DÜZCE ÜNİVERSİTESİ | LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MOBİL UYGULAMA İLE API SERVİSLERİ ARASINDA SAYAÇ TABANLI GÜVENLİK MEKANİZMASI

RESUL SİLAY

YÜKSEK LİSANS TEZİ
ELEKTRİK-ELEKTRONİK VE BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

DANIŞMAN
DOÇ. DR. GÜLÇİN ERSÖZ DEMİR

DÜZCE, 2024

T.C.
DÜZCE ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MOBİL UYGULAMA İLE API SERVİSLERİ ARASINDA SAYAÇ TABANLI GÜVENLİK MEKANİZMASI

Resul Silay tarafından hazırlanan tez çalışması aşağıdaki jüri tarafından Düzce Üniversitesi Lisansüstü Eğitim Enstitüsü Elektrik-Elektronik ve Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Doç. Dr. Gülçin Ersöz Demir

Düzce Üniversitesi

Jüri Üyeleri

Doç. Dr. Gülçin Ersöz Demir
Düzce Üniversitesi _____

Prof. Dr. İbrahim Yücedağ
Düzce Üniversitesi _____

Dr. Öğr. Üyesi Semih Çakır
Zonguldak Bülent Ecevit Üniversitesi _____

Tez Savunma Tarihi: 12/02/2024

BEYAN

Bu tez çalışmasının kendi çalışmam olduğunu, tezin planlanmasından yazımına kadar bütün aşamalarda etik dışı davranışımın olmadığını, bu tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara kaynak gösterdiğimi ve bu kaynakları da kaynaklar listesine aldığımı, yine bu tezin çalışılması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını beyan ederim.

12 Şubat 2024

Resul Silay

TEŞEKKÜR

Yüksek Lisans öğrenimimde ve bu tezin hazırlanmasında gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Doç. Dr. Gülçin Ersöz Demir'e en içten dileklerimle teşekkür ederim.

Bu çalışmam boyunca bana zamanından fedakârlık gösteren ve ilgisini esirgemeyen sevgili eşim ve aileme sonsuz teşekkürlerimi iletiyorum.

12 Şubat 2024 **Resul SİLAY**

İÇİNDEKİLER

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

ÇİZELGE LİSTESİ

Sayfa No

KISALTMALAR

AAR	Android Archive / Android Arşiv
ABI	Application Binary Interface / Uygulama İkili Arabirimi
AES	Advanced Encryption Standard / Gelişmiş Şifreleme Standardı
API	Application Programming Interface / Uygulama Programlama Arayüzü
ART	Android Runtime / Android Çalışma Zamanı
CPU	Central Processing Unit / Merkezî İşlem Birimi
GUI	Graphical User Interface / Grafiksel Kullanıcı Arayüzü
HAL	Hardware Abstraction Layer / Donanım Soyutlama Katmanı
HTTPS	Hypertext Transfer Protocol Secure
JNI	Java Native Interface / Java Yerel Arayüzü
JSON	JavaScript Object Notation / JavaScript Nesne Notasyonu
JWT	JSON Web Token
MFA	Multi-Factor Authentication / Çok Faktörlü Kimlik Doğrulaması
MITM	Man–In-The-Middle
MVC	Model-View-Controller
NDK	Native Development Kit / Yerel Geliştirme Kiti
OTP	One-Time Password / Tek Kullanımlık Şifre
RAM	Random Access Memory / Rastgele Erişimli Hafıza
REST	Representational State Transfer / Temsili Durum Transferi
RSA	Rivest-Shamir-Adleman
SDK	Software Development Kit / Yazılım Geliştirme Kiti
SMS	Short Message Service / Kısa Mesaj Servisi
SSL	Secure Sockets Layer / Güvenli Yuva Katmanı
TLS	Transport Layer Security / Taşıma Katmanı Güvenliği
VM	Virtual Machine / Sanal Makine
Wi-Fi	Wireless Fidelity

SİMGELER

GB	Gigabyte
MB	Megabyte
ms	Milisaniye

ÖZET

MOBİL UYGULAMA İLE API SERVİSLERİ ARASINDA SAYAÇ TABANLI GÜVENLİK MEKANİZMASI

Resul SİLAY

Düzce Üniversitesi
Lisansüstü Eğitim Enstitüsü, Elektrik-Elektronik ve Bilgisayar Mühendisliği
Anabilim Dalı
Yüksek Lisans Tezi
Danışman: Doç. Dr. Gülçin ERSÖZ DEMİR

Şubat 2024, 72 sayfa

Bu çalışma, mobil cihazların geniş kullanımı ve beraberinde getirdiği güvenlik zorluklarına odaklanarak, mobil uygulamalar ile Uygulama Programlama Arabirimi (Application Programming Interface-API) servisleri arasındaki güvenli iletişimi güçlendirmeyi amaçlayan 'cTrust' isimli sayaç tabanlı yenilikçi bir güvenlik mekanizması geliştirmeyi hedeflemektedir. Sayaç tabanlı güvenlik mekanizması, mobil uygulamaların API servisleriyle güvenli bir şekilde iletişim kurabilmesi için kullanılan etkili bir strateji sunmaktadır. Geliştirilen mekanizma, mobil uygulamaların doğrulama süreçlerini başlatır ve tek kullanımlık token bilgisini (cToken) üreterek, sürekli değişen bir doğrulama süreci sunmaktadır. Doğrulama sürecinde kullanılan sayaç bilgisi, gizli anahtar ile şifrelenip sadece API servisi tarafında çözülerek, tek kullanımlık doğrulama mekanizmasının etkinliğini artırmaktadır. Bu strateji, API servislerinin güvenliğini artırmak adına etkili bir çözüm olarak öne çıkmaktadır. Çalışma, mevcut güvenlik yaklaşımlarına alternatif bir bakış açısı sunarak mobil uygulama ve API servisleri arasındaki güvenli etkileşimi sağlamayı amaçlamaktadır. Sayaç mekanizması, geleneksel güvenlik yöntemlerine ek olarak benzersiz bir güvenlik katmanı ekleyerek, saldırılara karşı daha dayanıklı bir yapı oluşturmaktadır. Bu bağlamda, mobil uygulama ve API servisleri arasındaki güvenliği artırarak, dijital iletişimde alternatif bir güvenlik mekanizması sunmayı amaçlamaktadır. Sonuç olarak, mobil uygulamaların ve API servislerinin güvenliği günümüzde kritik bir öneme sahiptir. Sayaç tabanlı güvenlik mekanizması cTrust, güvenliği artırmak için etkili bir çözüm sunmakta ve sektördeki güvenlik standartlarına uygun bir potansiyel içermektedir. Bu çalışma, mobil uygulamalar ile API servisleri arasındaki güvenliği artırmayı amaçlarken aynı zamanda alternatif ve bağımsