifrelemek için şu adımları izleyebilir:

- Şifrelenmiş c mesajını elde eder ($c^d \mod N = m \mod N$ şeklinde, c açık anahtarı ile şifrelenmiş).
- Rastgele bir s değeri seçer ve $s^e \cdot c \mod N \equiv c'$ olacak şekilde bir c' mesajı oluşturur.
- Saldırgan c' değerini sunucuya gönderir ve şifreyi çözmesini ister.

- Sunucu $m' \equiv (c')^d \mod N$ şifre çözme işlemini yapar ve geri gönderir.
- $c^d \mod N \equiv m$ olduğundan ötürü $m' \equiv s \cdot m \mod N$ yani $m \equiv m' \cdot s^{-1} \mod N$ olur. Bu şekilde saldırgan m mesajını elde etmiş olur.

6.4.2.2 Bleichenbacher Saldırısının Senaryosu

Bleichenbacher saldırısı bir seçili şifreli metin saldırısıdır. Ancak; saldırganın sunucuya şifresini çözmesi için göndereceği değerin PKCS #1 formatına uygun olması gerekmektedir. Bu yüzden saldırganın ilk önce göndereceği $c' \equiv c \cdot s^e$ değerini oluştururken, şifreli metin açıldığında oluşacak yapının PKCS #1 formatına uygun olmasını sağlayacak bir s değeri seçmelidir. Rastgele seçilen s değerlerinden (yaklaşık olarak 30000-130000 arası denemede bir adet) PKCS uyumlu değere ulaşılır. Her seferinde elde edilen s değerleri ile yeni bilgiler elde edilir ve saldırı yüzeyi küçülür. Birkaç milyon deneme sonrasında saldırgan m mesajını tam olarak saptayabilir. Bu yüzden saldırının diğer bir adı da 'Milyon Mesaj Saldırısı' dır. Senaryo olarak saldırganın $m \equiv c^d \mod N$ eşitliği ile şifrelenen c şifreli metnini çözmek istediğini varsayalım. Saldırgan s tamsayı değerini seçer ve aşağıdaki değeri hesaplar:

$$c' \equiv c \cdot s^e$$

Sonrasında c' değerini sunucuya gönderir. Eğer sunucu c' değerinin PKCS formatına uygun olduğunu söylerse, saldırgan gönderdiği $m \cdot s$ metninin açık halinin ilk iki baytının 00 ve 02 olduğunu anlamış olur. Sonrasındaki işlemlerin kolaylığı açısından PKCS uyumlu bir mesaja denk olabilen matematiksel bir değer seçelim:

$$B = 2^{8(k-2)}$$

Burada k sayısı N değerinin bayt olarak uzunluğu idi. B sayısı ise; ilk baytı 00 ikinci baytı da 01 olan ve geri diğer tüm baytları 00 olan bir sayıdır. Saldırganın ilk amacı 00 ve 02 ile başlayan bir değer olduğu için aşağıdaki gibi bir değer elde etmiş olur:

$$2B \leq ms \mod N < 3B$$

Bu şekilde $2B$ ile $3B$ arasındaki değerlere denk gelen $ms \mod N$ sayıları toplanabilir. Sonrasında başka kısıtlamalar da yapılarak m değerine ulaşılabilir. Bu aşamalar aşağıdaki gibidir:

- Adım 1 - Körleştirme: Şifresi çözülmek istenen mesajın tam sayı karşılığı olan c mesajı için farklı rastgele s_0 tam sayı değerleri seçilir.

$$c_0 \leftarrow c \cdot (s_0)^e \mod N$$

$$M_0 \leftarrow [2B, 3B-1]$$

$$i \leftarrow 1$$

- Adım 2 - PKCS uyumlu mesajlar bulmak: M_i aralığında bulunan s_i değerleri bulunur ve kaydedilir. Her bir s_i değerinin bulunduğu M_i aralığı, bulunduğu aralıktaki değerin aranılan mesajı içerip içermediğinin kontrolü yapılmak üzere sonraki aşamaya aktarılır.

- Adım 3 - Çözümleri daraltmak: Her bir s_i değeri için aşağıdaki kontrol yapılır:

$$[a,b] \in M_i \text{ ve } \frac{as_i - 3B + 1}{n} \leq r \leq \frac{bs_i - 2B}{n} \text{ için;}$$

$$M_i \leftarrow \bigcup_{(a,b,r)} \left[\max\left(a, \left\lceil \frac{2B+rn}{s_i} \right\rceil\right), \min\left(b, \left\lfloor \frac{3B+rn}{a} \right\rfloor\right)\right]$$

  işlemi ile M_i değerleri daraltılır. Sonraki adımda M_i içerisinde kalan değer yada değerlere göre sonuç belirlenir.

- Adım 4 - Sonuca Ulaşmak: Eğer M_i aralığı 1 adet değer içeriyorsa (yani $M_i = [a,a]$ ise) $m \leftarrow a(s_0)^{-1} \mod N$ olur. Bu da demektir ki, cevap $m \equiv c^d \mod N$ eşitliğini sağlayan m mesajı bulunmuştur. Eğer i değeri 1 adet değer içermiyorsa, 2. adıma dönerek bir sonraki değer için işlemler devam ettirilir.

6.4.3 Saldırıya Karşı Alınan Önlemler

Bu saldırının önüne geçebilmek için, sunucuların dolgulama aşaması sorunları ile ilgili ayrıntılı bilgi vermemesi gerekir. Eğer sunucuya PKCS#1 formatına uygun mesaj gelmemişse sunucunun bu bilgiyi geri dönmeyerek saldırıyı önlemesi mümkündür. Bunu sağlamak için de sunucu hata mesajı dönmek yerine rastgele bir ön ana giz değeri dönerek saldırganın fikir yürütebilmesini engelleyebilir.

Bu açıklıktan ötürü PKCS #1 tuzlama aşamasının gereksiz olduğu düşünülebilir. Ancak PKCS nin sonraki versiyonlarında OAEP adı verilen farklı bir tuzlama yöntemi eklemiştir [9]. OAEP yönteminde tuzlamanın yanı sıra özet fonksiyonu da kullanıldığı için Bleichenbacher saldırısına karşı dayanıklı haldedir. Ancak bu, SSL içerisinde halen PCKS#1 v1.5 kullanıldığı gerçeğini değiştirmemektedir.

6.5 Geliştirilmiş Bleichenbacher Saldırısı

Klima, Pokorny ve Rosa tarafından geliştirilen bir yönteme göre, Bleichenbacher saldırısı hem geliştirilmiş, hem de saldırıya karşı alınan önlem de etkisiz hale getirilmiştir [58]. Bleichenbacher saldırısına karşı alınan önlem, hata durumunda uyarı mesajı göndermeyip, rastgele bir $PreMasterSecret$ değeri oluşturmak ve el sıkışma senaryosuna devam etmektir. Böylece hem $Finished$ mesajı sırasında tarafların farklı anahtarlar oluşturduğu farkedilerek oturum sonlanacak, hem de saldırganın PKCS uyumlu mesaj oluşturup oluşturamadığını anlaması engellenecektir. Bununla birlikte SSLv3.0'dan SSLv2.0'a versiyon düşürme saldırılarının da önüne geçmek için alınan önlemler Bleichenbacher saldırısının da önüne geçebilmiştir [97]. Bu önlem ile $PreMasterSecret$ değeri ile birlikte sadece dolgulama değeri değil, oturum için anlaşılan en düşük ve en yüksek SSL/TLS versiyonu bilgisi de eklenmiştir. Ancak bu değişiklik yüzünden bazı implementasyonlar farkedilebilir uyarı mesajları göndermiştir (örn. OpenSSL'de $decode\_error$ gibi). Versiyon farklılığı ile oluşan bu uyarı mesajını kullanarak bir saldırgan tahminlerde bulunarak saldırıyı gerçekleştirebilir.

Geliştirilmiş Bleichenbacher saldırısının senaryosu yaklaşım olarak Bleichenbacher saldırı gibi olduğu için detaylı anlatılmamıştır. Bu saldırıdan çıkarılması gereken ders, bir saldırıyı engellemek için alınan önlemin yeni bir saldırıya yol açabilmesidir.

Klima, Pokorny ve Rosa, saldırı adımlarını paralelleme ile saldırıyı hızlandırabileceklerini de göstermişlerdir [58]. 2012 yılında Bardou, Focardi, Kawamoto, Siminonato, Steel ve Tsay tarafından saldırı çok daha optimize hale getirilmiş ve daha az işlemle gerçekleştirilebilmiştir [7].

6.6 BEAST

BEAST (SSL/TLS'e karşı tarayıcı açıklığı) saldırısı, TLSv1.0 ve önceki versiyonlarda ortaya çıkan, şifre blok zincirleme (Cipher Block Chaining - CBC) yönteminde bulunan bir açıklıktan kaynaklanmaktadır [95]. 2011 yılında Duong ve Russo tarafından ortaya çıkarılan bir saldırı yöntemine dayanmaktadır [31]. Saldırının geçmişi 2001 yılına dayansa da, pratik bir atak olmadığı düşünülmüş ve TLSv1.1 versiyonunda açıklık kapatıldığı için dikkate alınmamıştır. Fakat, TLS in yeni versiyonlarının çıkması eski versiyonların kullanılmaması anlamına gelmediği için açıklık devam etmiştir. Saldırı, el sıkışma gerçekleştikten ve simetrik anahtar ile gizli iletişim başladıktan sonra gerçekleşir. Taraflar simetrik şifreleme için AES ve şifreleme yöntemi için CBC modunu seçmişlerse, saldırıya açık hale gelirler.

BEAST saldırısını kritik yapan unsur, saldırıların nasıl gelişebildiğini göstermesidir. BEAST saldırısı kendisinden sonraki birçok saldırı için temel oluşturmuş ve CRIME gibi daha gelişmiş saldırılara yol açmıştır. Saldırının on yıl boyunca aktif olması ve tarayıcı geliştiricileri tarafından umursanmaması da saldırının on yıl boyunca açtığı yaraları büyütmüştür.

6.6.1 Saldırının Ön Koşulları

BEAST saldırısının gerçekleşebilmesi için aşağıdaki gereksinimlerin sağlanması gerekir:

- Saldırgan, kurbanın ağ trafiğini aktif bir saldırgan olarak kontrol edebilmelidir.
- İstemci, HTTPS kullanarak bir web sitesine bağlanmış ve TLS bağlantısı kurmuş olmalıdır. Bu TLS bağlantısının versiyonu TLSv1.0 olmalı ve anlaşılmış simetrik şifreleme yöntemi AES CBC kipi olmalıdır.
- İstemci saldırganın yönettiği bir web sitesine erişmiş olmalı ve kurbanın tarayıcısına *BEAST zararlı kodu* yüklenmiştir. *BEAST zararlı kodu* sadece http isteklerinin içerisine byte enjekte etmektedir.

İstemci tarafından sunucuya bir istek gönderildiğinde saldırgan bu paketi görebilir. Saldırının gerçekleşebilmesi için saldırganın bir şekilde kurbanı kendi sitesine girmeye yönlendirmesi gerekir. Bunu yaparak kurbana tekrar tekrar istekler gönderebilir ve böylece saldırıyı gerçekleştirebilir.

6.6.2 Saldırının Senaryosu

BEAST saldırısı CBC mod şifrelemeyi hedef alır. CBC mod ile şifreleme yapılırken kullanılan başlangıç vektörünün (Initialization Vector - IV) tahmin edilebilmesi ile saldırgan CBC modun güvenliğini Elektronik kod kitabı (Electronic Code Book - ECB) seviyesine kadar indirebilir [81].

6.6.2.1 ECB Güvenliği

ECB yöntemi, en basit blok şifreleme yöntemidir. Verilen data belirli uzunluktaki bloklara ayrılır ve her bir blok birbirinden bağımsız olarak şifrelenir. Bu yöntem ile şifreleme yapıldığında aynı verinin her bir şifrelenmesi aynı sonucu verir. Saldırgan bu açıklığı kullanarak aşağıdaki şekilde saldırı gerçekleştirebilir:

- Saldırgan çözmek istediği bloğun ECB ile şifrelenmiş halini ele geçirir. Kullanılan algoritmaya göre blok uzunluğu değişebilir. Örnek olarak AES-128 kullanıldığını ve 16 bayt uzunluğunda veri olduğunu kabul edelim.
- 16 baytlık bir açık metin oluşturur. Blok şifrelemede 1 bit değişmesi durumunda tüm bloğun etkilenmesinden ötürü saldırgan tüm bloğu birden tahmin etmeye çalışır.
- Önceki adımda oluşturduğu açık metni kurbana şifreletir ve elindeki şifreli metin ile aynı olup olmadığına bakar. Eğer değerler aynı çıkarsa tahmini doğrudur ve gizli bilgi bulunmuştur. Eğer değerler farklı çıkarsa yeni bir blok değeri oluşturmak için önceki adıma geri döner.

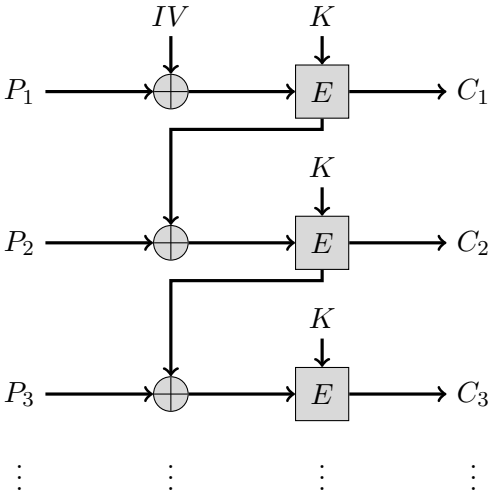
Saldırganın tüm bloğu tahmin etmesi gerekliliğinden ötürü bu saldırı efektif değildir. 16 bayt uzunluğundaki veriyi tahmin edebilmek için yaklaşık 2^{127} tahminde bulunması gerekir. Ancak, ilerleyen bölümlerde gösterileceği üzere, bu saldırının geliştirilmesi mümkün hale getirilebilir.

6.6.2.2 Tahmin Edilebilir IV ile CBC modu

CBC ve ECB arasındaki fark, CBC modunun her bir mesajı şifrelemeden önce IV kullanmasıdır (bkz. Şekil 6.4). Son blok hariç her bir şifreli blok, kendisinden sonra gelecek blok için de IV değeri görür. Böylece aynı metnin şifrelendiğinde her seferinde farklı sonuçlar vermesi sağlanır. Bu sayede ECB modundaki gibi deneme yapılarak veri ortaya çıkarılamaz.

Kullanılan IV değerinin etkili olabilmesi için her kullanılışında tahmin edilemeyen bir değer olması gerekir. Bu rastgeleliği sağlamak için düşünülen yöntemlerden birisi, her bir blok şifrelenmeden önce blok uzunluğunda rastgele değer üretilmesidir. Ancak bu pratik bir yöntem değildir. Çünkü; bir blok girdi için iki blok çıktı oluşur ve şifrelenen verinin açık verinin iki katına çıkmasına sebep olur. Hızlı bir çözüm olarak sadece ilk bloğun şifrelenmesi için rastgele değer üretilmesi ve sonraki her bloğun bir önceki bloğun çıktısını IV olarak kullanması yöntemi ortaya çıkmıştır. CBC ye adını veren zincir (chaining) kavramı bu şekilde ortaya çıkmıştır. SSLv3 ve TLSv1.0 da bu yöntem kullanılmıştır.

Saldırganın ilk IV değerini ya da hangi bloğun bir sonraki bloğun şifrelenmesinde IV olarak kullanıldığını bilmemesi durumunuda CBC modu güvenli kabul edilebilir. Fakat TLSv1.0 ve önceki versiyonlarındaki yaklaşıma göre sunucu ile istemci arasındaki tüm oturumda gönderilen paketler tek bir mesaj olarak kabul edilir. Yani ilk mesaj için bir IV değeri üretilir ve oturum bittiğinde tüm mesajlar bu IV ile başlayan zincirin parçasıdır.



$P_1||P_2||...||P_n$ = Açık Metin
$C_1||C_2||...||C_n$ = Şifreli Metin
K - Anahtar, E - Şifreleme Fonksiyonu
IV - Başlangıç Vektörü

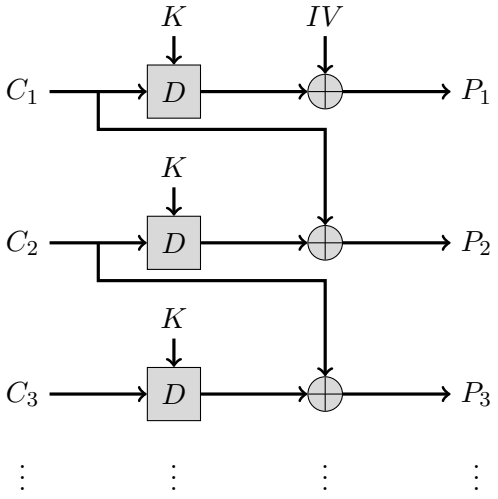
Şekil 6.4: CBC Kipi ile Şifreleme

Saldırgan aradaki bağlantıyı görebildiği için ve bir sonraki gizli verinin önceki gizli veri kullanılarak oluşturulacağını bildiği için BEAST saldırısına sebep olan açıklık ortaya çıkmış olur (TLSv1.1 ve yeni versiyonları için tüm oturum yerine gönderilen her kayıt için yeni IV kullanılmış ve saldırıdan kaçınılmıştır).

Saldırı, aktif saldırı yapan bir saldırganın araya şifrelenmesi için blok sokması ile ortaya çıkmaktadır. Saldırganın, şifrelenen son bloğun bir sonraki IV değeri olacağını bildiği için, CBC modunun sağladığı güvenlik ECB seviyesine düşer.

6.6.2.3 Saldırının Adımları

Bu saldırı senaryosunda örnek olarak saldırganın ikinci bloktan sonra saldırıyı gerçekleştirdiği varsayılsın. İlk iki blok tarayıcı tarafından gönderilir. Üçüncü blok ise saldırganın tarayıcıyı kullanarak gönderdiği bloktur. Saldırganın amacı, ikinci blokta yer alan veriyi çözmektir. İlk bloğun IV_1 değerinin bilemediği için ilk bloğu çözemez. Fakat ikinci blok IV değeri olarak birinci bloğun şifreli halini kullandığı için ikinci bloğa saldırabilir (IV_2 = ŞifreliBlok$_1$). İlk iki blok şifrelendikten sonra saldırgan tarayıcıyı ele geçirir ve kendisi şifreleme yaptırmaya başlar. Tahmini değerler oluşturup şifreletir ve şifreli hallerini kıyaslar. Tahmininin başarılı olması kendi şifrelettiği değerin (ŞifreliBlok$_3$) ikinci bloğun şifreli haline (ŞifreliBlok$_2$) eşit olması ile olur.



$P_1||P_2||...||P_n$ = Açık Metin
$C_1||C_2||...||C_n$ = Şifreli Metin
K - Anahtar, E - Şifreleme Fonksiyonu
IV - Başlangıç Vektörü

Şekil 6.5: CBC Kipi ile Şifre Çözme

IV değerlerinin bilinmesi ile saldırının nasıl CBC den ECB seviyesine gelebildiğini anlamak için aşağıdaki adımlar incelenebilir ($E(x)$ ifadesi x değerinin şifrelenmesi anlamına gelir. Şifreli bloklar C harfi ile, mesajlar M harfi ile gösterilir. $\oplus$ şekli xor işlemidir.):

$$C_2 = E(M_2 \oplus IV_2) = E(M_2 \oplus C_1)$$

$$C_3 = E(M_3 \oplus IV_3) = E(M_3 \oplus C_2)$$

Mesajlar ilk olarak IV değerleri ile xor lanır, sonrasında şifrelenir. Farklı IV değerleri(IV_2, IV_3) kullanıldığı için M_2 değeri M_3 değerine eşit olsa dahi C_2 ile C_3 birbirinden farklı olur. Ancak, saldırgan IV değerlerini bildiği için M_3 değerini IV değerlerinin etkisini kaldıracak şekilde ayarlayabilir. Bu nokta, saldırının kalbidir. Rastgele bir M_3 değeri seçilmeyip, seçilen rastgele bir değer (M_g) aşağıdaki gibi işleme sokulur ve saldırganın IV değerlerinden sıyrılması gerçekleşir:

$$M_3 = M_g \oplus C_1 \oplus C_2$$

M_3 değerinin şifreli hali ise aşağıdaki gibi olur:

$$C_3 = E(M_3 \oplus C_2) = E(M_g \oplus C_1 \oplus C_2 \oplus C_2) = E(M_g \oplus C_1)$$

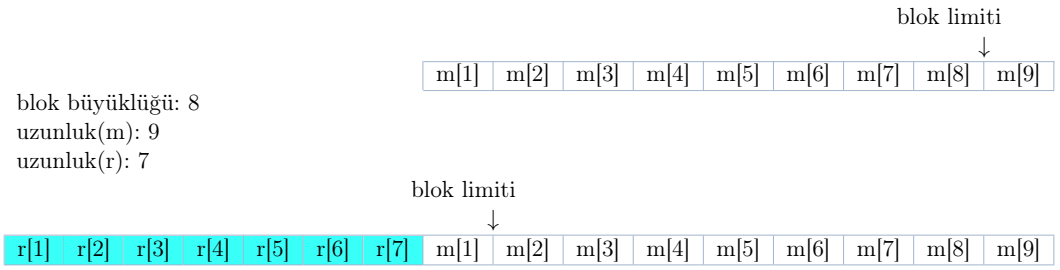
Eğer yapılan tahmin doğru ise ($M_g = M_2$ ise), aşağıdaki değer sağlanmış olur ve C_2 bloğunun içerdiği değer bulunur.

$$C_3 = E(M_g \oplus C_1) = E(M_2 \oplus C_1) = C_2$$

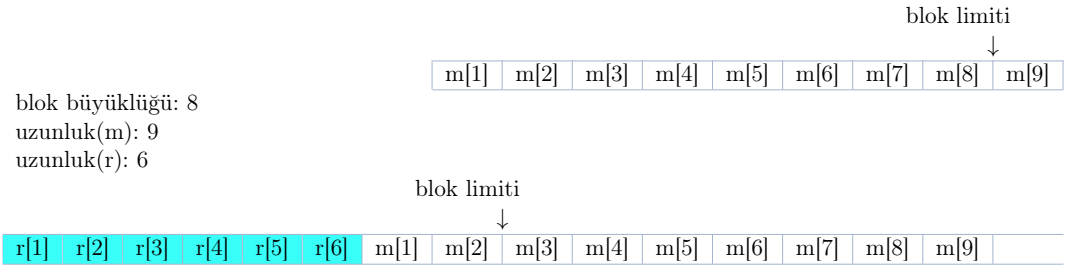
Tahmin edilebilir IV ile CBC modunun güvenliğinin ECB güvenliğine inmesine rağmen, ECB modunun güvenliği 16 bayt için halen 2^{127} deneme gerekmektedir. Bu yüzden BEAST saldırısı yıllar boyunca görmezden gelinmiş ve TLSv1.0 CBC modu ile kullanılmaya devam etmiştir. Buna karşın saldırıyı pratik hale getiren durum işlem gücünün artması değil, manipüle edilebilen küçük değerli veri parçaları (çerez, oturum bileti, parola vb.) olmuştur. Saldırganın tüm paketi değil, bir bayt değerini bile ifşa edebilmesi yeterli olabilmektedir. Duong ve Rizzo tarafından yeni bir saldırı modeli ortaya atılmıştır; 'blok halinde seçili sınır saldırısı' (blockwise chosen-boundary attack - BCBA) [31]. Bu saldırıya göre şifreleme için CBC mod kullanıldığı varsayılsın ve aşağıdaki tanımlamalar yapılsın:

b : blok büyüklüğü
m : dolgulama yapılmış mesaj
l : m mesajının bayt uzunluğu bilgisi($m[1], m[2], ..., m[l]$)
p : m mesajı s tane bloğa bölündüğünde blok değerleri($p_1, p_2, ..., p_s$)
x : iki blok arasındaki pozisyon değeri

Saldırıya göre, x değeri $m[b]$ ve $m[b+1]$ blokları arasındaki pozisyon olsun. BCBA modeline göre m mesajı şifrelenmeden önce saldırganın blok sınırlarını değiştirebildiği varsayılır (örneğin x değerini $m[1]$ den $m[2]$ ye taşıyabilir). Saldırganın blok sınırlarını değiştirmesinin yolu, mesajın arasına değer eklemesidir. Eğer saldırgan $r < m$ olacak şekilde r bayt uzunluğunda değeri m mesajına eklerse, mesajdaki tüm blok sınırları r pozisyon kadar sola kayar (bkz. Şekil 6.6). Bu şekilde blok uzunluğu 8 bayt ve m mesajı toplam 9 bayttır (dolgulama ile bu değer 16 olacaktır). Saldırgan, m mesajına 7 bayt ekleyerek blok pozisyonunu 7 bayt sola taşımıştır. Eklenmiş hali ile mesajın ilk bloğunu içeren 8 baytın ilk 7 baytı saldırgan tarafından bilinmektedir. İlk bloğun bilinmeyen tek bayt değeri sekizinci bayt yani $m[1]$ baytıdır. Bir bayt değeri ise $2^8 = 256$ denemede bulunabilir. Deneme ile $m[1]$ baytı bulunduktan sonra, saldırgan aynı saldırıyı baştan alır ve 7 bayt yerine 6 bayt ekler (bkz. Şekil 6.7). Bu sefer de $m[2]$ değerini bulabilmek için 256 deneme yapar. Toplamda 16×256 deneme ile (2^{14}) bir bayt değer elde edilir. Bölüm 6.6.2.1'de 2^{127} olan deneme sayısı burada 2^{14} de düşmüştür.



Şekil 6.6: BCBA Blok Kaydırma Yöntemi-1



Şekil 6.7: BCBA Blok Kaydırma Yöntemi-2

6.6.3 Saldırıya Karşı Alınan Önlemler

BEAST saldırısı istemci tarafında gerçekleştirilen bir saldırı olduğu için, ilk önlemler istemci tarafında alınmaya çalışılmıştır. OpenSSL 2004 yılında her bir TLS mesajı gönderilmeden önce boş bir mesaj gönderilerek saldırganın boş mesaja erişebilmesini sağlayan bir eklenti çıkarmışır. Ancak tarayıcılar boş uzunlukta şifreli mesaj gönderme kavramını uygulayamadılar. Henüz pratik atak geliştirilmediği için de bu önlem dikkate alınmamış oldu. 2012 yılında Adam Langley tarafından bulunan yöntem ile [62], gönderilecek mesajın ilk baytı ile geri kalan baytlar iki ayrı mesaj halinde gönderilip sadece ilk bayt tehlikeye atılmış oldu. Bu yöntem de bazı büyük sitelerin uyum sağlayamaması nedeniyle geç kabul gördü. Bununla birlikte sunucular da BEAST saldırısından istemcilerini koruyabilmek için CBC modu ile şifrelemeyi kabul etmemeye başladı.

6.7 CRIME

CRIME (Compression Ratio Info-Leak Mass Explotation) saldırısı Juliano Rizzo ve Thai Duong tarafından oturum anahtarlarının veya başka gizli bilgilerin ele geçirilmesini sağlayan yan kanal saldırısıdır. 2012 yılında ortaya çıkan bu saldırı, 2002 yılından bu yana bilinen bir açıklığın pratik halidir [56]. Taraflar arasında ortak oturum anahtarı oluşturulduktan sonra, çerezler de bu oturum anahtarı ile şifrelenirler. CRIME saldırısı da şifrelenmiş çerez içerisindeki gizli değeri ele geçirebilir.

6.7.1 Saldırının Ön Koşulları

Aynı oturum içerisinde tarayıcı tarafından her istekte aynı gizli değer/çerez gönderilir. Sıkıştırılmış metin şifrelenmiş olduğu halde şifreli halinin uzunluğu saklanamamaktadır. Bununla birlikte saldırgan, istemciye göndermek istediği veriyi sıkıştırmadan önce istediği bir değeri enjekte edebilme kabiliyetine sahiptir. Bununla birlikte saldırganın saldırıyı uygulanabilir hale getirebilmesi için aşağıdaki gereksinimleri de sağlaması gerekir:

- Saldırgan, kurbanın ağ trafiğini aktif bir saldırgan olarak kontrol edebilmelidir.
- İstemci, HTTPS kullanarak bir web sitesine bağlanmış ve TLS bağlantısı kurmuş olmalıdır.
- İstemci saldırganın yönettiği bir web sitesine erişmiş olmalı ve kurbanın tarayıcısına *CRIME zararlı kodu* yüklenmiştir. *CRIME zararlı kodu* sadece http isteklerinin içerisine byte enjekte etmektedir.

İstemci tarafından sunucuya bir istek gönderildiğinde saldırgan bu paketi görebilir. Saldırının gerçekleşebilmesi için saldırganın bir şekilde kurbanı kendi sitesine girmeye yönlendirmesi gerekir. Bunu yaparak kurbana tekrar tekrar istekler gönderebilir ve böylece saldırıyı gerçekleştirebilir.

6.7.2 Saldırı Senaryosu

Bu saldırı, TLS ile anahtar anlaşması yapıldıktan ve ortak bir simetrik anahtar ile oturum kurulduktan sonra gerçekleştirilir. Saldırganın amacı ise, ağ üzerinde şifreli halde iletilen "*cookie*" değerini ele geçirmektir. CRIME saldırı senaryosu, baş harflerin açıklaması ile özetlenebilir:

Compression Ratio Info leak Mass Exploitation

- **Compression (Sıkıştırma):** HTTPS ile haberleşme yapılırken sıkıştırma kullanılarak gönderilecek verinin daha az yer kaplaması sağlanır. TLS ile el sıkışması yapılırken de sıkıştırma yöntemleri belirlenir (bkz. Bölüm 4.3.1.2). Internet üzerinde kullanılan en popüler sıkıştırma yöntemlerinden birisi, DEFLATE dir [23]. DEFLATE algoritması temel olarak iki algoritmadan oluşur. Bunlar LZ77 ve Huffman Coding'dir [84]. Bu algoritmalar metni tarar ve yinelemeleri eler. LZ77 algoritması, birden fazla kez bulunan metin dizileri tekrardan yazılmaz, bunun yerine aynı değerin yazdığı bir başka adresi işaret eder.

(Örneğin: "Google is so googley" ifadesinde 'oogle' harf takımı ikinci kez tekrar edilmez ve bunun yerine "Google is so g(-13, 5)y" yazılır. Parantez içindeki ilk ifade kadar karakter geriye gidilir ve ikinci ifade kadar karakterin burada olduğu anlaşılır.) [84].

- **Ratio (Oran):** Sıkıştırma yapıldıktan sonra ne kadar çok metin küçülmüşse, o kadar çok benzer ifadeler şifreli metinde yer alıyor demektir. Eğer saldırgan, kurbanın şifreleyeceği metine bir metin parçası eklerse ve bu eklenmiş metinin şifreli halinin uzunluğuna bakarsa, şıkıştırmanın az yada çok olmuş olmasından, eklediği metin parçasının orjinal parça ile benzerliğini tahmin edebilir.

- **Info Leak (Bilgi Sızması):** TLS ile güvenli bağlantı kurulduktan sonra paylaşılan ortak anahtar ile şifreli iletişim yapılır. Bu aşamada gönderilen veriler şifrelidir. Fakat gönderilen verilerin uzunluğu şifreli olmadığı için, trafiği dinleyen birisi bu bilgiyi kolaylıkla öğrenebilir [81]. Saldırı da bu bilgi sayesinde gerçekleştirilebilir.

- **Mass Exploitation (Büyük Açıklık):** CRIME saldırısı, Mozilla ve Chrome başta olmak üzere bir çok platformda gerçekleşebilmektedir (CVE için bkz. [1]). 2012 yılında bu tarayıcılar toplam kullanımın %45 ine sahipti. Bununla birlikte web sunucularının %40 ı da bu açıklığa sahipti. Bunların arasında Dropbox, Github gibi büyük sunucular da bulunmaktaydı.

İstemci tarafından aşağıdaki gibi bir HTTP isteği yapılmış olsun [84]. Bu istek Şekil 6.8'teki gibidir:

```
POST / HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0)
Gecko/20100101 Firefox/14.0.1
Cookie: secretcookie=7xc89f94wa96fd7cb4cb0031ba249ca2
Accept-Language: en-US,en;q=0.8
(... body of the request ...)
```

Şekil 6.8: HTTPS İsteği Paketi

Bu HTTPS isteğinin uzunluğu $L = |(\text{şifrele}(\text{sıkıştır}(header + body)))|$ şeklindedir. Bu istek iletilirken şifrelenerek iletilmektedir. Saldırgan ağı kontrol ettiği için, bu isteğin şifreli halinin uzunluğunu bilmektedir. Saldırgan tarafından bilinen bir diğer bilgi ise; istemcinin sunucuya istekte bulunurken $Cookie : secretcookie = xxxx$ şeklinde bir çerez değeri de gönderdiğidir. Saldırgan, daha önce istemciye enjekte ettiği JavaScript kodu sayesinde istemcinin göndereceği bir pakete $Cookie : secretcookie = 0$ şeklinde metin parçası ekleyebilir. Bu eklemeden sonra HTTPS isteği Şekil 6.9'teki gibi görünür:

```
POST /secretcookie=0 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0)
Gecko/20100101 Firefox/14.0.1
Cookie: secretcookie=7xc89f94wa96fd7cb4cb0031ba249ca2
Accept-Language: en-US,en;q=0.8
(... body of the request ...)
```

ŞEKIL 6.9: Enjekte Edilmiş Paket

Yukarıdaki şekilde enjekte edilen değer ile başlayarak saldırgan *secretcookie* değerini tahmin etmek için aşağıdaki adımları izleyebilir (Not: Cookie değerleri her bir karakter için 64 farklı değer içerebilir [60].):

1. *secretcookie* = 0 ifadesi enjekte edilen paketin uzunluğu referans kabul edilir:
   $L = |$(şifrele(sıkıştır(secretcookie=0 değeri enjekte edilmiş paket)))$|$

2. secretcookie =0
   for j=1 to 32 (cookie karakter uzunluğu)
   for i=1 to 64;
   $L' = |$(şifrele(sıkıştır(secretcookie=i değeri enjekte edilmiş paket)))$|$
   if L' < L
   answer = i
   else;
   answer = 0
   return answer;
   secretcookie = secretcookie||answer
   return secretcookie

Bu şekilde 1 bayt uzunluğundaki bir karakter, maksimum 64 (bu değer optimize edilebilir [84]) deneme ile bulunabilir. 32 bayt değerindeki *secretcookie* değeri $64 \cdot 32 = 2^6 \cdot 2^5 = 2^{11}$ gibi kolay hesaplanabilir bir işlemle bulunabilir.

6.7.3 Saldırıya Karşı Alınan Önlemler

Saldırıya karşı alınabilecek en kolay ve mantıklı önlem TLS içerisindeki sıkıştırma yöntemlerini kullanmamak olmuştur. Hem Chrome hem de Firefox saldırıdan sonra TLS ile sıkıştırma özelliğini kaldırmış ve tarayıcıları güncellemiştir. Ancak halen TLS sıkıştırmasını destekleyen ve bü yüzden CRIME saldırısına karşı dayanıksız olan sunucular mevcuttur. Günümüzde CRIME ve türevleri saldırılara karşı dayanıklı olan bir sıkıştırma algoritması maalesef geliştirilememiştir.

6.8 TIME

CRIME saldırısından kısa süre sonra Tal Be'ery tarafından TIME saldırısı ortaya çıkarılmıştır [13]. CRIME saldırısının başarılı olabilmesi için saldırganın ağ paketlerini dinleyebilir olması kısıtı vardır. TIME saldırısı da sıkıştırma temelli bir saldırı olmasına rağmen saldırının bu kısıta ihtiyacı kalmamıştır. CRIME saldırısının gelişmiş hali olarak gösterilen TIME saldırısı, sıkıştırılmış verilerin büyüklüğünü giriş ve çıkış zaman farklılıklarından ölçerek gerçekleştirilir.

6.8.1 Saldırının Ön Koşulları

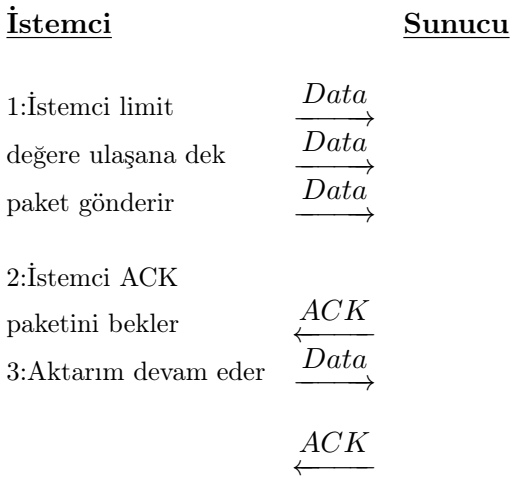
TIME saldırısının gerçekleşebilmesi için saldırganın;

- Saldırgan, istemcinin ağ trafiğini aktif bir saldırgan olarak kontrol edebilmelidir.
- Saldırganın ağ paketlerindeki süre farklılıklarını ölçebilmesi için istemci ile aynı yerel ağda bulunması gerekir.
- Kurban, HTTPS kullanarak bir web sitesine bağlanmış ve TLS bağlantısı kurmuş halde olmalıdır.
- Kurban saldırganın yönettiği bir web sitesine erişmiş olmalıdır. Bu sayede kurbanın tarayıcısına TIME saldırısı için gerekli olan zararlı kod yüklenebilmektedir.

6.8.2 Saldırı Senaryosu

TCP ile iletişim yapılırken (TLS protokolü TCP kullanır) taraflardan birisinin diğerine çok fazla data göndererek boğması engellenir. Taraflar arasında fiziksel olarak büyük uzaklıklar olduğunda bu durum sorun çıkarabilir. Örneğin bir paketin Londra'dan New York'a gitmesi yaklaşık 45 milisaniyedir [81]. Eğer gönderilen paketin iletilme onayı gelmeden başka bir paket gönderilmezse 90 milisaniye içinde sadece 1 paket gönderilebilir (round trip time - gidiş geliş gecikmesi için bkz. [78]). İletişimi hızlandırmak için TCP birden fazla paket gönderimine izin verir. Ancak aşırı yükleme olmaması için de onay gelmeden gönderilebilecek paket sayısına limit belirlenir. Şekil 6.10'da limit değeri 3 olması durumundaki akış gösterilmiştir.

Gönderilen veri yeterince küçükse tek seferde gönderilebilir. Fakat tek seferde gönderilemeyecek kadar büyükse önce gönderilebilecekler gönderilir ve ilk gönderilen için onay



Şekil 6.10: TCP Sıkışıklık Kontrolü Örneği

beklenir. Onay geldikten sonra gönderilmeye devam edilir. Bu gecikmenin süresi Londra-New York arası için 90ms yani gidiş geliş gecikmesi süresi kadardır. Bir bayt fazla olmasından ötürü istenen metin tek parça yerine iki parçada gönderilirse 90ms fark oluşur. Bu fark tarayıcıya yüklenen bir JavaScript ile ölçülebilir. Bu sayede saldırı gerçekleştirilebilir. Çünkü bu aşamadan sonra CRIME'da olduğu gibi sıkıştırma ile oynayarak metnin uzunluğunu ve dolayısı ile iki parçada gönderilip gönderilmediğini anlamak mümkündür. Eğer data manipüle edilebilirse ve her bir baytın doğru tahmin edilmesi halinde metin tek parçaya sığacak hale getirilirse CRIME saldırısı gibi tahmin aşamaları ile gizli değere ulaşılabilir.

6.8.3 Saldırıya Karşı Alınan Önlemler

TIME saldırısı CRIME saldırısının geliştirilmiş bir halidir ve CRIME için geçerli olan önlemler TIME için de geçerlidir. Bununla birlikte TIME saldırısı teorik bir saldırı olmanın önüne geçememiştir [81]. Çünkü bu saldırıyı gerçek hayata geçirmek için engeller bulunmaktadır. Örneğin, ağ paketlerinin farklı yol izleme veya ağ üzerindeki cihazların anlık performans değişikliklerinden ötürü 90ms lik farkı ölçmek pek mümkün olmamaktadır. Yine de saldırıyı ortaya çıkaran yazarın ifadesine göre tek bir bağlantı ve teker teker sorgu yaparak ilerleme ile bu saldırıyı gerçekleştirmek mümkün olabilir.

6.8.4 BREACH

HTTPS cevaplarına yönelik bir başka sıkıştırma saldırısı olan BREACH, Ağustos 2013 te ortaya çıkarılmıştır [43]. CRIME ve TIME saldırılarına benzeyen BREACH, HTTPS isteklerine değil, HTTPS cevaplarına yönelik olmasıdır. Saldırı, sunucuya tahminler içeren istekler gönderip, sıkıştırılmış cevabın uzunluğunun ölçülmesi şeklindedir. Gelen cevap

ne kadar kısa ise, tahmin edilen değerler de o kadar yakındır. CRIME saldırısında olduğu gibi bu saldırıda da TLS sıkıştırması özelliğinin kaldırılması ile koruma sağlanmış olur.

6.9 Lucky 13

2013 ün şubat ayında AlFardan ve Patersen tarafından TLSv1.0 ve 1.1 sürümlerinin yeni bir açıklığa sahip olduğu ortaya çıkarıldı. Bu açıklığa Lucky13 adı verildi [5]. Bu saldırı, araya giren saldırganın oturum anahtarı oluşturulurken Şifre-Bloğu Zincirleme (Cipher Block Chaining - CBC) kipi kullanılması halinde şifreli metinlerden açık metinleri ortaya çıkarabilmektedir. CBC kipi içeren bir şifre paketi ile oluşan oturumda şifre çözme işlemi yapılırken küçük zaman farklılıkları oluşur. Lucky13 saldırısı bu zaman farklılıkları kullanılarak gerçekleştirilir.

6.9.1 Saldırının Ön Koşulları

- Sunucunun SSLv3 veya TLSv1.0 versiyonlarını AES CBC kipinde desteklemesi gerekir.
- Saldırganın ağ paketlerindeki süre farklılıklarını ölçebilmesi için istemci ile aynı yerel ağda bulunması gerekir.

6.9.2 Saldırı Senaryosu

Mesaj Doğrulama Kodu (Message Authentication Code - MAC) mesajların bütünlüğünü ve kimlik doğrulamasını sağlamak için kullanılan bir yöntemdir [16]. Gönderilecek mesaj önce şifrelenir, sonra şifreli metinin MAC'i alınır. Ancak TLS protokolünde bu yöntem farklılık gösterir. Mesaj açık halde blok içerisine eklenir, sonra açık metine MAC uygulanır. Sonra da mesaj 255 bayt uzunluğuna ulaşana kadar dolgulama yapılır. Böylece mesaj, şifreleme blokları için uygun uzunluğa ulaşmış olur. Bu hali ile mesaj şifrelenir ve iletir. Mesajı alan taraf önce şifreyi çözer, sonra bir önceki şifreli blok ile XOR'lar. Açık metin elde edildikten sonra ilk olarak dolgulama kontrol edilir. Dolgulamada bir sorun olmaması halinde dolgu baytları silinir ve gönderilen verilerin bütünlüğünün kontrolü için MAC değeri hesaplanır.

Ancak, SSLv3 ve TLSv1.0 protokollerinde CBC kipi ile şifreleme yapılırken dolgulamanın korunması ile ilgili bir sorun bulunmaktadır. Dolgulama verisi MAC ile korunmamaktadır. Şifreli metine müdahale edebilen ve istemcinin yerine geçip sunucuya birçok mesaj

Tablo 6.1: AES CBC Kipi MAC ve dolgulama yöntemi

8 bayt	5 bayt	n bayt
Sıra	Başlık	Veriler
<————-MAC (20 bayt)————>		

Gönderilecek veriler paket sırası ve ön bilgisi ile birlikte MAC işleminden geçirilir. MAC sonucu uzunluğu kullanılan özet algoritmasına göre değişir, bu örnekte 20 alınmıştır.

n Bayt	20 Bayt	0-255 Bayt
Veriler	MAC	Dolgulama
<————————————Şifreleme————————————>		
Şifreli Metin		

Metin şifrelenirken bir önceki işlemde oluşturulan MAC değeri verilerin yanına eklenir. Blok uzunluğunu tamamlamak için de gerektiği kadar dolgu değeri eklenir.

gönderebilen bir saldırgan tuzlama değerlerinin uygunluğu üzerinden saldırı gerçekleştirebilir. BEAST saldırısında olduğu gibi, saldırgan şifresini çözülmesini istediği bir mesajı enjekte edebilir (bkz. Bölüm 6.6).

Şifreli metnin şifresinin çözülmesi aşamasında ilk önce tuzlama kontrol edilir ve eğer tuzlama düzgün ise MAC kontrolü yapılır. Bu aşamada sorun çıkması halinde sunucu, tuzlama kontrolü veya MAC kontrolü aşamasında hata olduğunu bildirir. Saldırgan dönen hata mesajlarına göre şifreli mesajları manipüle ederek mesajın şifresiz halini ortaya çıkarabilir.

Tuzlama üzerinden tahmin yürütülmesinin önüne geçebilmek için sunucunun farklı uyarı mesajları dönmesi yerine yalnızca uyarı mesajı gönderilmesi sağlandı. Böylece saldırgan her durumda aynı uyarı ile karşılaşmış oldu. Ancak yöntemin gerçeklenmesinde gözardı edilen bir durum vardı. Tuzlama hatası ile oluşan mesajlar, MAC hatası ile oluşan mesajlardan daha çabuk ulaştırılıyordu. Böylece yan kanal saldırısına karşı açıklık ortaya çıktı. Bunun önüne geçebilmek için ise TLSv1.2'de MAC kontrolünün tuzlama hatalı olsa dahi yapılması yöntemi ortaya çıktı [25]. Ancak; tuzlama hatalı olduğu zaman, mesajın ne kadarının verileri içerdiği, ne kadarının da tuzlama olduğu anlaşılamaz. Bu yüzden mesajın tamamına MAC uygulanır. Mesajın tamamına MAC uygulamak ise tuzlama çıkarıldıktan sonra MAC uygulamaya göre daha uzun sürer ve bu işlemler arasındaki zaman farkından Lucky13 saldırısı ortaya çıkmış olur [84].

Şifre çözme aşamasında ilk olarak dolgulama kontrol edilir. Eğer dolgulama geçerli ise ondan sonra MAC kontrol edilir. Bu aşamalardan birisinde sorun çıkması halinde sunucu uyarı mesajıyla sorunun dolgulamada veya MAC'de olduğunu bildirir. Saldırgan son şifreli bloğa değerler enjekte ederek saldırganın uyarı mesajı dönüş sürelerini kıyaslar (bloğa

enjekte yapılması kısmı BEAST saldırısındaki gibidir. bkz. 6.6). Geri dönüş sürelerindeki farklılıklar MAC işlemi için özet alındığında kaç bayt değerin işleme tabi tutulduğunun belirlenmesini sağlar. Örnek olarak Alfardan ve Patersen'in makalede örnek verdiği 85 baytlık mesajın yapısı Tablo 6.2'deki gibi incelenebilir.

TABLO 6.2: Sunucuya İletilen Şifreli Veri Paketi

Başlık	IV	Veriler	MAC	Dolgulama
5 bayt	16 bayt	44-p bayt	20 bayt	p bayt

Sunucu aldığı bu mesajın şifresini çözer (Tablo 6.3):

TABLO 6.3: Sunucuya İletilen Şifreli Veri Paketi

Veriler	MAC	Dolgulama
44-p bayt	20 bayt	p bayt

Sonra da Tablo 6.1'de MAC işlemine eklenmiş olan sıra ve başlık değerleri açık metne eklenerek MAC yapılarak bütünlük kontrolü yapılır (Tablo 6.4):

TABLO 6.4: Sunucuya İletilen Şifreli Veri Paketi

Sıra	Başlık	Veriler
8 bayt	5 bayt	44-p bayt

Eğer herhangi bir dolgulama yoksa, $p = 0$ ve veri uzunluğu $44bayt$ demektir. Bu durumda üç farklı senaryo vardır:

1. Dolgulama yanlıştır ve 57 bayt MAC işlemine tabi tutulmuştur.
2. Son bayt değeri $0x00$'dır ve 56 bayt MAC işlemine tabi tutulmuştur.
3. Son iki bayt değeri $0x01$ ve $0x01$'dir ve 55 baytMAC işlemine tabi tutulmuştur.

Yukarıdaki senaryolardan ikinci ve üçüncü arasında küçük bir işlem farkı vardır. Bu işlem farkı MAC işlemi yapılırken özet algoritmasının girdi değerinin bir bayt artmasıdır ve bu artış, özet alınmasını çok küçük de olsa geciktirir. Bu zaman farkı sayesinde saldırgan son iki baytın değerini anlayabilir. Bundan sonraki aşama ise bir kısmı bilinen şifreli bir metnin çözülmesidir. Bu aşama Bleichenbacher saldırısındakine benzer bir işlemdir (bkz. Bölüm 6.4).

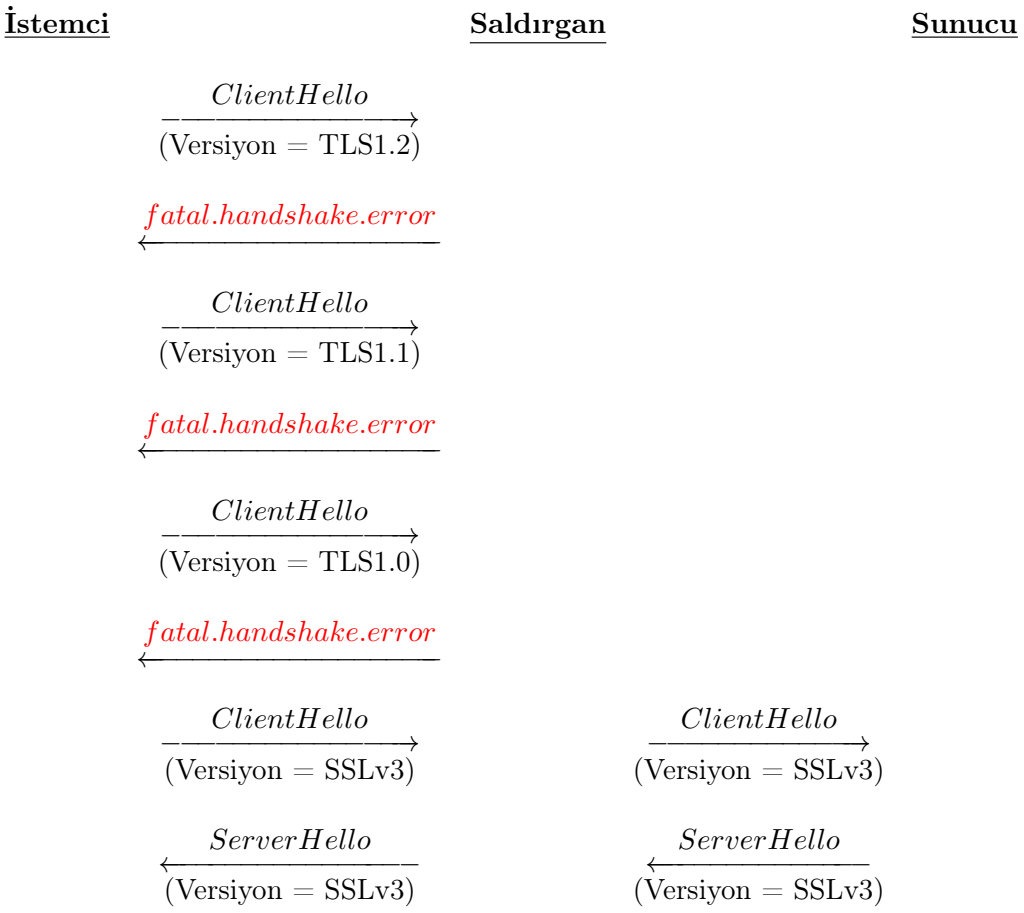
6.9.3 Saldırıya Karşı Alınan Önlemler

Lucky 13 saldırısından korunabilmek için MAC işlemlerindeki zaman farklılıklarının yansıtılmamasını sağlamak gerekir. Böylece saldırgan enjekte ettiği değerlerin açık metnini

tahmin edemez ve saldırıyı gerçekleştiremez. Bu zaman farklılıklarını sağlayabilmek için rastgele işlemler yapılabilir veya bekleme süreleri eklenebilir. Bununla birlikte, AES GCM ve AES CCM kiplerini kullanmak da Lucky 13 saldırısına karşı alınabilecek önlemlerdir.

6.10 POODLE

POODLE saldırısı, 2014 yılında Google güvenlik ekibi tarafından yayınlanmış bir saldırıdır [76]. SSLv3'ün kesin olarak güvensiz olduğunu ortaya çıkaran bu saldırı, tarafların TLS versiyonlar ile el sıkışma yapmak istemesi durumunda dahi bu saldırıya açık olunduğu gösterildi. TLS versiyonları ile el sıkışma başarısız olursa, taraflar SSLv3 ile konuşmaya çalışabilir. Bu durum ağ kesintilerinden kaynaklı olabileceği gibi, bir saldırgan tarafından da zorlanmış olabilir (bkz. Şekil 6.11).



Bu aşamadan sonra, Bölüm 6.10.2'deki saldırı senaryosu başlar

Şekil 6.11: POODLE İçin Versiyon Düşürme

Eski sistemlere uyum sağlayabilmek için, pek çok istemci versiyon düşürme dansı (downgrade dance) adı verilen süreci izler [29]. Bu süreçte ilk önce, istemci desteklediği en

yüksek versiyon (örn. TLSv1.2) ile güvenli iletişim kurmak istediğini söyler. Eğer bu girişim başarısız olursa protokol versiyonu düşürülerek tekrar denenir. Normal olarak beklenen süreç, taraflardan birisinin mevcut versiyonu desteklememesi halinde versiyon düşürmeye gidilmesidir. Ancak; ağ paketlerinin iletiminde oluşan bir kusur veya aktif bir saldırgan müdahalesi de versiyon düşürmeye yol açabilir. Bu yüzden, sunucu ile istemci arasındaki ağı kontrol eden bir saldırgan, tarafları SSLv3 ile iletişime ikna edebilir.

6.10.1 Saldırının Ön Koşulları

POODLE saldırısının gerçekleşebilmesi için aşağıdaki koşulların sağlanması gerekmektedir:

1. İstemcinin ve sunucunun SSLv3 ile iletişimi destekliyor olmalıdır.
2. İstemcinin, saldırganın zararlı yazılımı yükleyebilmesi için izin vermiş olması gerekmektedir. Bu durum sosyal mühendislikle yada istemcinin dikkatsizliği ile sağlanabilir. Yüklenen bu zararlı yazılım sonrasında, mesajların manipüle edilebilmesi ve tekrar tekrar gönderilebilmesi için kullanılacaktır.
3. Saldırganın, sunucu ve istemci arasındaki trafiği aktif saldırgan olarak kontrol altına almış olması gerekmektedir.

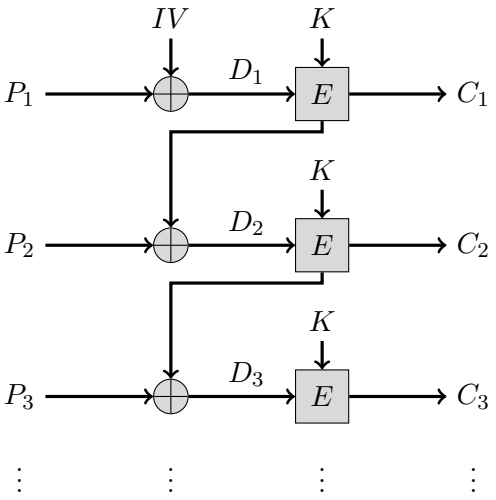
6.10.2 Saldırı Senaryosu

SSLv3 versiyonunda kullanılan simetrik şifrelemeler RC4 dizi şifreleme metodu ve CBC kipi ile blok şifreleme metodudur. RC4 metodu aynı mesajın birden çok şifrelenmesi durumunda bilgi sızdırdığı için güvensiz kabul edilmektedir [4]. POODLE saldırısı ise diğer metod olan CBC şifreleme yöntemine yöneliktir. SSLv3, CBC kip ile şifreleme yaparken PKCS#5 standardına göre hareket eder[54]. CBC kipi ile şifrelemenin en büyük sorunu, dolgulamanın uzunluğunun değişken olması ve MAC (mesaj doğrulama kodu) ile korunmamasıdır. Bundan ötürü, şifre çözülürken dolgulamanın doğrulaması yapılamaz.

SSLv3 ile CBC kipinde şifreleme yapılmadan önce, mesaj uzunluğunun bloklara bölünebilmesi için 1 ile L bayt arasında (L, blok uzunluğunun bayt değeri olsun) dolgulama yapılır [54]. Eğer son blokta dolgulama yapmak için yer kalmamışsa, sonraki blok dolgulama yapılarak gönderilir (bkz. Tablo 6.5). Mesajı alan taraf (sunucu olsun) mesajın son bloğunda bulunan son bayt değerini okur ve o değer kadar baytı siler. Sunucu bloğun tamamının (blok uzunluğu 16 bayt olsun) $0x0f = 15$ olduğunu düşünerek diğer baytlara bakma ihtiyacı duymaz. Silinme işleminden sonra kalan kısım mesajın kendisidir. Sonrasında ise MAC kodu ile mesajın bütünlüğü kontrol edilir.

TABLO 6.5: 64 Bitlik Dolgulama Örneği

	BLOK # 1								BLOK # 2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Örnek-1	E	L	M	A												
Örnek-1(Dolgulama)	E	L	M	A	0x04	0x04	0x04	0x04								
Örnek-2	A	N	A	N	A	S										
Örnek-2(Dolgulama)	A	N	A	N	A	S	0x02	0x02								
Örnek-3	Ş	E	F	T	A	L	İ									
Örnek-3(Dolgulama)	Ş	E	F	T	A	L	İ	0x01								
Örnek-4	P	O	R	T	A	K	A	L								
Örnek-4(Dolgulama)	P	O	R	T	A	K	A	L	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08
Örnek-5	M	Ü	R	D	Ü	M	E	R	İ	Ğ	İ					
Örnek-5(Dolgulama)	M	Ü	R	D	Ü	M	E	R	İ	Ğ	İ	0x05	0x05	0x05	0x05	0x05



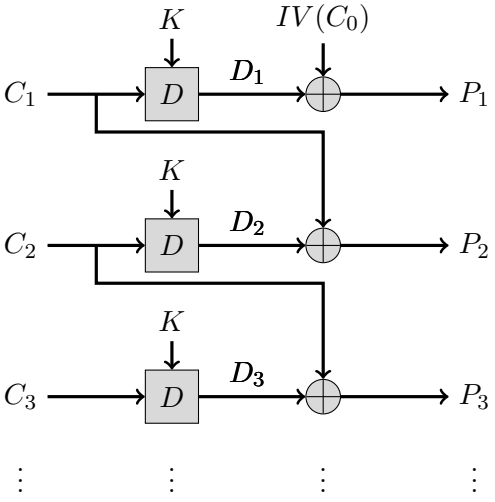
$P_1||P_2||...||P_n$ = Açık Metin
$C_1||C_2||...||C_n$ = Şifreli Metin
K - Anahtar, E - Şifreleme Fonksiyonu
$D_1, D_2...D_n$ - Ara Değerler
IV - Başlangıç Vektörü

ŞEKIL 6.12: CBC Kipi ile Şifreleme

Saldırının incelenebilmesi için Şekil 6.13'de bulunan değerler incelenebilir. C_1,C_2 ... C_n değerleri şifreli blokları, P_1,P_2 ... P_n değerleri de açık metin bloklarını gösterir. D_1,D_2 ... D_n değerleri ise şifre çözme işleminden çıkmış ve IV ile XOR'lanması halinde açık metini ortaya çıkaracak ara değerlerdir. Bu değerler ile aşağıdaki şekilde saldırı gerçekleştirilebilir:

1. Saldırgan enjekte etmiş olduğu script sayesinde erişmek istediği bloğu (C_3 olsun) son blok (C_n) ile yer değiştirerek gönderir (bkz. Şekil 6.14). Böylece saldırganın hedefi C_n içerisindeki değeri yani P_n'i bulmak olur.

2. Sunucu blokların şifrelerini çözer ve son bloğun son bayt değerine bakar. Burada gördüğü değer kadar baytın dolgu değeri olduğunu düşünür ve sondan başlayarak



$P_1||P_2||...||P_n$ = Açık Metin
$C_1||C_2||...||C_n$ = Şifreli Metin
K - Anahtar, E - Şifreleme Fonksiyonu
$D_1, D_2...D_n$ - Ara Değerler
IV - Başlangıç Vektörü

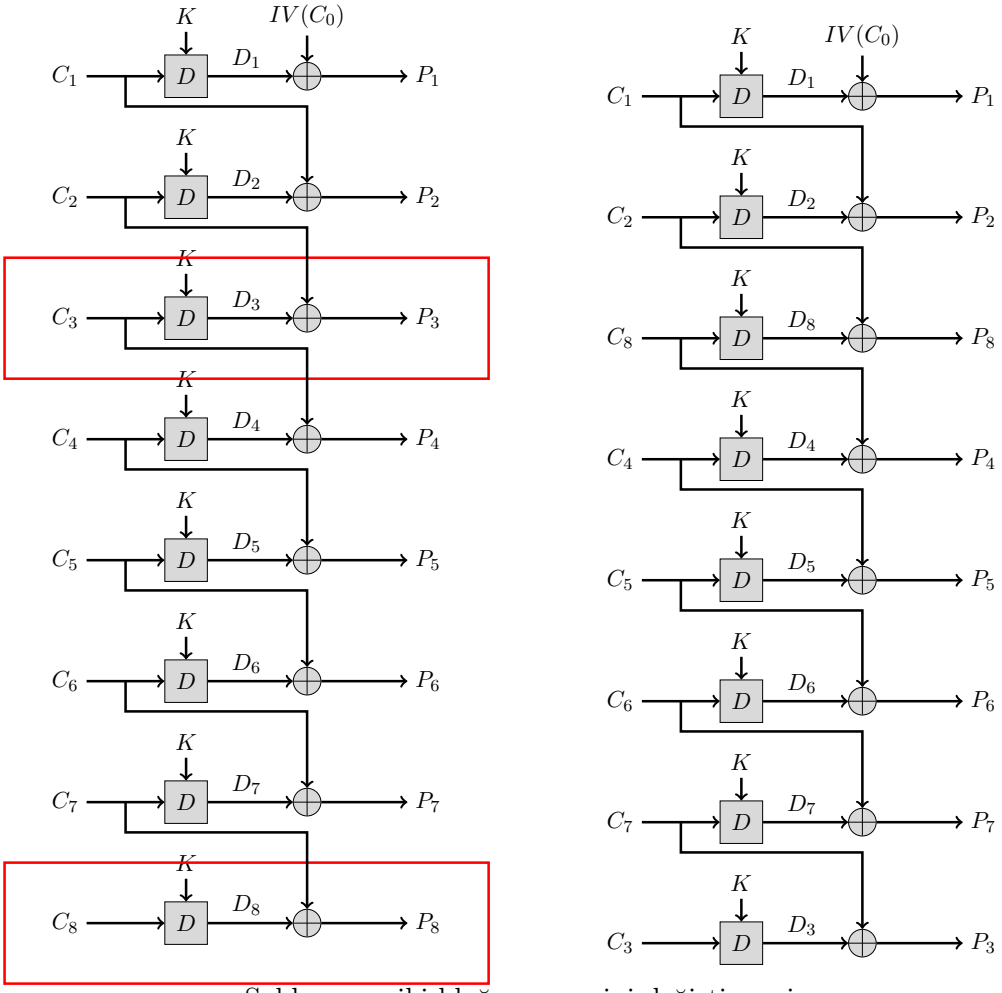
Şekil 6.13: CBC Kipi ile Şifre Çözme

o miktarda baytı siler (bu örnek için dolgulama miktarı 15 olsun yani tüm blok dolgu olsun).

3. Sunucu dolgu kısmını sildikten sonra mesajın doğrulamasını yapar. Saldırgan tarafından 3. blok ile son blok yer değiştirildiği için sunucuya gelen son bloğun son baytının hesaplanması ile ortaya çıkan değerin (bu değer $D_n[15] \oplus C_{n-1}[15] = P_n[15]$ işlemi ile hesaplanır) 15 olma olasılığı 1/256 olur. Bu da demektir ki; saldırgan yaklaşık olarak 256 denemeden 255 tanesinde sunucudan "dolgulama hatalı" hatası alırken 1 tanesinde almaz ve $D_n[15] \oplus C_{n-1}[15] = P_n[15]$ işleminin sonucunun 15 olduğunu öğrenmiş olur.

4. Saldırgan şifreli blokları bildiği için $C_{n-1}[15]$ baytının değerlerini de bilmektedir. Bir önceki adımdan elde ettiği $P_n[15]$ baytı değeri ile bu değeri XOR'laması halinde $D_n[15]$ baytının değerini öğrenebilir ($P_n[15] \oplus C_{n-1}[15] = D_n[15]$).

5. Saldırının ilk adımında üçüncü blok son blokla yer değiştirilidiği için halen doğru bir değere ulaşılamamıştır. Son bloğa taşınmadan önceki pozisyonu olan hedef bloğun, üçüncü bloktaki yerinde şifre çözülmesi yapılması halinde $C_{n-1}[15]$ baytı ile değil, $C_2[15]$ baytı ile XOR'lanması gerekir. Hedef bloğun tekrar üçüncü sıraya taşınması halinde $D_n[15]$ değeri $D_3[15]$ olmuş olur. Bundan sonra ise saldırgan $P_3[15] = C_2[15] \oplus D_3[15]$ işlemi ile hedef bloğun son baytını hesaplayabilir.



Saldırganın iki bloğunun yerini değiştirmesi.
Bu örnek için blok sayısı 8 ve değiştirilen bloklar $3-8$ olarak alınmıştır.
$P_1||P_2||...||P_n$ = Açık Metin
$C_1||C_2||...||C_n$ = Şifreli Metin
K - Anahtar, E - Şifreleme Fonksiyonu
$D_1, D_2...D_n$ - Ara Değerler
IV - Başlangıç Vektörü

ŞEKIL 6.14: POODLE Saldırısı

Yukarıdaki senaryoda tek bir bayt ele geçirilebilmiştir. Ancak, bloğun son baytı olmayan baytlar da hesaplanabilir. Bunu yapabilmek için saldırgan, istemci şifreleme yapmadan önce metine baytlar ekleyerek son bloğun son baytını kaydırabilir. Böylece 32 baytlık bir "cookie" değerini için 31 adet kaydırma ve ve her bayt için 256 deneme ile elde edebilir ($31x256 \approx 2^{13}$ işlem).

6.10.3 Saldırıya Karşı Alınan Önlemler

POODLE saldırısı SSLv3 protokolünün devre dışı bırakılması ile engellenebilir. Bununla birlikte istemciler, SSLv3 versiyonuna kadar versiyon düşürülmesini iptal ederek de kendilerini koruyabilirler. Ancak eski sistemlerle entegre olabilmek için SSLv3'ü kullanmak zorunda olan sunucular bulunmaktadır. Bu sunucuların saldırıdan korunabilmek için alternatif bir çözüm bulunmamaktadır.

6.11 Heartbleed

Heartbleed, yaygın bir TLS uygulaması olan OpenSSL kütüphanesinde bulunan bir güvenlik açığı ile ortaya çıkmıştır [32]. İsmini, açıklığa sebep olan heartbeat eklentisinden ötürü almıştır. Temel olarak heartbeat, bağlantı kurulan iki taraftan birisinin diğerine "Sana şu uzunlukta bir mesaj atıyorum, sen de bana gönder" tarzı bir mesaj atabilmeyi sağlar. Cihazların birbiri ile bağlantılarını kontrol etmelerini sağlayan ve normalde zararsız görünen bu özellik, kod geliştiricilerin gözden kaçırdıkları küçük bir ayrıntı yüzünden heartbleed açıklığı ortaya çıkmıştır. Eğer taraflardan birisi diğerine, "Sana 64 Kilobayt şu veriyi gönderiyorum" deyip, yalnızca 1 bayt gönderirse, cevap veren taraf, kalan 65534 bayt uzunluğundaki, olmayan bilgiyi vermek isterken, kendi hafızasındaki (RAM) verileri gönderir. Bu durumda, kötü niyetli bir kullanıcı, sunucunun hafızasında bulunan kritik verilere (diğer kullanıcıların anahtarları, şifreleri vs.) erişebilir. Bu bilginin uzunluğunun sunucu tarafından kontrol edilmemesi, heartbleed açıklığının temelidir. Bu açıklıktan faydalanmak isteyen bir istemci, normalde erişmemesi gereken bilgilere erişebilir.

Heartbeat eklentisi; birbiri ile bağlantı kuran cihazların, birbirleri ile iletişimlerinin aktif olup olmadığını kontrol etmek için gönderilen periyodik sinyallerden oluşan, keepalive ve watchdog benzeri bir protokoldür. Protokolün temel çalışma mantığında; istemci, sunucunun çalışır durumda olduğunu test edebilmek için bir istekte bulunur. Bu isteğin içinde beklediği cevabın uzunluğu da istemci tarafından verilir. Şekil 6.15'deki kod parçasında da görüldüğü üzere, 16 bit uzunluğundaki payload-length (istenilen veri uzunluğu) değişkeni istemcinin gönderdiği bir değerdir ve heartbeat mesaj yapısındaki değişkenlerden birisidir.

```
struct {
HeartbeatMessageType type;
uint16 payload-length;
opaque payload[HeartbeatMessage.payload_length];
opaque padding[padding_length];
} HeartbeatMessage;
```

Şekil 6.15: OpenSSLv1.0.1 Heartbeat Mesaj Yapısı

Kötü niyetli bir kullanıcı tarafından bu değişken manipüle edilerek maksimum değer olan (16 bit değiken için) 65535 yapılır. Sunucu, heartbeat mesajına cevap dönerken mesaj uzunluğu bilgisi olarak istemciden aldığı değeri esas alır. İstemcinin kötü niyetli olarak istediği veri uzunluğunun, istemcinin normalde istediği verinin olması gereken uzunlukla aynı uzunlukta olup olmadığını kontrol etmez. Bu kontrolü yapmadığı için de, istemcinin istediği cevabı verdikten sonra, hafızasında(RAM) bulunan ve olası gizlilik ihlalleri içeren verileri de (diğer kullanıcılara ait şifre ve anahtar bilgileri gibi) istemci ile paylaşabilir.

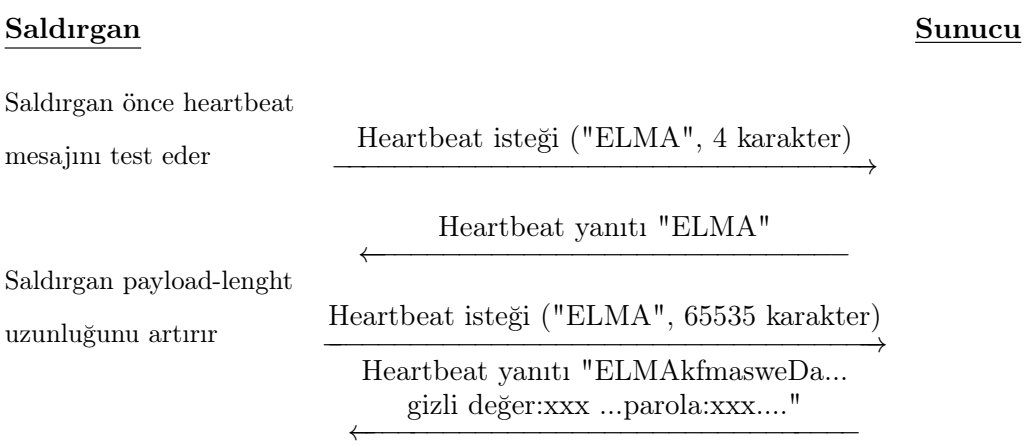
6.11.1 Saldırının Ön Koşulları

Saldırının gerçekleşebilmesi için aşağıdaki koşulların sağlanması gerekir:

- Saldırılmak istenen sunucu veya istemcinin SSL/TLS uygulaması olarak OpenSSLv1.0.1 ile OpenSSLv1.0.1f arası versiyonlardan birisini kullanıyor olması gereklidir.
- Kurban cihazın heartbeat mesajlarına cevap vermesi gereklidir.
- Saldırganın kurban ile SSL/TLS bağlantısı kurmuş olması gereklidir.

6.11.2 Saldırı Senaryosu

Heartbleed saldırısını gerçekleştirmek oldukça basittir. Hedef IP adresine *payload − length* uzunluğu artırılan bir heartbeat mesajı gönderilerek uygulanabilir (bkz. Şekil 6.16). Bununla birlikte Kali işletim sistemi içerisinde bu saldırı yardımcı programlar ile denenebilir [33].



Şekil 6.16: Heartbleed Saldırı Senaryosu

6.11.3 Saldırıya Karşı Alınan Önlemler

Heartbleed saldırısına karşı OpenSSL tarafından alınan önlem, heartbeat mesajında Şekil 6.15 de gösterilen $payload - lenght$ değişkeninin uzunluk değeri ile uyuşup uyuşmadığını kontrol etmek olmuştur. Bununla birlikte birtakım sunucuların heartbeat mesajlarını reddetmesi de saldırıya karşı koruma sağlamıştır.

6.12 FREAK

2016 yılının Ocak ayında, OpenSSL tarafından CVE-2015-0204 kodlu bir açıklık yayınlandı. Bu açıklığa göre istemci, ithal RSA anahtarı içeren şifre paketleri ile el sıkışmayı kabul etmektedir. İstemci başlangıçta bu şifre paketlerini önermemiş olsa dahi sunucunun isteği ile tercih etmektedir. Bu açıklık ortaya çıktıktan 2 ay sonra FREAK (Factoring RSA Export Keys - İthal RSA Anahtarlarını Hesaplama) adı verilen saldırı ortaya çıktı ve *www.nsa.gov* sitesine ortadaki adam saldırısı ile sunuldu [12].

6.12.1 Saldırının Ön Koşulları

FREAK saldırısının gerçekleşebilmesindeki temel neden, 1990 larda ortaya çıkan ithal şifre paketlerine dayanmaktadır.İthal şifre paketleri kavramı Amerika Birleşik Devletleri'nin 1992 yılında kriptografik protokollerin ülke dışında kullanımını kısıtlayan bir yasadan ötürü ortaya çıkmıştır [18]. Bu yasaya göre şirketler ABD dışına kripto ihraç ederken zayıf şifreleme yöntemleri kullanmak zorundadır. Bu kıstasa uygun şifre paketlere export paketleri denir ve bu paketlerde RSA ile şifreleme için maksimum değer, 512 bittir.2000 yılında ABD'nin ithal kripto ile ilgili politikasında yumuşamaya gidildi ve ithal şifre paketleri kullanım zorunluluğu ortadan kalktı. Ancak, ithal şifre paketlerini pek çok SSL/TLS kütüphanesi desteklemeye devam etti. Yıllar içerisinde ithal şifre paketlerinin anahtar boyları aynı kaldı ve buna karşın bilgisayarların işlem gücü arttı. İthal şifre paketleri de zaman geçtikçe daha güvensiz hale geldi.

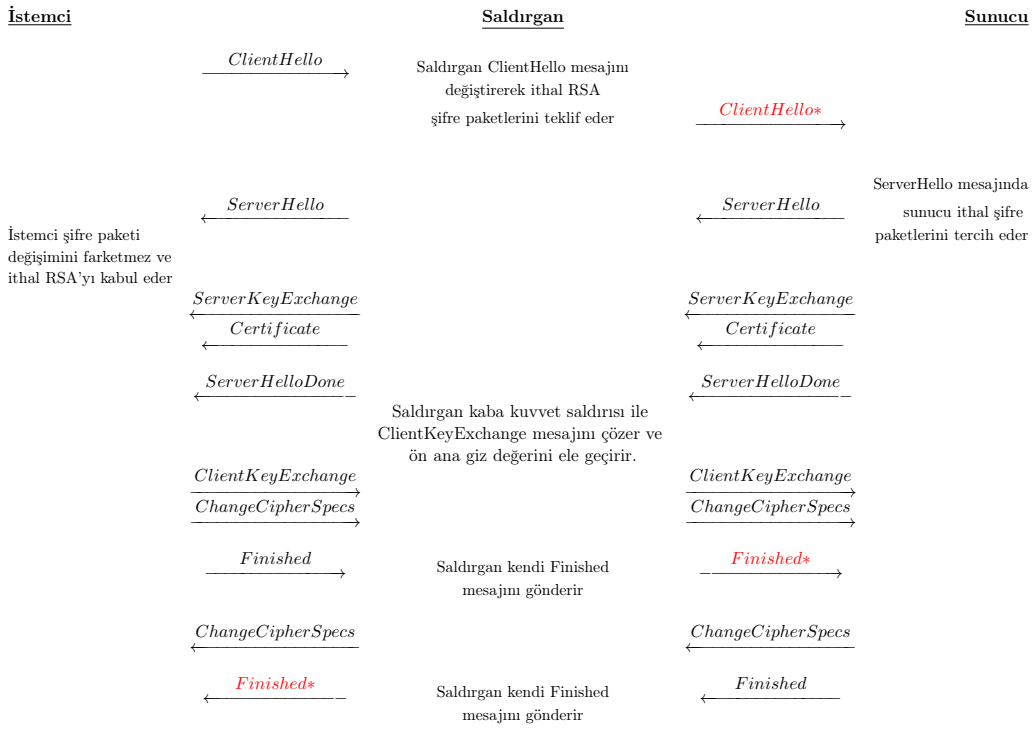
6.12.2 Saldırı Senaryosu

RSA ile yapılan bir el sıkışma sürecinde, istemci rastgele bir ön ana giz (premaster secret) değeri üretir ve bu değeri sunucunun açık anahtarı ile şifreleyerek sunucuya gönderir. Anahtar anlaşmasının güvenliği kullanılan RSA anahtarının güvenliği kadardır. İthal şifre paketleri ile anlaşma yapılırsa sunucu zayıf bir RSA anahtarı oluşturur, sonrasında güçlü bir anahtar ile (kendi gizli anahtarı ile) imzalar ve zayıf anahtarı $ServerKeyExchange$

(bkz. Bölüm 4.3.3) mesajı ile istemciye iletir. Sonrasında istemci mecbur olarak bu zayıf anahtar ile ön ana giz değerini şifreler. Oluşturulan anahtar zayıftır fakat güçlü bir imza algoritması kullanılmış ise mesajlar anlık olarak değiştirilemez ve bu yüzden aktif saldırı yapılamaz.

Günümüz bilgisayarlarının güçleri düşünüldüğünde ithal şifre paketleri anahtar boyu anlamında zayıftır. Kullanılması durumunda zayıf anahtar ile anlaşma sağlanır. Saldırgan el sıkışma mesajlarına müdahale edemese bile tüm trafiği pasif saldırgan olarak kaydeder ve sonrasında kaba kuvvet saldırısı (brute-force) ile ön ana giz değerini bulur ve tüm şifreli metinleri çözer.

Normal bir RSA anahtar anlaşmasında *ServerKeyExchange* mesajı protokol gereği olarak kullanılmaz. Ancak bazı TLS uygulamalarında bu mesaj kabul edilmektedir. FREAK saldırısını gerçekleştirmek için de saldırganın yapması gereken şey, bu mesajı istemciye kabul ettirmek ve kurulacak bağlantının zayıf olmasını sağlamaktır. Bunu yapabilmek için iki engel vardır. Birincisi, enjekte edilen veri hedef sunucunun kullandığı imza ile imzalanmalıdır. İkincisi ise saldırganın yaptığı müdahalenin geçerli olması için *Finished* mesajını da düzgün bir şekilde oluşturması gerekir.



Şekil 6.17: Freak Saldırı Senaryosu (Kırmızı mesajlar saldırganın oluşturduğu mesajlardır.)

Birinci engelin aşılabilmesi için, yani sunucunun zayıf anahtarı imzalamasını sağlamak için, yalnızca ithal şifre paketlerini kabul eden sunuculara yönelik saldırı gerçekleştirilebilir. Bu sayede saldırgan sunucuya yalnızca ithal şifre paketlerini sunarak saldırıyı

başlatabilir. Bu sayede sunucunun *ServerKeyExchange* mesajında ithal şifre paketlerinden birini seçerek göndermesi sağlanır.

Saldırganın doğru bir şekilde *Finished* mesajını oluşturabilmesi ise daha büyük bir engeldir. Bu mesaj, şifreli iletişime geçildikten sonraki mesaj olduğu için simetrik anahtarla şifrelidir ve tüm el sıkışma sürecindeki mesajların özetlerini içerir (mesajlar ve özetlerinin akışı için bkz. Bölüm 3.6). Bu mesajla taraflar birbirlerine doğru mesajların ulaştığını teyit eder ve ortak simetrik anahtar (ithal şifre paketi ile oluşan zayıf anahtar) ile şifreleyip birbirlerine gönderir. Mesajlar şifreli olduğu için saldırganın mesajları değiştirmesi tüm mesajı bozar. Bu yüzden *Finished* mesajları paylaşılmadan önce saldırgan güçlü bir bilgisayara sahip ise zayıf RSA anahtarını kaba kuvvet ile kırabilir ve iki tarafa da kendi finished mesajlarını gönderebilir.

Yukarıda anlatılan iki engeli aşan birisi, FREAK saldırısını gerçekleştirebilir.

6.12.3 Saldırıya Karşı Alınan Önlemler

FREAK saldırısının başlangıçta yalnızca OpenSSL için geçerli bir saldırı olduğu düşünülmüş fakat sonrasında pek çok mobil cihazın da saldırıya karşı dayanıksız olduğu ortaya çıkmıştır [45]. Bu yüzden FREAK saldırısının önlem alınana kadar ne çapta bir zarar verdiği tahmin edilememektedir.

FREAK saldırısından korunabilmek için ithal şifre paketlerinin kullanımı kesinlikle kaldırılmalıdır. Geriye yönelik uyumluluk sağlayabilmek için sunucuların eski yöntemleri de desteklemesi saldırılar için açık kapı bırakmaktadır.

6.13 Logjam

Logjam saldırısı, 2015 yılında bir grup araştırmacı tarafından Diffie-Hellman anahtar anlaşma protokolüne yapılmış bir saldırıdır [3]. Bu saldırı yöntem olarak FREAK saldırısına benzerlik göstermektedir. İki saldırıda da ortak olan durum ithal şifre paketlerinin kullanılmasıdır. Ancak Logjam, hem saldırı senaryosu hem de Diffie-Hellman'ı hedef almasından ötürü kritik bir saldırıdır.

6.13.1 Saldırının Ön Koşulları

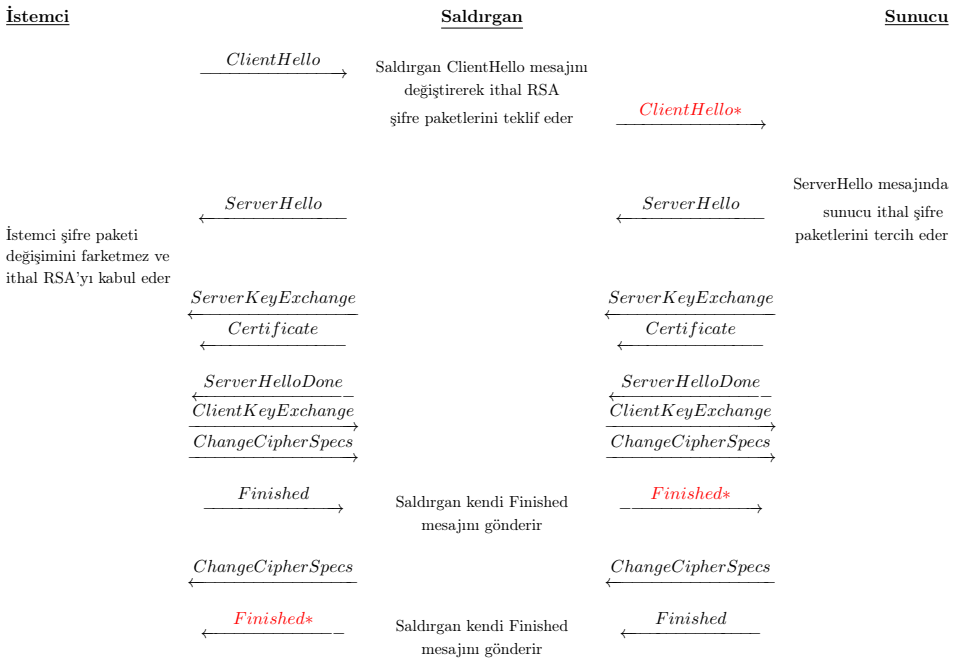
Saldırının ön koşulu, sunucu ve istemcinin ithal şifre paketlerini kullanmalarıdır. Günümüzde kullanılan Diffie-Hellman protokolünün son hali (bkz. Bölüm 3.6) ileriye yönelik gizliliği sağlamaktadır fakat ithal şifre paketleri ile yapılan el sıkışmalarda (bkz. Bölüm

3.2) geçici değerler kullanılmamaktadır. Bu yüzden taraflar arasındaki oturum anahtarı her seferinde aynı olmaktadır. Bununla birlikte ithal şifre paketlerinde Diffie-Hellman kullanılırken sonlu grubun boyu 512 bit olmak zorundadır. Logjam saldırısı, bu anahtar büyüklüğünün yeterince güvenlik sağlamaması nedeniyle mümkün olmuştur.

Logjam saldırısının teknik olarak özeti, Diffie-Hellman ile el sıkışmasında kullanılacak p asal sayısı ile ayrık logaritma problemini, dolayısıyla da Diffie-Hellman problemini çözmektir (Diffie-Hellman problemi için bkz. Bölüm 2.2.2).

6.13.2 Saldırı Senaryosu

Günümüzde web sunucularının neredeyse tamamı anahtar uzunluğu yüksek ve güvenli algoritmalar kullanmaktadır. Fakat, istemcinin talep etmesi halinde ithal şifre paketlerini kabul eden sunucular da bulunmaktadır. En popüler 1 milyon web sitesinin %7'si ve mail sunucularının %29'u bu sunuculardır [3]. Logjam saldırısı bu 512 bit uzunluğundaki ithal Diffie-Hellman anahtar anlaşma protokolüne karşı gerçekleştirilir. Bununla birlikte anahtar anlaşması yapılırken tarafların hangi parametrelerle şifreli iletişim kuracağını belirlediği $ClientHello$ ve $ServerHello$ mesajlarının açık halde gönderilmesi de Logjam saldırısını mümkün kılan bir diğer etkendir. Saldırgan $ClientHello$ mesajı sırasında araya girip istemcinin tercih ettiği güvenli bir şifre paketi yerine 512 bitlik Diffie−Hellman şifre paketi gönderebilir. Saldırının bundan sonraki aşamaları FREAK saldırısındaki gibidir:



Şekil 6.18: Logjam Saldırı Senaryosu (Kırmızı mesajlar saldırganın oluşturduğu mesajlardır.)

1. İstemci *ClientHello* mesajını sunucuya gönderir. Bu mesajın içinde Bölüm 4.3.1.2'de sunulan değerler bulunur. Saldırının gerçekleşebilmesi için Diffie−Hellman ile el sıkışma yapılması gerekir. RSA ve ECDHE ile kurulan bağlantılara Logjam saldırısı uygulanamaz.

2. Saldırgan *ClientHello* mesajını engeller ve şifre paketini ithal Diffie−Hellman içeren şifre paketi ile değiştirir.

3. Sunucu, ithal şifre paketlerini kabul eder ve ona uygun parametreleri istemciye gönderir.

4. İstemci bu aşamada sunucudan gelen parametrelerin kendi istediği Diffie−Hellman yöntemine uygun olduğunu farkedemez. Gelen değerlere göre ortak anahtar oluşturulur.

5. Saldırgan, sunucunun Diffie-Hellman el sıkışmalarında kullandığı p değerini önceden öğrenmiştir. Bu sayede sunucu ve istemci ile eş zamanlı olarak oturum anahtarını oluşturabilir.

6. *Finished* mesajı içerisinde el sıkışma paketlerinin özet değerleri bulunur ve bu sayede araya giren saldırı tespit edilebilir. Ancak saldırgan el sıkışma bitmeden Number Field Sieve algoritması sayesinde oturum anahtarını elde eder ve kendisi *Finished* mesajını oluşturur. Tarafların birbirlerine göndermek istedikleri *Finished* mesajlarını engeller ve kendi oluşturduğu mesajları gönderir.

7. Saldırgan oturum anahtarı ile aradaki bağlantının şifresini çözebilir ve hem aktif hem de pasif olarak saldırı gerçekleştirebilir.

6.13.2.1 Number Field Sieve Algoritması

Sonlu gruplar içerisinde ayrık logaritma probleminin çözümü için en hızlı ve pratik yol; Number Field Sieve (NFS) algoritmasıdır [66]. NFS kullanılarak 512-bit uzunluğunda ayrık logaritma işlemi 4000 çekirdek işlem gücü ile yaklaşık bir haftada çözülebilir. Bu demektir ki; yeterince uzun anahtar boyları için gerçek zamanlı olarak ayrık logaritmayı çözmek NFS algoritmasıyla dahi mümkün değildir.

Ancak, NFS algoritması biri diğerini takip eden iki farklı alt işleme bölünebilmektedir:

1. Verilen bir p asal sayısı için ön hesaplama yapılması;
   - Polinom seçme
   - Eleme (sieving)

- Lineer Cebir

gibi yalnızca p'ye bağlı işlemler gerçekleşir. Burada elde edilen çıktılar bir tabloda tutulur.

2. Verilen $g^a \mod p$ için a değerini bulmak. İniş (descent) adı verilen bu son aşamada, ilk adımda elde edilen tablo kullanılır.

Dikkat edilmesi gereken husus, ilk aşamada gerçekleştirilen tüm işlemlerin gereken işlem gücünü büyük oranda kapsamasıdır. Bununla birlikte ilk aşamada yazılan işlemler ikinci aşamadan bağımsızdır. İkinci kısım ilk kısıma nazaran çok daha verimli gerçekleştirilebilmektedir. Örneğin 512 bit için NFS algoritmasının ikinci kısmı yalnızca birkaç işlemci/dakika sürede sonuç dönmektedir. Bu işlem, gerçek zamanlı olarak uygulanabilir. NFS algoritmasının ilk kısmının önceden yapılması ile Logjam saldırısı pratik olarak mümkün olmaktadır.

6.13.3 Saldırıya Karşı Alınan Önlemler

Logjam saldırısına karşı alınabilecek en kesin çözüm, ithal şifre paketlerinin kullanılmasını ortadan kaldırmaktır. Ancak halen ithal paketleri destekleyen sunucular bulunmakta ve TLS uygulamalarının kurulumunda varsayılan olarak desteklenmektedir.

Bununla birlikte web tarayıcılarının anahtar anlaşmalarında minimum anahtar boyuna müdahale etmeleri de etkili bir önlemdir. Günümüzde önde gelen tarayıcılar (örn. Chrome, Firefox vb.) anahtar boylarını en az 768-bit olarak belirlemiştir. Bu önlem NFS algoritmasını etkisiz kılmasa dahi, algoritmanın çalışma süresini günümüz ve yakın gelecek teknolojisinde erişilmesi zor seviyelere yükseltebilir.

6.14 DROWN

DROWN (Decrypting RSA using Obsolete and Weakened eNcryption - zayıf ve geçmişte kalmış şifrelemelerle RSA şifresi çözmek) saldırısı, 2016 yılı ağustos ayında bir grup araştırmacı tarafından ortaya çıkarılan çapraz protokol saldırısıdır [6]. Çapraz protokol kavramı bu saldırının, SSLv2 protokolündeki açıklık ile TLS protokollerinin güvenliğinin etkilemesinden ötürü kullanılır.

DROWN saldırısı genel hali ile Bleichenbacher saldırısının (Bleichenbacher saldırısı için bkz. Bölüm 6.4) geliştirilmiş hali kullanılarak ve birtakım protokol açıklıkları kullanılarak gerçekleştirilen bir saldırıdır. 2048-bit uzunluğundaki bir RSA TLS şifreli metnini

çözebilmek için saldırganın, 1000 adet TLS el sıkışmasını ele geçirmesi, 40000 adet SSLv2 bağlantısı gerçeklemesi ve toplamda 2^{50} çevrimdışı işlem yapması gerekir.

6.14.1 Saldırının Ön Koşulları

DROWN saldırısını gerçekleştirilmek için bir implementasyona ihtiyaç duymamaktadır ve yalnızca eski RSA ithal şifre paketlerinin kullanılması yeterlidir. SSLv2 ve ithal şifre paketleri kullanılarak gerçekleştirilen bağlantılar ile sunucunun RSA gizli anahtarı ele geçirilir. Sunucunun (veya başka bir sunucunun) aynı gizli anahtarı hem SSLv2'de hem de TLS'de kullanması durumunda TLS bağlantılarına saldırı gerçekleştirilebilir.

6.14.2 Saldırı Senaryosu

Saldırıyı gerçekleştirebilmek için saldırganın, TLS'e ait $ClientKeyExchange$ mesajını SSLv2 ye ait $ClientKeyExchange$ mesajına çevirmesi gerekmektedir. Bu mesajın şifresinin çözülmesi halinde ön ana giz değeri elde eden saldırgan oturum anahtarını oluşturabilir. Saldırının genel adımları aşağıdaki gibidir:

1. Saldırgan pasif bir şekilde sunucunun gerçekleştirdiği TLS RSA anahtar anlaşma mesajlarını dinler ve kaydeder.
2. Saldırgan kayıt ettiği ve 48 bayt uzunluğunda ön ana giz barındıran bu mesajlardan birisini RSA PKCS#1 v1.5 şifreli mesajına çevirerek SSLv2 mesajını elde eder [6].
3. Saldırgan geçerli bir SSLv2 RSA mesajını elde ettikten sonra Bleichenbacher saldırısını uygulayarak şifreli mesajı çözebilir (Bleichenbacher saldırısı için bkz. Bölüm 6.4)
4. Son olarak saldırgan elde ettiği mesajı tekrar TLS versiyonuna ait açık metne dönüştürür. Bu metin 1. adımda elde edilen anahtar anlaşma mesajlarından birisinin açık halidir.

6.14.3 Saldırıya Karşı Alınan Önlemler

DROWN saldırısı sunuculara yönelik bir saldırıdır. Bu yüzden istemcilerin alması gereken bir önlem bulunmamaktadır. DROWN saldırısından korunabilmek için sunucuların SSLv2 kullanmaması ve mevcut TLS bağlantılarında kullanılan gizli anahtarların SSLv2 kullanan bir sunucuda veya yazılımlarında kullanılmadığından emin olması gerekmektedir. Eğer SSLv2'nin kullanılması halinde ise aşağıdaki güncellemeler ile kullanılması gerekmektedir.

- **OpenSSL:** OpenSSL 1.0.2 ve önceki versiyon kullanıcılarının 1.0.2g versiyonuna, OpenSSL 1.0.1 ve önceki versiyon kullanıcılarının da 1.0.1s versiyonuna güncelleme yapmaları gerekmektetir.
- **Windows Server:** Windows Vista, Windows Server 2008, Windows 7 ve Windows Server 2008R2 versiyonlarında SSLv2 desteği bulunmaktadır. Bu destek kullanılırken sunucuların 'subkey' adı verilen alt anahtar kullanımını devre dışı bırakmaları gerekmektedir.
- **NSS:** NSS 3.13 versiyonu itibari ile SSLv2 desteği bulunmamaktadır. Bu yüzden kullanıcılarının güncelleme yapması ve gizli anahtarlarının ifşasını kontrol etmeleri gerekmektedir.

6.15 SSL/TLS'e Yönelik Protokol Saldırılarının Özeti

SSL ve TLS protokollerine yönelik son 25 yılda pek çok saldırı gerçekleştirilmiştir. Bu saldırıların bazıları pratik ve kritik boyutlarda olurken, bazı saldırılar da teorik olmanın önüne geçememiştir. Bu çalışmada ele alınan protokol saldırıları, özet halinde Tablo 6.6'deki gibi incelenebilir.

Tablo 6.6: SSL/TLS'e Yönelik Protokol Saldırılarının Özeti

Tarihi	**Saldırının Adı**	**Etkisi**	**Kullandığı Açıklık**	**Alınan Önlem**
1996	Şifre paketi düşürme [97]	Güçlü algoritmayı zayıf algoritma ile değiştirir.	Şifre paketinin doğrulanmaması	NULL paketlerin kaldırılması
1996	Şifreli iletişime geçme mesajını düşürme [97]	*ChangeCipherSpec* mesajını düşürür.	*Finished* mesajının beklenmemesi	*ChangeCipherSpec* mesajından önce *Finished* mesajının gönderilmemesi
1996	MAC'in tuzlama uzunluğunu kapsamaması [97]	Şifreli metin uzunluğunu ortaya çıkarır	Tuzlamanın uzunluğunun MAC'e dahil edilmemesi	Tuzlamanın MAC'den önce yapılması
1998	Bleichenbacher [14]	Şifreli metni ortaya çıkarır.	PKCS#1 tuzlama formatı	PKCS#1 kütüphanesinin kullanılmaması
2003	Geliştirilmiş Bleichenbacher [58]	Şifreli metni ortaya çıkarır.	PKCS#1 tuzlama formatı	PKCS#1 kütüphanesinin kullanılmaması
2011	BEAST [31]	Çerezlerin açık metnini ele geçirir.	CBC kipinde IV tahmin edilebilmesi	Aşikar IV kullanımı
2012	CRIME [56]	Oturum anahtarını ele geçirir.	HTTP isteklerinin sıkıştırılmış boyutları	TLS sıkıştırma metodlarının kullanılmaması
2013	TIME [13]	Oturum anahtarını ele geçirir.	HTTP cevaplarının sıkıştırılmış boyutları	TLS sıkıştırma metodlarının kullanılmaması
2013	BREACH [43]	Oturum anahtarını ele geçirir.	HTTP cevaplarının sıkıştırılmış boyutları	TLS sıkıştırma metodlarının kullanılmaması
2013	Lucky 13 [5]	Şifreli mesajı sunucuya çözdürür.	Tuzlamanın MAC'e dahil edilmemesi	Uyarı mesajlarının süre farklılıklarının ortadan kaldırılması
2014	Poodle [76]	Çerezlerin açık metnini ele geçirir.	Tuzlama içeriğinin kontrol edilmemesi	SSLv3'ün kullanılmaması
2014	Heartbleed [32]	Kurbanın hafızasına (RAM) erişir.	*heartbeat* mesajında uzunluk bilgisi alınması	$payload - lenght$ uzunluğunun kontrol edilmesi
2016	FREAK [12]	Oturum anahtarını elde eder.	İthal RSA şifre paketlerinin kullanılması	İthal RSA şifre paketlerinin kullanılmaması
2015	Logjam [3]	Oturum anahtarını elde eder.	İthal DH şifre paketlerinin kullanılması	İthal DH şifre paketlerinin kullanılmaması
2016	DROWN [6]	Şifreli metni ortaya çıkarır.	SSLv2'de kullanılan gizli anahtarın TLS'de kullanılması	SSLv2'de kullanılan gizli anahtarın TLS'de kullanılmaması

Bölüm 7

Diğer Saldırılar

7.1 Rastgele Sayı Tahmini

1996 yılında Goldberg ve Wagner tarafından yayınlanan makalede [44], Netscape tarayıcısı ile yapılan SSL bağlantılarında kullanılan rastgele sayıların kalitesi incelendi. Yazarlar uygulamanın kaynak koduna erişim sağladılar ve rastgele sayı üreten algoritmanın açıklıklarını ortaya çıkardılar. Bu algoritmanın;

- Anlık saat bilgisi,
- Mevcut işlemin numarası (process id)
- İşlemi oluşturan ana işlemin numarası

gibi değerlere bağlı olduğu ortaya çıktı. Bununla birlikte ithal şifre paketlerinin de kullanılıyor olmasının (bkz. Bölüm 6.12.1) oluşacak anahtarın güvenliğini düşürdüğünü, böylece bağlantıların kaba kuvvet saldırılarına karşı dayanıksız hale geldiği dile getirildi.

Rastgele sayı üreteçlerinin sağlıklı çalışmasının yalnızca SSL/TLS protokollerine ait bir sorun olmamasına rağmen kurulan bağlantılarda üretilen sayıların rastgele olması için gerekli önlemlerin alınması gerekmektedir.

7.2 Debian OpenSSL Uygulaması Açıklığı

2008 yılında Luicano Bello tarafından yapılan kod incelemede, OpenSSL uygulamalarının bazı sürümlerde rastgele sayı üretecinin tahmin edilebilir olduğu ortaya çıktı [10]. Bu açıklık Debian sürümlerini kapsamaktaydı ve Eylül 2006'dan Mayıs 2008'e kadar olan

tüm sürümler için geçerliydi. Açıklık, geçersiz olduğu düşünülen iki satır kodun silinmesi ile meydana geldi.

```
MD Update(&m, buf , j ) ;
[ . . ]
MD Update(&m, buf , j ) ; /* purify complains*/
```

Şekil 7.1: OpenSSL Debian kaynak kodundan çıkarılan satırlar

Çıkarılan satırlar anahtar uzayının büyük oranda küçülmesine sebep oldu ve böylece kaba kuvvet saldırısına dayanıksız hale geldi.

Buradan çıkarılan sonuç ise yazılımcıların güvenlik ile ilgili kod parçalarını yazarken açıklamaları ihmal etmemeleri ve yazdıkları kodun ne anlama geldiğini tam olarak belirtmelerinin gerekliliğidir.

7.3 Uyarı Mesajları ile DOS Saldırısı

2009 yılında Zhao tarafından ortaya çıkarılan saldırıya göre, TLS el sıkışma senaryosunda DOS saldırısına izin veren bir açıklık bulunmaktadır [103]. Saldırının gerçekleşmesi aşağıdaki gibi iki farklı şekilde olabilmektedir:

- İlk tip saldırı TLS'in uyarı protokolünü hedef almaktadır. El sıkışma sürecinde henüz hangi kriptografik bileşenler ile şifreleme yapılacağı belirlenmemiş olan adımlar mevcuttur. Bu aşamada oluşan uyarı mesajları herhangi bir kimlik doğrulama içermemektedir ve bu yüzden kurban farklı mesajlarla kandırılabilir. Sahte bir hata mesajı ile bağlantının direkt olarak kapatılması sağlanabilir.
- İkinci tip saldırı ise, araya giren saldırganın taraflara aynı mesajı tekrar ederek veya yanıltıcı bir uyarı mesajı göndererek el sıkışma sürecinin manipüle etmesidir.

Buradan ortaya çıkan sonuç, DOS gibi basit bir saldırı dahi TLS protokolü için tehlike arz etmektedir ve protokol tasarlanırken temel saldırılar göz önünde bulundurulmalıdır.

7.4 SSL/TLS'in Üst Katmanda Engellenmesi

2009 yılı Şubat ayında Moxie Marlinspike tarafından 'sslstrip' adı verilen bir program yayınlandı [70]. Bu program temel olarak istemci tarafından yapılan HTTPS isteklerini

HTTP isteklerine çevirmektedir. Böylece sunucu ile istemci arasında SSL/TLS kurulmasını engelleyerek açık olan trafik üzerinden aktif veya pasif saldırı gerçekleştirebilir.

Bu saldırının gerçekleşebilmesi için 'sslstrip' programının istemcinin bilgisayarına yüklenmiş olması gerekmektedir. Bununla birlikte istemciler böyle bir saldırı ihtimalini göz önünde bulundurarak HTTPS ile girmek istedikleri bir siteden HTTP cevabı gelip gelmediğini tarayıcılarından kontrol etmelidirler.

Bölüm 8

SSL/TLS Analiz Aracı (Webtester)

8.1 Giriş

Günümüz iletişim dünyasının ihtiyaçlarını karşılamak üzere ilgili IETF çalışma grubu tarafından SSL/TLS protokolünün yeni sürümleri için çalışmalar yürütülmektedir. Bu bağlamda, 2018 Ağustos'unda TLS 1.3 versiyonu yayınlandı[80]. Öte yandan, bu protokolün açıklıkları her ne kadar yeni versiyonlarla kapatılmaya çalışılmakta olsa da, halihazırda kullanılmakta olan sistemlerin geriye dönük destek ihtiyaçları nedeniyle SSL/TLS eski sürümlerine destekleme zorunluluğu doğabilmektedir. Bu nedenle protokolün yeni versiyonlarıyla açıklıklarının kapatılmaya çalışılmasınının yanı sıra, güncel sistemlerin protokol ayarlarının doğru yapılıp eski versiyonlarla kurulan bağlantıların da güvenlik zaafiyetine neden olmayacağından emin olmak gerekir. Bunu sağlayabilmek için de özellikle sunucu tarafında mevcut saldırılara karşı dayanıklılık testlerinin yapılması kaçınılmazdır.

Örneğin farklı sunucu ve istemcilerden oluşan açık veya kapalı bir sistem olduğunu varsayalım. Bu sistem içerisinde 2006 yılından önce üretilmiş bir cihaz ve yazılım olarak mevcut zamandaki son SSL/TLS versiyonu olan TLSv1.0'ı destekliyor olsun. Bu cihaz donanımsal yetersizlikler, 7/24 çalışma zorunluluğu, gömülü sistem olması gibi pek çok nedenden ötürü yeni versiyonları destekleyemeyebilir. Güncel SSL/TLS versiyonlarını kullanamayan cihazların diğer tüm cihazlarla kuracağı bağlantılar risk altındadır. Bu desteklerin verilmesi, sistemleri olası versiyon düşürme saldırılarına zayıf bırabilecektir. Sonuç olarak bir sistem kontrol edilirken geçmiş tüm versiyonlar ve açıklıklar kontrol edilmeli ve olası tehditlere karşı önlemler alınmalıdır.

Önceki bölümlerde detaylı olarak incelenen SSL/TLS saldırıları göstermektedir ki; web güvenliğinin sağlanması bir çok parametre ve koşula bağlıdır. Güncel protokollerle birtakım önlemler alınmış olsa bile (Örneğin; BEAST [31], [76] Logjam [3] vb.), protokolün gerçeklenmesi aşamasında da saldırı tehlikeleri (Örneğin; Heartbleed [32], Debian

OpenSSL [10] vb.) bulunmaktadır. Güvenli iletişim için ihtiyaç duyulan bu gereksinimleri karşılamak adına SSL/TLS uygulamalarının denetlenmesi ve eksik noktaların belirlenmesi için analiz araçlarına gereksinim duyulmaktadır. Bu araçlar/programlar SSL/TLS protokolünün farklı versiyonlarını, bu versiyonlarda izin verilen ya da desteklenen parametreleri, elektronik sertifikaları ve bilinen saldırılara dayanıklılığı öçlmek adına kontrol ve testler gerçekleştirerek; daha önce bahsedilen saldırılara karşı sistem dayanıklılığını ölçümlemektedirler.

SSL/TLS ile haberleşme gerçekleştirilirken sunucu ve istemci olarak iki taraf bulunmaktadır [25]. İstemci tarafında alınabilecek temel önlem, güncel istemci programları kullanmak ve güvenli sertifikası olmayan sunuculara karşı dikkatli olmaktır. Güvenli sertifikası olmayan sunucular ile iletişim sağlandığında gizlilik, bütünlük ve erişilebilirlik kıstaslarının karşılanamayabileceğinin bilincinde olunmalıdır. Sunucu tarafında ise daha detaylı ve dikkatli önlemler alınmalıdır. Çünkü bir sunucu birbirinden farklı yazılımlara ve donanımlara sahip çok sayıda istemcinin ihtiyacını karşılamaya çalışmaktadır. Bununla birlikte kendini güncelleyemeyen ve güvenli şifre paketleri kullanamayan istemcileri de destekleyebilmek için güvenlik açığı bulunan protokoller kullanmak zorunda kalabilir. Böyle durumlarda protokol versiyonunun güncellenmesi ile alınamayan önlemlerin şifre paketleri düzenlemesi ya da uyarı protokolünde el ile değişiklik yapılması gibi önlemlerle önüne geçilmesi gerekir. Sunucu tarafının güvenliğinin sağlanmasının istemci tarafında göre daha önemi ve kritik olmasından ötürü, analiz araçlarında genellikle sunucular test edilir.

8.2 Analiz Araçlarının Özellikleri

Analiz araçları çalışma yöntemleri temel olarak hedef sunucu ile farklı varyasyonlarda SSL/TLS bağlantıları kurmaya çalışarak sunucunun desteklediği algoritmaları ve protokolleri ortaya çıkarmaktadır. Sunucu, istemcinin kendisini doğrulayabilmesi için, sertifikasını gönderir. Bu sertifika bilgisinden de analiz açısından değerli bilgiler elde edilebilmektedir (Sertifika içeriği için bkz. Bölüm:2.1.1.1). Sertifika bilgileri web üzerindeki sunucular için kimlik doğrulama özelliği açısından hayati öneme sahiptir. Kapalı ve iç ağlarda ise sunucuların sertifika otoritelerinden alınmış sertifikaları olmayabilir. Sunucu sertifikaları ilgili kuruluş içerisindeki bir sertifika otoritesinden alınmış ve kendisi tarafından imzalanmış bir sertifika dahi olabilir. Bunun nedeni olarak kapalı ağlarda sunucuların sertifikalarının ön yükleme ile devreye alınıp, istemcilere bu sunucuların adres ve sertifikalarının baştan kaydedilmesi ve OSCP protokolünün web üzerinde çalışıyor olması gösterilebilir.

Analiz araçları çalışma türleri açısından offline(çevrimdışı) ve online(çevrimiçi) olarak iki farklı türde hizmet verebilmektedir.

- **Online Analiz Araçları:** Internet üzerinde çalışan, genellikle kaynak kodu erişimi olmayan, web arayüzü üzerinden kullanılan uygulamalardır. Analiz edilmek istenen sunucunun IP adresi programa verilir ve arka planda çalışıp sonucu kullanıcıya iletir. Bu tarz programların iki ciddi dezavantajı bulunmaktadır. Birincisi; kaynak kodları bilinmedikleri için analiz verilerinin arka planda nasıl kullanıldığı veya üçüncü partilere satılıp satılmadığı sorularını ortaya çıkarmalarıdır. İkincisi ise; yalnızca Internet üzerinde çalışmasından ötürü kapalı ağlarda veya şirket ağlarında kullanılamamasıdır.

  Online analiz araçları genellikle web güvenliği sektöründeki firmalar tarafından ücretsiz hizmet olarak sunulur. Symantec firmasının 'SSL Checker'[92] ve Qualys firmasının 'SSL Server Test'[79] araçları bu türe örnek gösterilebilir.

- **Offline Analiz Araçları:** Offline araçlar online araçlardan farklı olarak tüm ağlarda kullanılabilir. Bu özelliği sağlayan farklılık kaynak kodunun veya çalıştırılabilir betiğin kapalı ağlara taşınıp orada çalıştırılabilmeleridir. Bu tarz araçlar genellikle açık kaynak kodlu araçlardır. Kaynak kodu bilinmeyen bir analiz programının kapalı bir ağda çalıştırılması ciddi tehlikeler barındırabilir ve kesinlikle tercih edilmemelidir. Kaynak kodu paylaşılan araçların fonksiyonları incelenebilir ve ihtiyaca göre kullanılabilir ya da düzenlenebilir.

  Offline analiz araçları kapsamlı tarama aracı şeklinde olabileceği gibi, spesifik bir amaca yönelik geliştirilmiş de olabilir. Bu tarz araçlara örnek olarak 'sslscan' [51] ve 'TestSSL' [30] verilebilir.

8.3 Webtester Uygulaması ve Gerekliliği

SSL/TLS protokolünün kullanım senaryolarının geniş bir alanı kapsamasından ötürü güvenlik analizi konusunda yapılan çalışmalar da çeşitlilik göstermektedir. Örneğin; 'SSLINT' aracı, sadece TLS sertifika validasyonu ile ilgili açıklıkları incelerken [47], 'SMVHUNTER' aracı yalnızca Android tabanlı uygulamalarda SSL/TLS açıklıklarını tespit etmektedir [46]. Buna karşın, kapalı ağlara yönelik ve ağ üzerinde tarama yapabilen bir uygulama bulunmamaktadır. Offline analiz aracı olup, bu beklentiye en yakın olan iki araç 'sslscan' ve 'TestSSL' araçlarıdır. Bu iki uygulama da kapalı bir ağda bulunan sunucuların SSL/TLS açıklıklarını tarayıp raporlayabilmektedir. Ancak; hem kontrol edilen maddelerin çokluğu, hem de yalnızca verilen sunucu IP'lerinin kontrol edilmesinden ötürü bir ağ üzerinde SSL/TLS taraması yapılması durumunda yeterli olamamaktadırlar.

Yukarıda bahsedilen kısıtlamaları aşmak adına, bu çalışma kapsamında kapalı ağlardaki sunucuların hem tespit edilip hem de taranmasını sağlayabilecek bir analiz aracı 'Webtester' geliştirilmiştir. Diğer uygulamalardan farklı olarak Webtester uygulaması kapalı

ağların ihtiyaçlarına göre özelleştirilmiş bir araçtır. Webtester aracı geliştirilirken, aşağıdaki argümanlar göz önünde bulundurulmuştur.

1. Suncularını kendi bünyelerinda kapalı ağda bulunduran kuruluşlar, sistemin kapalı olmasına güvenerek ve maliyeti gözeterek güvenilir sertifika otoritelerinden onaylı sertifika almayı gerekli görmeyebilirler. Bu yüzden sertifika zinciri, sertifika şeffaflığı, çevrimiçi sertifika durumu protokolü (OSCP) gibi özelliklerin kontrolüne ihtiyaç duyulmamaktadır.

2. TLS uzantıları kontrolleri ile sağlanan, sunucu adı, durum isteği, istemci sertifikası bilgisi gibi özellikler Internet üzerinde kullanışlı olmasına rağmen kapalı ortamlarda ihtiyaç duyulmayan özelliklerdir.

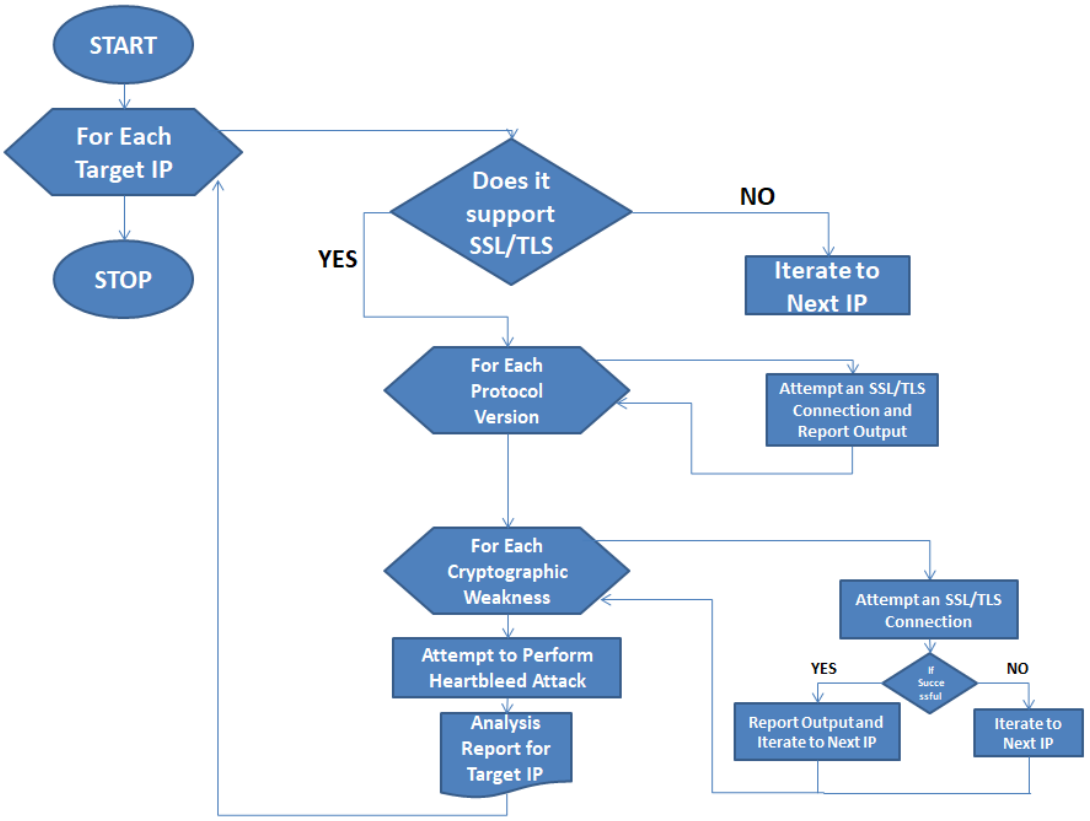
3. Kapalı ağlarda analiz süreci öncesinde bilinen sunucuların yanı sıra, zaafiyet gösterebilecek tüm cihazların (özellikle 443 nolu portun açık olma durumu) taranması ve tespit edilmesi gerekmektedir. Çünkü temel amaç tek bir sunucuyu değil, mevcut olan tüm sunucuları hızlı bir şekilde tespit edip sırayla kontrol etmektir.

8.4 Analiz Araçlarında Performans Arttırma Önerileri

Analiz araçları karşılaştırılırken performans ölçütü belirleyici bir kriter olmaktadır. Özellikle de geniş çaplı taramalarda performans farkı önemli bir tercih sebebidir. Bu bölümde analiz araçlarında performans artışı için sunduğumuz ve Webtester uygulamasında kullandığımız yaklaşımlar anlatılacaktır. Bununla birlikte Bölüm 8.4.2'da performans artışı sağlanması için çıkarılmasını tavsiye ettiğimiz maddeler bulunmaktadır.

Webtester uygulaması, hem Internet üzerinde hem de kapalı ağlarda SSL/TLS protokolünün güvenliğinin test edilmesi gereken durumlarda hızlı ve faydalı bir alternatif olması amacıyla geliştirilmiştir. Açık kaynak olmasından ötürü uygulamayı kullanan insanin kendi sunucularına ait bilgilerin ele geçirilmesinden endişe edilmeksizin kullanılabilir ve ihtiyaca göre modifiye edilebilir. Benzer şekilde kapalı ve açık ağlarda çalışabilen uygulamalar olan 'sslscan' ve 'TestSSL' uygulamalarına göre birtakım farklılıklar içermektedir. Bu farklılıklardan ötürü daha hızlı analiz yapılması mümkün olmaktadır. Hız artışının sağlanması, birtakım algoritmik yaklaşımlarla ve 'sslscan' ve 'TestSSL' gibi uygulamalarda olup da kapalı ağlarda güvenlik taramasında pratik etki sağlamayacağı düşünülen özelliklerin çıkarılması ile sağlanmıştır. Şekil 8.1'de Webtester uygulamasının algoritmasının akış diyagramı yer almaktadır:

Uygulama temel hatları ile aşağıdaki adımları gerçekleştirir:



ŞEKIL 8.1: Webtester Uygulaması Akış Şeması

1. Girdi olarak verilen bir yada birden fazla IP adresi için SSL/TLS desteği olup olmadığının anlaşılması için, 443 portu üzerinden bağlantı isteği gerçekleştirilir. Başarılı olan IP'lerle sonraki aşamaya geçilir.

2. Sonrasında olarak hedef sunucunun SSLv2, SSLv3, TLSv1.0, TLSv1.1 ve TLSv1.2 protokollerini destekleyip desteklemediği kontrol edilir. Bu aşamada sağlıklı bir cevap alabilmek için ilgili protokolün desteklediği tüm şifre paketleri *Client_Hello* paketine eklenir.

3. Uygulamanın üçüncü aşamasında güvensiz şifre takımları aracılığı ile saldırılar test edilir. Örneğin, sunucunun anonim şifre paketlerini destekleyip desteklemediğini anlamak için tüm şifre paketleri teklif edilir. Sunucu bu paketlerden birisini seçip cevap olarak dönmesi halinde çıktı olarak uyarı verilir ve sonraki güvensiz algoritmaya geçilir.

4. Dördüncü aşamada Heartbleed saldırısı gerçekleştirilmeye çalışılır. Diğer saldırılardan farklı olarak bu saldırının mevcudiyetinin farkedilmesinin en pratik yolu saldırıyı test etmektir.

5. Uygulamanın son aşamasında elde edilen bilgiler çıktı olarak verilir ve tarama süresi paylaşılır. Eğer farklı bir IP varsa uygulama birinci aşamadan tekrar başlar.

Algoritmanın matematiksel hesaplaması yapılırken aşağıdaki faktörler belirleyici olmuştur:

- Uygulamanın iki, üç ve dördüncü aşamalarında toplamda 16 adet SSL/TLS bağlantısı gerçekleştirilmektedir. İkinci ve üçüncü aşamalardaki birtakım çıktılar sonraki aşamalarda yapılacak SSL/TLS bağlantı sayısını azaltabilir. Bu seçenekler aşağıdaki gibidir:
  - İkinci aşamada sunucunun SSLv2, SSLv3 ve TLSv1.0 versiyonlarının desteklememesi durumunda sunucunun RC2 ve RC4 algoritmalarını desteklemesi mümkün değildir. Çünkü TLSv1.1 itibari ile bu algoritmalar desteklenmemektedir [93]. Bu yüzden RC2 ve RC4 algoritmalarının tek seferde denendiği SSL/TLS bağlantısı yapılmaz.
  - İthal şifre paketleri yalnızca SSLv2 ve SSLv3 versiyonlarında bulunur. Bu yüzden SSLv2 ve SSLv3'ü desteklemeyen sunucularda ithal şifre paketi desteği denenmesine gerek yoktur.
  - BEAST saldırısına karşı dayanıksız sunucular SSLv3 veya TLSv1.0 destekliyor olmalıdır. Bu iki versiyonu desteklemeyen sunucuların AES-CBC kipini desteklemeleri saldırı için yeterli değildir. Bundan ötürü SSL3 ve TLSv1.0 protokollerinden ikisini de desteklemeyen sunucular için AES-CBC kipi desteği aranmaz.
  - İleriye yönelik gizlilik özelliğinin sağlanamaması için RSA algoritması desteği veya DH/ECDH algoritma desteğinden birisinin sağlanması yeterlidir. İki algoritma da test edilirken ileriye yönelik gizlilik şartı için taranmaktadır. Bu yüzden iki yöntemden birisinde özelliğin sağlanamaması halinde diğerinin test edilmesine gerek yoktur.

  Yukarıda bahsedilen 4 özellikten her biri, uygulamada gerçekleştirilmek istenen 16 SSL/TLS bağlantısından bir tanesinin yapılmasını gereksiz kılmakta ve performans artışı sağlamaktadır. Bu getirinin sonucu olarak en iyi durumda 12, en kötü durumda 16 ve ortalama olarak 15 SSL/TLS bağlantısı gerçekleştirilmektedir.

- Uygulamada paralel bağlantılar kurulmayıp, işlemler sıralı bir biçimde ilerlemektedir. Her bir aşamada işlem yükü açısından yapılan işlemler aynı süre içerisinde gerçekleştirilmektedir. Her aşamada yapılan işlem soket açıp kapama ve $Client\_Hello$ paketinden $Finished$ paketine kadar yapılan ağ bağlantılarıdır. İşlem yükü hesaplanırken bu işlemler ile yapılan SSL/TLS bağlantıları çarpılır.

- Yapılan testlere göre ve Tablo 8.2'den elde edilen verilere göre soket açmak ortalama $0,001$ saniye, SSL/TLS bağlantısı kurmak ortalama $0,145$ saniye sürmektedir.

Yapılan testlere göre ve Tablo 8.2'den elde edilen verilere göre yapılan işlemlerin maliyeti aşağıdaki gibi hesaplanabilir:

$$\text{İşlem Maliyeti} = \text{Gerçekleşen SSL/TLS bağlantı adedi} * \text{SSL/TLS başına maliyet}$$

$$\text{SSL/TLS başına maliyet} = 0,145sn + 0,001sn = 0,146sn$$

Bu değerlere göre Tablo 8.1'deki gibi bir sonuç elde edilmiştir.

Tablo 8.1: Zaman Hesaplama Tablosu

	Yapılan SSL/TLS Adedi	**Bağlantı Başına Süre**	**Toplam Süre**
En İyi Senaryo	12	0,146sn	1,752sn
En Kötü Senaryo	16	0,146sn	2,336sn
Ortalama Senaryo	**15**	**0,146sn**	**2,19sn**

8.4.1 Webtester Uygulamasında Uygulanan Pratik Yaklaşımlar

Webtester uygulaması geliştirilirken temel motivasyon olan performans artışını sağlayabilmek için mevcut uygulamalara göre birtakım farklılıklar geliştirilmiştir. Bu farklılıkların bir kısmı hedef sunucunun analizinde yapılan protokol testleri, şifreleme algoritmaları tespiti gibi aşamalarda ağ trafiğini azaltmaya yöneliktir. Bununla birlikte sunucunun analizi için kritik öneme sahip olmayan bazı detayların kontrol edilmemesi ile de performans artışı sağlanmıştır. Aşağıdaki listede Webtester uygulamasının performans artışı için sahip olduğu yaklaşımlar yer almaktadır:

- SSL/TLS protokolünde şifre paketinin belirlenmesi için istemci tarafından gönderilen *Client_Hello* paketinin içinde desteklenen şifre paketleri bulunmaktadır (bkz. Bölüm 4.3.1.2). Her bir şifre paketi 2 bayt uzunluğunda bir veri boyutundadır. Bu yüzden teorik olarak 65536 farklı şifre paketi olabilir. Ancak pratikte SSL/TLS implementasyonları tarafından desteklenen toplam şifre paketi sayısı protokollerin kabul gördüğü şifrelerin kombinasyonları ile sınırlıdır[25]. Bu kombinasyonlar toplamda 328 adettir. Webtester uygulaması hedef sunucu ile kurduğu ilk SSL/TLS bağlantısında hem SSLv3'den TLSv1.2 ye kadar tüm protokolleri destekleyip, hem de tüm şifre paketlerini sunmaktadır. "TestSSL" ve "sslyze" uygulamaları ise her bir protokolü kendi destekledikleri şifre paketleri ile toplamda 5 bağlantı ile tespit etmektedirler. Bu yaklaşımla edinilen kazanım daha az anahtar anlaşması yaparak performans artırımıdır. Dezavantajı ise tek bir SSL/TLS bağlantısında hem SSLv2,

hem de diğer protokolleri test etmek mümkün olmadığından, sunucunun yalnızca SSLv2 versiyonunu desteklediği durumlarda Webtester uygulaması başarısız olup, SSL/TLS bağlantısı kuramadığını söylemektedir. Qualys firmasının Ekim 2018 raporuna göre (Qualys firmasının raporu için link: *www.ssllabs.com/ssl−pulse*) web üzerinde sunucuların SSLv2 desteği oranı yalnızca %2.3 seviyesindedir. Bu sitelerin bazılarının diğer protokolleri de destekledikleri de göz önünde bulundurulduğunda bu değer daha da düşmektedir. Webtester uygulaması %2.3'den daha düşük bir oranda hata payına karşılık performans artışı sağlamıştır.

- Sunucular için ileriye yönelik gizlilik sağlanması kritik öneme sahiptir(İleriye yönelik gizlilik için bkz Bölüm:2.3.1). Bu yüzden Webtester uygulamasında da diğer uygulamalarda olduğu gibi anahtar anlaşmalarında ileriye yönelik gizlilik kontrol edilmektedir. Bu adımı gerçekleştirmek için "sslyze" ve "TestSSL" uygulamaları ileriye yönelik gizlilik sağlayan şifre paketleri ile SSL/TLS bağlantısı kurmaya çalışırlar ve bağlantının gerçekleşip gerçekleşmemesine göre sonucu elde ederler. Webtester uygulamasında hız kazanmak için bu yaklaşım tercih edilmemiştir. Bunun yerine RSA ve ECDH gibi ileriye yönelik gizlilik sağlamayan anahtar anlaşma protokollerinin desteklendiğinin farkedilmesi durumunda uyarı verilerek fazladan SSL/TLS bağlantısı yapılmasının önüne geçilmiştir.

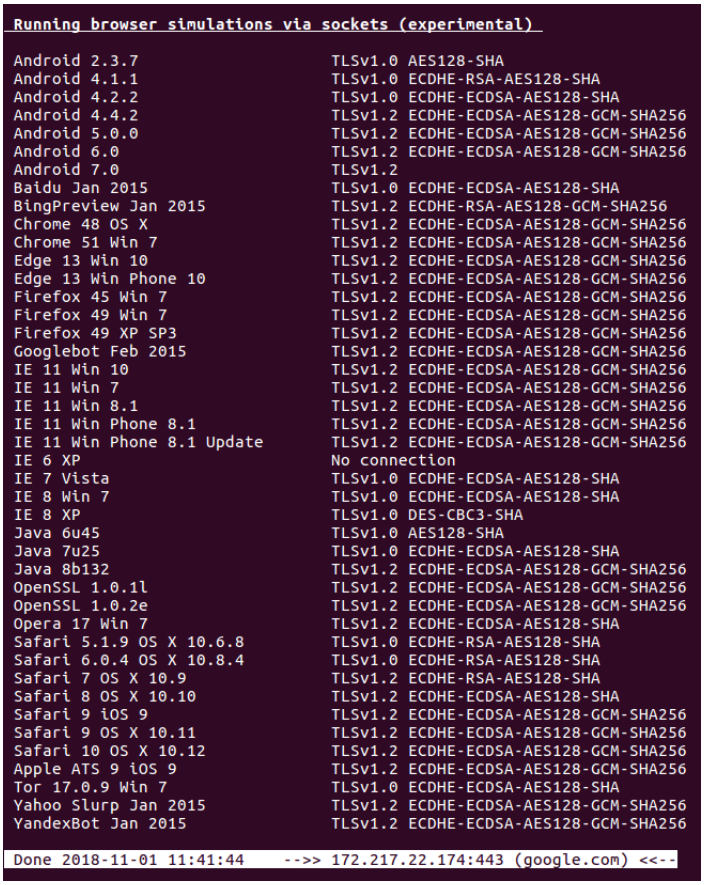
- Webtester uygulamasının performans artışı için tercih ettiği bir başka yaklaşım da, desteklenen şifre paketlerini teker teker tespit etmek yerine yalnızca desteklenen algoritmaları ortaya çıkarmaktır. Uygulama çıktısı olarak "sslyze" ve "TestSSL" uygulamalarında desteklenen şifre paketleri verilir. Bu bilginin sağlanabilmesi için her bir şifre paketi ile el sıkışma senaryosu gerçekleştirilmesi gerekir. "sslyze" uygulaması OpenSSL kütüphanesi tarafından desteklenen 159 şifre paketini test etmektedir (OpenSSL şifre paketleri için link: *www.openssl.org/docs/man*1.0.2/*apps/ciphers.html*). "TestSSL" uygulaması ise 283 şifre paketi ile deneme yapmaktadır. Buna karşın Webtester uygulaması:
  - RSA içeren 52 şifre paketi
  - DH/ECDH içeren 101 şifre paketi
  - Export özelliğine sahip 15 şifre paketi
  - MD5 içeren 13 şifre paketi
  - NULL parametresi içeren 21 şifre paketi
  - DES/DES içeren 36 şifre paketi
  - AES CBC kipi içeren 64 şifre paketi
  - Anonymous parametresi içeren 27 şifre paketi
  - SHA1 içeren 112 şifre paketi

– RC2/RC4 içeren 21 şifre paketi

içeren birer SSL/TLS bağlantısı ile ilgili algoritmanın veya özelliğin barındırılıp barındırılmadığını tespit edebilmektedir. Böylece desteklenen algoritmalar ve özellikler toplamda 10 SSL/TLS bağlantısı ile anlaşılmaktadır. Bu işlemler yapılırken hedef algoritmaya göre birtakım şifre paketleri birden fazla gönderilmiştir. Örneğin $TLS\_ECDHE\_RSA\_WITH\_NULL\_SHA$ şifre paketi hem RSA, hem NULL hem de SHA1 içerdiği için 3 kez $Client\_Hello$ paketlerinde yer almıştır.

8.4.2 Webtester Uygulamasında Yer Almayan Özellikler

Bununla birlikte, Webtester uygulamasında performans artışı sağlamak için vazgeçilen özellikler de bulunmaktadır. Bu özelliklerin bazıları kapalı ağlarda ihtiyaç duyulmayan özellikler olması sebebiyle, bazıları da sağladıkları bilgiye oranla performans kaybına değer bulunmayan özelliklerdir. Genel hatlarıyla bu farklılıklar aşağıdaki gibidir:



Şekil 8.2: TestSSL için farklı platformlarda SSL/TLS Testi

- SSL/TLS analiz uygulamalarında uygulama yükünün büyük kısmını farklı platformlarda test yapılması almaktadır. Örneğin "TestSSL" programı hedef sunucunun farklı linux dağıtımları, 32 ve 64 bit Windows versiyonları ve Andorid iOS gibi

mobil ortamlardaki tarayıcılarla el sıkışma senaryosu gerçekleştirmektedir. Bunu başarmak için her bir ortama ait özelliklere göre SSL/TLS bağlantısı kurmaya çalışır. "TestSSL" uygulaması şekil 8.2'de görüldüğü gibi bir çok platforma ait el sıkışması senaryosu gerçekleştirmiştir. Benzer şekilde "sslyze" uygulaması da farklı platform ve istemci seçenekleri ile test yapmaktadır. Bu testlerin desteklenen platformların ve ilgili platformlar için sağlanan güvenlik seviyelerinin anlaşılması açısından büyük katkısı vardır. Ancak bu özellik; performans kaybı ve Webtester uygulamasının hedefi olan hız ve kolaylık sağlama özellikleri göz önüne alındığında terich edilmemiştir.

- Webtester uygulamasında kontrol edilmeyen bir diğer özellik ise elektronik sertifikalara ait işlemlerdir. Bu işlemler sunucunun kimliğinin doğrulanması ve sertifika zincirinin kontrol edilmesi gibi önemli detaylar içerir. Sunucudan desteklediği kimlik doğrulama yöntemlerine göre (Örneğin ECDSA, DSA, RSA vb.) sertifika bilgilerini göndermesi istenir. Bu bilgiler kontrol edilerek sunucunun kimliği doğrulanır. Sonrasında da sertifikanın temin edildiği otorite ile ilişki kontrol edilir. İstemcinin kimlik doğrulama işlemini tamamlayabilmesi için gönderilen sertifikanın kendi güvendiği bir sertifika otoritesi tarafından imzalanmış olması gerekmektedir. Bu güveni sağlamak, sertifika otoritesinin güvendiği ara otoriteyle de mümkündür (Sertifikalar için bkz. 2.1.1). Arz ettiği öneme karşın sertifika kontrolü aşamasının performans açısından birtakım kayıpları vardır. Bunlar aşağıdaki gibi sıralanabilir:
  - Sertifikanın geçerli olabilmesi için sertifika geçerlilik tarihinin, sertifikanın izin yetkisi olan alan adının, ilgili imzalama algoritmasına ait bilgilerin eklenip eklenmediği gibi bilgilerin kontrol edilmesi gerekir.
  - Sertifika özelliklerinin şifre paketinde tercih edilen değerlerle aynı anahtar boyu ve algoritmada olup olmadığının kontrol edilmesi gerekir.
  - Sertifika kontrolü yapılmak için sunucunun desteklediği tüm kimlik doğrulama yöntemleri ile sertifika göndermesi gerekir. Eğer sunucu 3 farklı kimlik doğrulama yöntemi destekliyorsa ilk SSL/TLS bağlantısı haricinde fazladan 2 SSL/TLS bağlantısı yapılması gerekir.
  - Sertifika elde edildiğinde CRL ve OCSP kontrolleri yapılır (CRL için bkz. Bölüm 2.1.5.1, OCSP için bkz. Bölüm 2.1.5.2). Bu kontroller hem bilgisayarın işlem yükünü artırır hem de ağ paketleri alışverişi yapılacağından ötürü paket dönüşleri beklenir.
  - Sertifikayı imzalayan otoritenin de imzası kontrol edilir ve kök sertifikaya gidene kadar aradaki sertifikaların yetkileri ve imzaları kontrol edilir.

Yukarıda sebep olan performans kayıplarından arınmak amacıyla sertifika kontrolleri Webtester uygulamasında yer almamaktadır. Sertifika kontrollerinin yapılmamasındaki bir diğer motivasyon da; kapalı ağlarda sunucuların parayla alınmış ve Internet üzerinden kontrolü sağlanan sertifikaları tercih etmemeleridir. Kapalı ağlarda sunucular ya kendi imzaladıkları sertifikalarla (self signed sertificate) ya da kapalı ağda bulunan HSM benzeri bir cihazdan elde edilen bir sertifika kullanmalarıdır. Webtester uygulamasının hem hızlı bir alternatif olmak istemesi hem de çoğunlukla kapalı ağlarda kullanılmak için tasarlanmış olması için sertifikalar kontrol edilmemektedir.

Not 8.1. Webtester uygulaması yukarıda ifade edilen tüm bu farklı yaklaşımlardan ötürü daha az SSL/TLS bağlantısı yapmaktadır. Standart bir hedef sunucu için;

- TestSSL uygulaması 289,
- sslyze uygulaması 165,
- Webtester uygulaması 16

SSL/TLS bağlantısı yapmaktadır. Tüm uygulamalarda en maliyetli kısımın SSL/TLS bağlantısı olmasından ötürü performans farkı bu şekilde ölçülebilir.

8.5 Webtester Uygulamasında Kontrol Edilen Maddeler

Güvenli bir SSL/TLS protokolü konfigürasyonu için dikkat edilmesi gereken pek çok husus bulunmaktadır. Genel olarak bu hususlar aşağıda incelenmiş olup, geliştirdiğimiz uygulamamızda da kontrol edilen maddeler olmuşlardır.

- **Gizli Anahtar Kullanmak**

  Gizli anahtarlar oluşturulurken 80-bit ve üzeri simetrik algoritmalara denk güvenlik sağlayan algoritmalar yeterli kabul edilir [41]. Bu nedenle,

  – RSA için 2048-bit ve üzeri uzunluk yeterlidir.
  – ECDSA için 224-bit ve üzeri uzunluk yeterlidir.

  Denk güvenlik seviyesinde ECDSA algoritması RSA'ya kıyasla çok daha hızlıdır [81]. Ancak, RSA algoritmaları günümüzde daha yaygın kullanıldığı için her iki seçenek için birer anahtar da oluşturulabilir. Bunun yanında, DSA algoritması 1024-bit üzerindeki anahtar boyları büyük oranda desteklenmediği için yeterli güvenlik sağlayamaz ve kullanılmamalıdır [17].

- **Güvenli Protokollerin Kullanılması**

  Eylül 2018 itibariyle TLSv1.3'ün henüz gerçeklenmemiş olması nedeniyle mevcut olarak 5 protokol (SSLv2, SSLv3, TLSv1.0, TLSv1.1, ve TLSv1.2) bulunmaktadır.

  – SSLv2 protokolü kritik açıklıklar bulundurmaktadır (örneğin Bleichenbacher saldırısı [14]). Bu yüzden SSLv2 desteklenmemelidir.

  – SSLv3 protokolünde kullanılan simetrik şifreleme yöntemlerinden hem RC4, hem de CBC kipi günümüzde güvensizdir [31]. Bu yüzden SSLv3 desteklenmemelidir.

  – TLSv1.0 protokolü BEAST saldırısına karşı önlemler alınmış olarak kullanılmakta olsa bile tavsiye edilmemektedir.

  – TLSv1.1 ve TLSv1.2 ciddi güvenlik açıklıkları bulundurmayan ve güvenli protokollerdir. Bununla birlikte DROWN saldırısından korunabilmek için SSLv2 ile kullanılmakta olan sertifikalar TLS versiyonlarında kullanılmamalıdır.

  Uygulamamızın ilk aşamasında hedef sunucunun desteklediği protokoller test edilmektedir. Desteklenen protokollerin belirlenmesi, açıklıkların anlaşılabilmesi için büyük öneme sahiptir. Pek çok saldırı protokol güncellenmesi ile etkisiz hale geldiği için ve bazı saldırıların yalnızca belirli protokol sürümleri ile çalışmasından ötürü (örneğin BEAST saldırısı CBC kipi kullanılsa dahi TLSv1.1 de gerçekleştirilemez.) öncelikle protokol desteği kontrol edilmelidir.

- **Güvensiz Şifre Takımları**

  TLS ile kurulan iletişimin güvenliği, seçilen şifre takımı ile belirlenir. Aşağıdaki formatta olan şifre takımları güvensizdir ve kullanılmamalıdır:

  – ???_**RSA_EXPORT**_???_WITH_???_???

  – ???_???_**NULL**_WITH_???_???

  – ???_???_???_WITH_**RC2**_???

  – ???_???_???_WITH_**RC4**_???

  – ???_???_???_WITH_**DES**_???

  – ???_???_???_WITH_???_**MD5**

  – ???_???_???_WITH_???_**SHA1**

  Uygulamamızda sunucunun yukarıda listelenen şifre paketlerini destekleyip desteklemediği test edilmektedir. Her bir madde için mevcut olan şifre paketleri *ClientHello* paketinde eklenerek SSL/TLS bağlantısı kurulmak istenmiştir. Gönderilen pakette sadece ilgili algoritmaya ait şifre paketleri eklendiği için (örneğin SHA1 desteği

kontrol edilirken sadece SHA1 içeren şifre paketleri gönderilir) az sayıda bağlantı ile hızlı bir şekilde sunucunun kurulum özellikleri anlaşılabilir.

- **İleriye Yönelik Gizlilik Sağlamayan Şifre Takımları**

  İleriye yönelik gizlilik, gizli anahtar ifşası gibi durumlarda kurulan bağlantının güvenliğinin sağlanabilmesi için gereklidir. Aşağıdaki formatta olan şifre takımları ileriye yönelik gizlilik sağlayamadıkları için kullanılmamalıdır:

  – ???_**RSA**_???_WITH_???_???

  – ???_**DH**_???_WITH_???_???

  – ???_**ECDH**_???_WITH_???_???

8.6 Webtester Uygulamasının Performansı

Önceki bölümlerde de bahsedildiği üzere Webtester uygulaması benzerleri ile kıyaslandığında detaylı analiz ve düşük hata payı gibi özelliklerden taviz vererek, karşılığında yüksek performanslı ve pratik bir yaklaşım öne sürmüştür. Bu durumdan ötürü performans artışı karşısında tespit kabiliyetinde azalmaya sebep verecek bir takas yapılmıştır. Ancak uygulama geliştirme aşamasında göz önünde bulundurulan detaylar ve pratik yaklaşımlar sayesinde Webtester uygulaması değerli bir alternatif olarak kendisini tanıtmaktadır. Bu bölümde Webester uygulamasının kapalı ve açık ağlarda "sslyze" ve "TestSSL" uygulamaları ile performans testleri yapılmıştır. Buna ek olarak açık ağlarda başarı oranı testi yapılmıştır.

8.6.1 Açık ve Kapalı Ağlarda Performans Testleri

Webtester uygulamasının diğer uygulamalara göre performans farkını test edebilmek için birtakım testler yapılmıştır. İlk olarak açık ağda yurt içinde en çok erişim sağlanan ve HTTPS protokolünü destekleyen siteler için uygulamalar çalıştırılmış ve Şekil 8.2'daki sonuçlar elde edilmiştir.

Bununla birlikte, kapalı ağlarda sanal sunucular oluşturulmuş ve bu sunuculara farklı SSL/TLS konfigürasyonları yapılmıştır. Sunucular "Ubuntu 14.4 Server" işletim sistemi üzerinde "Apache 2.4" ve "OpenSSL 1.1.1" versiyonları kurulması ile oluşturulmuştur. Sunucularda yer alan SSL/TLS konfigürasyon farklılıkları tablo 8.3'daki gibidir. Bu konfigürasyonlara göre uygulamaların çalışma süreleri tablo 8.4'da yer almaktadır.

Tablo 8.2: Online Çalışma Süresi Karşılaştırması

Runtimes (Saniye)	Run-1	Run-2	Run-3	Run-4	Run-5	Webtester	TestSSL	sslyze
Google.com.tr	2,58278	2,634385	2,579197	2,634	2,741093	2,634291	24,972	7,983
Facebook.com	2,293766	2,206636	2,246678	2,244771	2,29175	2,2567202	27,348	11,825
Eksisozluk.com	0,850828	0,761125	0,764246	0,743234	0,752003	0,7742872	32,027	10,238
Onedio.com	0,728606	0,848785	0,768519	0,76946	0,782283	0,7795306	20,337	8,475
Twitter.com	2,108905	2,181846	2,170356	2,147076	2,159346	2,1535058	33,837	12,913
Sahibinden.com	0,75006	0,813765	0,839943	0,797766	0,843068	0,8089204	20,086	12,391
Hurriyet.com.tr	0,747841	0,765826	0,774504	0,785612	0,791916	0,7731398	21,58	10,981
Sabah.com.tr	2,82166	2,754933	3,144672	2,828507	2,784946	2,8669436	21,328	7,515
Live.com	2,515994	2,568236	2,582134	2,579347	2,607058	2,5705538	25,058	11,085
Yandex.com.tr	3,196748	3,200767	3,250077	3,268768	3,730358	3,3293436	25,588	13,547
Ensonhaber.com	3,175294	2,238133	2,310035	2,25593	2,238533	2,443585	25,926	13,588
Instagram.com	8,699061	8,481175	8,611573	8,879679	8,400518	8,6144012	59,038	21,222
Blogspot.com.tr	4,177972	3,062163	2,691674	3,707179	2,717552	3,271308	24,986	9,284
Sozcu.com.tr	0,288096	0,279062	0,884472	1,079288	1,810305	0,8682446	18,376	
Hepsiburada.com	1,398132	0,786489	0,820191	1,910362	0,767712	1,1365772	20,048	8,938
Mynet.com	1,351913	0,827347	0,855698	0,856158	0,814285	0,9410802	20,545	13,231
Haberturk.com	0,70244	0,743627	0,770647	0,766572	0,766824	0,750022	20,794	8,423
Acunn.com	0,69384	0,811595	0,785018	0,758731	0,782924	0,7664216	21,072	9,871
Yenisafak.com	2,883425	2,850747	2,88201	2,869614	2,837179	2,864595	23,662	15,069
Donanimhaber.com	0,848237	0,781686	0,836088	0,818684	0,783218	0,8135826	31,763	9,915
Garanti.com.tr	0,762222	0,82761	0,884823	0,796248	0,814494	0,8170794	20,174	13,748
Kizlarsoruyor.com	0,756929	0,817608	0,817117	0,868161	0,856376	0,8232382	21,865	9,258
R10.net	2,259833	2,156335	2,176054	2,160143	2,206692	2,1918114	22,417	8,278
Msn.com	7,508607	7,025061	7,971217	7,973803	7,979902	7,691718	36,635	18,013
Stackoverflow.com	2,223386	2,221129	2,2435	2,179487	2,273401	2,2281806	23,897	13,545
Sporx.com	2,298259	2,304078	2,334013	2,481555	2,362143	2,3560096	23,763	10,622
ORTALAMA						**2,21**	**25,66**	**11,60**

Tablo 8.3: Offline Sunucu Konfigürasyonları

	Desteklediği Protokol Versiyonları	**Desteklediği Algoritmalar**	**Bulundurduğu Açıklıklar**
Server-1	TLSv1.2,TLSv1.1 TLSv1.0,SSLv3	DHE,ECDHE RSA,ECDSA, DSA,SHA1, SHA256,MD5, AES128,3DES	Beast,Crime, TIME,Poodle
Server-2	TLSv1.2,TLSv1.1 TLSv1.0	DHE,ECDHE RSA,ECDSA, DSA,SHA1, SHA256,3DES, AES128,AES256	Lucky13
Server-3	SSLv3	DH,ECDH RSA,SHA1,MD5, AES128	Beast, Lucky13
Server-4	TLSv1.1,TLSv1.0 SSLv3, SSLv2	DHE,ECDHE RSA,ECDSA, DSA,SHA1,3DES SHA256,MD5 AES128,DES AES256	Drown, Freak, Logjam, Bleichenbacher
Server-5	TLSv1.1,TLSv1.0 SSLv3	DHE,ECDHE RSA,ECDSA, DSA,SHA1, SHA256,MD5, AES256,RC4	Poodle

Tablo 8.4: Offline Çalışma Süresi Karşılaştırması

Runtimes (Saniye)	Run-1	Run-2	Run-3	Run-4	Run-5	Webtester (Ortalama)	TestSSL	sslyze
Server-1	0,523	0,533	0,512	0,547	0,521	0,5272	6,357	3,158
Server-2	0,401	0,411	0,407	0,389	0,405	0,4026	6,616	3,494
Server-3	0,389	0,393	0,399	0,385	0,393	0,3918	5,797	3,219
Server-4	0,525	0,533	0,522	0,543	0,544	0,5334	6,389	3,351
Server-5	0,489	0,456	0,466	0,481	0,465	0,4714	5,912	3,11
ORTALAMA						**0,47**	**6,21**	**3,27**

8.6.2 Başarı Oranları

Uygulamanın performans testlerinin yanı sıra, Hedef sunucuya ait elde edilmek istenen bilgileri ne derece başardığı da önemlidir. Önceki bölümlerde bahsedildiği gibi Webtester uygulaması performans artışı sağlamak için birtakım taramaları diğer uygulamalardan farklı olarak gerçekleştirmektedir. Bu farklılıklardan ötürü de hedef sunucular ile ilgili edinilen bilgiler eksik veya yanlış olabilmektedir. Bu yüzden hedef kriterler kullanılarak başarı oranları tespit edilmiştir. Başarı ölçütleri olarak seçilen kriterler aşağıdaki gibidir:

1. Bağlantı protokollerinin tespit edilmesi kriteridir. SSLv2, SSLv3, TLSv1.0, TLSv1.1, TLSv1.2 olmak üzere 5 protokol için her birinin olup olmadığının tespiti 5 adet kriteri temsil eder.
2. Bölüm 8.4.1'da detaylı bahsedilen her bir özelliğin tespit edilmesi kriterleridir. Toplamda 10 adet kriter bu kısımda bulunmaktadır. Bu kriterler saldırıların anlaşılmasında kullanılmaktadır.
3. Heartbleed saldırısını test edebilme kriteridir. Heartbleed saldırısı ön koşulları tespit edilip çıkarım yapmak ile tespit edilememektedir. Bu yüzden saldırıyı gerçekleştirmek gerekir. Bir adet kriter de bu aşamada eklenmiştir.

Toplamda Webtester uygulamasının her bir çalışmasında 16 kriteri doğru tespit edip etmediği kontrol edilerek başarı yüzdesi ortaya çıkarılabilir. Başarı oranı testi için aşağıdaki koşullarda işlemler gerçekleştirilmiştir.

- Hedef sunucular öncelikle $https://www.ssllabs.com/ssltest/$ sitesinde test edilip, sonuçlar belirlenmiş 16 kriter üzerinden Webtester uygulaması ile kıyaslanmıştır. Toplamda 20 farklı sunucu için test yapılmıştır.
- Başarı yüzdesi hesaplanırken objektif bir sonuç elde edebilmek amacıyla kapalı ağdaki cihazlar göz önünde bulundurulmamış, Internet üzerinde tarama yapılmıştır.
- Her bir tarama 10 kez denenmiş ve böylece ağ üzerinde oluşabilecek hata paylarından bağımsız sonuç elde edilmiştir.
- Webtester uygulamasının sonuçları "SSLLabs" web sitesindeki uygulamanın sonuçları doğru kabul edilerek kıyaslanmıştır.

Yukarıda belirtilen kıstaslara göre yapılan testler sonucunda aşağıdaki sonuçlar elde edilmiştir:

Tablo 8.5: Başarı Oranları

	True	False
Positive	157	2
Negative	158	1
Success Rate	%99	

- Protokol versiyonları tespitinde 5 kriter üzerinden 20 site için 100 üzerinden 99 kriterde aynı sonuç elde edilmiştir. Farklı sonuç elde edilen kısım, Bölüm 8.4.1'da öngörülen ve yalnızca SSLv2 desteklenen sunucuların tespiti kısmı olmuştur. Tespit edilmesi gereken özelliğin farkedilememesinden ötürü elde edilen bu farklılık "False Negative" olarak kabul edilebilir. 20 site için desteklenen 67 adet desteklenen protokol bilgisinde, 32 adet desteklenmeyen protokol bilgisinde hemfikir olunmuştur. Bu aşama için 67 adet "True Positive", 32 adet "True Negative" verisi elde edilmiştir.

- Desteklenen algoritmaların tespitinde 10 kriter üzerinden 20 site için 200 üzerinden 200 kriterde aynı sonuç elde edilmiştir. 200 kritere ait 92 adet desteklenen algoritma, 108 adet desteklenmeyen algoritma tespit edilmiştir. 92 adet desteklenen algoritma "True Positive", 108 adet desteklenmeyen algoritma "True Negative" olarak kabul edilmiştir.

- Heartbleed saldırısı tespitinde 1 kriter üzerinden 20 site için 20 üzerinden 18 kriterde aynı sonuç elde edilmiştir. Elde edilen 2 farklı sonuç, heartbleed açıklığı bulunmayan sunucuların gönderdiği cevapları yanlış okumaktan kaynaklanmaktadır. Webtester uygulamasının açıklık olduğunu bildirdiği bu iki durumda aslında açıklık bulunmamaktadır. Aynı sonuç elde edilen 18 durum "True Negative", diğer 2 durum "False Positive" olarak kabul edilmiştir.

Sonuç olarak 320 kriter üzerinden 3 kriterde farklılık ortaya çıkmıştır. Elde edilen sonuçlara ait Tablo 8.5'deki gibidir:

Bölüm 9

Sonuç

9.1 Sonuç

Bu tez çalışmasında biz, Internet güvenliğinin en önemli parçası olan SSL/TLS protokollerini ve açık anahtar altyapısını inceledik. İnceleme boyunca konuların hem teknik altyapıları hem de güncel yansımaları anlaşılır bir şekilde ifade etmeye çalıştık.

Tez çalışması sırasında gözlemlenen bilgi ve birikimlerle kapalı ağlara özel ve pratik bir web uygulamasının gerekli olduğu farkedilmiştir. Bu nedenle Webtester adında uygulama geliştirdik. Uygulama mevcut protokolleri ve şifre paketlerini kontrol eden ve komut satırı üzerinden çalıştırılabilen bir uygulamadır. Tez sonrasında kaynak kodları paylaşılacak olan kod üzerinde uzun vadede geliştirmeler yapılabilir. Bu geliştirmeler, grafik arayüzü ekleme, paralel paketlerle daha hızlı tarama gibi özelliklere sahip olabilir.

Bu çalışmada ortaya çıkmaktadır ki, bilgi güvenliği ve özelinde ağ güvenliği günden güne önemini artırmaktadır. Ağ güvenliği kavramı da çok disiplinli bir kavram haline gelmiştir. Ağ üzerinde oluşabilecek ve bir çok farklı senaryodan kaynaklanabilecek sorunlar vardır. Bu sorunların çözümü için bütünlük, mahremiyet ve güvenlik içeren verilerin korunması, iletilmesi ve doğrulanması işlemleri başarılı bir şekilde yapılmalıdır.Bizim amacımız, ağ güvenliğinin sağlanabilmesinin zorlu bir süreç olduğunu ve güvenlik kavramının hiçbir zaman kesin olarak çözümlenebilecek bir kavram olmadığını gösterebilmektir.

Çalışma sürecinde bahsedilen açıklıkların büyük çoğunluğu şifreleme ve özet alma yöntemlerinden güvensiz olan algoritmaların kullanılmasından kaynaklanmıştır. Bu saldırıların önüne geçebilmek için güvenli şifre paketleri tercih edilmelidir.

Kaynakça

[1] CVE-2012-4929. Available from MITRE, CVE-ID CVE-2012-4929., Dec. 3 2012. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2012-4929`.

[2] J. Aas. Let's Encrypt: Delivering SSL/TLS Everywhere. URL `https://letsencrypt.org/`. (accessed: 2018-10-29).

[3] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, et al. Imperfect forward secrecy: How diffie-hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17. ACM, 2015.

[4] N. Aifardan, D. Bernstein, K. Paterson, B. Poettering, and J. Schuldt. On the security of rc4 in tls and wpa. In *USENIX Security*, 2013.

[5] N. J. Al Fardan and K. G. Paterson. Lucky thirteen: Breaking the tls and dtls record protocols. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 526–540. IEEE, 2013.

[6] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt. DROWN: Breaking TLS with SSLv2. In *25th USENIX Security Symposium*, Aug. 2016.

[7] R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, and J.-K. Tsay. Efficient padding oracle attacks on cryptographic hardware. In *Advances in Cryptology–CRYPTO 2012*, pages 608–625. Springer, 2012.

[8] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski. Arpki: Attack resilient public-key infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 382–393. ACM, 2014.

[9] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 92–111. Springer, 1994.

[10] L. Bello. Dsa-1571-1 openssl–predictable random number generator. *Debian Security Advisory*, 2008.

[11] M. Benantar. Method and system for a secure binding of a revoked x.509 certificate to its corresponding certificate revocation list, June 13 2002. URL `https://www.google.com/patents/US20020073310`. US Patent App. 09/734,809 (accessed 2018-02-10).

[12] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of tls. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 535–552. IEEE, 2015.

[13] T. Be'ery and A. Shulman. A perfect crime? only time will tell. *Black Hat Europe*, 2013, 2013.

[14] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs# 1. In *Annual International Cryptology Conference*, pages 1–12. Springer, 1998.

[15] C. Boyd and A. Mathuria. *Protocols for authentication and key establishment*. Springer Science & Business Media, 2013.

[16] J. Buchmann. *Introduction to cryptography*. Springer Science & Business Media, 2013.

[17] H. Böck. DSA should die. URL `https://www.ietf.org/mail-archive/web/tls/current/msg15773.html`. (accessed: 2018-09-30).

[18] E. P. I. Center. Epic - international traffic in arms regulations. URL `https://epic.org/crypto/export_controls/itar.html`.

[19] C. Cimpanu. 14,766 Let's Encrypt SSL Certificates Issued to PayPal Phishing Sites. URL `https://www.bleepingcomputer.com/news/security/14-766-lets-encrypt-ssl-certificates-issued-to-paypal-phishing-sites/`.

[20] G. I. Davida. *Chosen signature cryptanalysis of the RSA (MIT) public key cryptosystem*. Department of Electrical and Computer Science, College of Engineering and Applied Science, University of Wisc., 1982.

[21] B. den Boer and A. Bosselaers. Collisions for the compression function of md5. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 293–304. Springer, 1993.

[22] A. W. Dent and C. J. Mitchell. *User's Guide To Cryptography And Standards (Artech House Computer Security)*. Artech House, Inc., 2004.

[23] L. P. Deutsch. Deflate compressed data format specification version 1.3. 1996.

[24] T. Dierks and C. Allen. The tls protocol version 1.0. 1999.

[25] T. Dierks and E. Rescorla. Rfc 5246: The transport layer security (tls) protocol. *The Internet Engineering Task Force*, 2008.

[26] W. Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5):560–577, 1988.

[27] W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[28] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, Apr. 2011. ISSN 0001-0782. doi: 10.1145/1924421.1924442. URL `http://doi.acm.org/10.1145/1924421.1924442`.

[29] B. Dowling and D. Stebila. Modelling ciphersuite and version negotiation in the tls protocol. In *Australasian Conference on Information Security and Privacy*, pages 270–288. Springer, 2015.

[30] Drwetter. drwetter/testssl.sh. URL `https://github.com/drwetter/testssl.sh`.

[31] T. Duong and J. Rizzo. Here come the xor ninjas. *White paper, Netifera*, 2011.

[32] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson. The matter of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pages 475–488, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3213-2. doi: 10.1145/2663716.2663755. URL `http://doi.acm.org/10.1145/2663716.2663755`.

[33] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, et al. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*, pages 475–488. ACM, 2014.

[34] D. Eastlake 3rd and P. Jones. Us secure hash algorithm 1 (sha1). Technical report, 2001.

[35] P. Eckersley. Sovereign key cryptography for internet domains. *https://git. eff. org*, 2011.

[36] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 10–18. Springer, 1984.

[37] C. Evans, C. Palmer, and R. Sleevi. Public key pinning extension for http. Technical report, 2015.

[38] P. FIPS. 186-4. *Digital Signature Standard (DSS)*, 2013.

[39] H. F. Gaines. *Cryptanalysis: A study of ciphers and their solution.* Courier Corporation, 2014.

[40] S. Galperin, S. Santesson, M. Myers, A. Malpani, and C. Adams. X. 509 internet public key infrastructure online certificate status protocol-ocsp. 2013.

[41] D. Giry. Keylength - nist report on cryptographic key length and cryptoperiod (2016). URL `https://www.keylength.com/en/4/`. (accessed: 2017-08-01).

[42] D. Giry. Keylength – cryptographic key length recommendation, 2008. `http://www.keylength.com/` (accessed 2017-02-04).

[43] Y. Gluck, N. Harris, and A. Prado. Breach: reviving the crime attack. *Unpublished manuscript*, 2013.

[44] I. Goldberg and D. Wagner. Randomness and the netscape browser. *Dr Dobb's Journal-Software Tools for the Professional Programmer*, 21(1):66–71, 1996.

[45] D. Goodin. Https-crippling freak exploit affects thousands of android and ios apps, Mar 2015. URL `http://arstechnica.com/security/2015/03/https-crippling-freak-exploit-hits-thousands-of-android-and-ios-apps/`.

[46] D. S. J. S. G. Greenwood and Z. L. L. Khan. Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps. 2014.

[47] B. He, V. Rastogi, Y. Cao, Y. Chen, V. Venkatakrishnan, C. Xiong, R. Yang, and Z. Zhang. Sslint: A tool for detecting tls certificate validation vulnerabilities. 2016.

[48] C. Hlauschek, M. Gruber, F. Fankhauser, and C. Schanes. Prying open pandora's box: KCI attacks against TLS. In *Proceedings of the 9th USENIX Conference on Offensive Technologies*, WOOT'15, Berkeley, CA, USA, 2015. USENIX Association.

[49] R. Housley, W. Polk, W. Ford, and D. Solo. Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile. Technical report, 2002.

[50] R. Housley, W. Ford, W. Polk, and D. Solo. Rfc 5280: Internet x. 509 public key infrastructure certificate and crl profile, 2008.

[51] Ioerror. ioerror/sslscan. URL `https://github.com/ioerror/sslscan`.

[52] F. IT. Black tulip: Report of the investigation into the diginotar certificate authority breach, August 2012. `https://www.rijksoverheid.nl/binaries/rijksoverheid/documenten/rapporten/2012/08/13/black-tulip-update/black-tulip-update.pdf`.

[53] B. Kaliski. Pkcs# 1: Rsa encryption version 1.5. 1998.

[54] B. Kaliski. Pkcs# 5: Password-based cryptography specification version 2.0. 2000.

[55] B. S. Kaliski Jr. An unknown key-share attack on the mqv key agreement protocol. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):275–288, 2001.

[56] J. Kelsey. Compression and information leakage of plaintext. In *International Workshop on Fast Software Encryption*, pages 263–276. Springer, 2002.

[57] T. H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, and V. Gligor. Accountable key infrastructure (aki): A proposal for a public-key validation infrastructure. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 679–690, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2035-1. doi: 10.1145/2488388.2488448. URL `http://doi.acm.org/10.1145/2488388.2488448`.

[58] V. Klima, O. Pokornỳ, and T. Rosa. Attacking rsa-based sessions in ssl/tls. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 426–440. Springer, 2003.

[59] H. Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In *Annual International Cryptology Conference*, pages 546–566. Springer, 2005.

[60] D. M. Kristol and L. Montulli. Http state management mechanism. 2000.

[61] R. Laboratories. Frequently asked questions about today's cryptography, 2000. `http://www.nordugrid.org/documents/rsalabs_faq41.pdf` (accessed 2017-05-04).

[62] A. Langley. Imperialviolet. URL `https://www.imperialviolet.org/2012/01/15/beastfollowup.html`.

[63] B. Laurie, A. Langley, and E. Kasper. Certificate transparency. Technical report, 2013.

[64] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003.

[65] A. Lenstra, X. Wang, and B. de Weger. Colliding x. 509 certificates based on md5-collisions.

[66] A. K. Lenstra, H. W. Lenstra Jr, M. S. Manasse, and J. M. Pollard. The number field sieve. In *The development of the number field sieve*, pages 11–42. Springer, 1993.

[67] D. J. Malan, M. Welsh, and M. D. Smith. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 71–80. IEEE, 2004.

[68] M. Marlinspike. Ssl vulnerability - moxie.org, May 2008. URL `https://moxie.org/ie-ssl-chain.txt`.

[69] M. Marlinspike. Defeating ocsp with the character '3'. *Blackhat 2009*, 2009.

[70] M. Marlinspike. More tricks for defeating ssl in practice. *Black Hat USA*, 2009.

[71] U. M. Maurer. Towards the equivalence of breaking the diffie-hellman protocol and computing discrete logarithms. In *Annual International Cryptology Conference*, pages 271–281. Springer, 1994.

[72] P. Mell, T. Grance, et al. The nist definition of cloud computing. 2011.

[73] A. Menezes, M. Qu, and S. Vanstone. Some new key agreement protocols providing mutual implicit authentication. *SecondWorkshop on Selected Areas in Cryptography (SAC 95)*, 1995.

[74] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[75] V. S. Miller. Use of elliptic curves in cryptography. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 417–426. Springer, 1985.

[76] B. Möller, T. Duong, and K. Kotowicz. This poodle bites: exploiting the ssl 3.0 fallback. *Security Advisory*, 2014.

[77] M. S. Niaz and G. Saake. Merkle hash tree based techniques for data integrity of outsourced data. In *GvD*, pages 66–71, 2015.

[78] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing tcp's retransmission timer. Technical report, 2011.

[79] Qualys. SSL Server Test. URL `https://www.ssllabs.com/ssltest/`.

[80] E. Rescorla. The transport layer security (tls) protocol version 1.3. Technical report, 2018.

[81] I. Ristić. Bulletproof SSL and TLS. *Feisty Duck*, 2014.

[82] R. Rivest. The MD5 message-digest algorithm. United States, 1992. RFC Editor.

[83] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[84] P. G. Sarkar and S. Fitzgerald. Attacks on ssl a comprehensive study of beast, crime, time, breach, lucky 13 & rc4 biases. *iSEC Partners*, 2013.

[85] A. P. Sarr, P. Elbaz-Vincent, and J.-C. Bajard. A secure and efficient authenticated diffie–hellman protocol. In *European Public Key Infrastructure Workshop*, pages 83–98. Springer, 2009.

[86] K. Schmeh. *Cryptography and public key infrastructure on the Internet*. John Wiley & Sons, 2006.

[87] B. Segal. A short history of internet protocols at cern. *Professional webpage. April. http://ben. home. cern. ch/ben/TCPHIST. html*, 1995.

[88] R. Shirey. Rfc 4949–internet security glossary, 2007.

[89] A. Sotirov, M. Stevens, J. Appelbaum, A. K. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. Md5 considered harmful today, creating a rogue ca certificate. In *25th Annual Chaos Communication Congress*, number EPFL-CONF-164547, 2008.

[90] M. Stevens, A. Lenstra, and B. de Weger. Chosen-prefix collisions for md5 and colliding x.509 certificates for different identities. In M. Naor, editor, *Advances in Cryptology - EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings*, pages 1–22, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-72540-4. doi: 10.1007/978-3-540-72540-4_1.

[91] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. The first collision for full sha-1. In *Annual International Cryptology Conference*, pages 570–596. Springer, 2017.

[92] Symantec. Check Your SSL/TLS Certificate Installation. URL `https://www.websecurity.symantec.com/support/ssl-checker`. (Accessed: 2018-09-21).

[93] S. Turner and T. Polk. Prohibiting secure sockets layer (ssl) version 2.0. Technical report, 2011.

[94] O. Vanska. A finnish man created this simple email account - and received microsoft's security certificate.

[95] S. Vaudenay. Security flaws induced by cbc padding — applications to ssl, ipsec, wtls... In L. R. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, The Netherlands, April 28 – May 2, 2002 Proceedings*, pages 534–545, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-46035-0. doi: 10.1007/3-540-46035-7_35.

[96] B. Vesterås. Analysis of key agreement protocols. Master's thesis, Department of Computer Science and Media Technology, Gjovik University College, 2006.

[97] D. Wagner and B. Schneier. Analysis of the ssl 3.0 protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, WOEC'96, pages 29–40, Berkeley, CA, USA, 1996. USENIX Association.

[98] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the hash functions md4 and ripemd. pages 1–18, 2005. doi: 10.1007/11426639_1.

[99] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full sha-1. In *Annual International Cryptology Conference*, pages 17–36. Springer, 2005.

[100] H.-A. Wen, C.-L. Lin, and T. Hwang. Provably secure authenticated key exchange protocols for low power computing clients. *Computers & Security*, 25(2):106–113, 2006.

[101] F. Xia, L. T. Yang, L. Wang, and A. Vinel. Internet of things. *International Journal of Communication Systems*, 25(9):1101, 2012.

[102] A. Young and M. Yung. Kleptography: Using cryptography against cryptography. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 62–74. Springer, 1997.

[103] Y. Zhao, S. Vemuri, J. Chen, Y. Chen, H. Zhou, and Z. Fu. Exception triggered dos attacks on wireless networks. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 13–22. IEEE, 2009.

[104] X. Zheng. Phishing with unicode domains. URL `https://www.xudongz.com/blog/2017/idn-phishing/`.

[105] S. Zhioua. The middle east under malware attack dissecting cyber weapons. In *Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on*, pages 11–16. IEEE, 2013.

[106] R. Zuccherato. Root certificate management system and method, July 16 2001. US Patent App. 09/906,504.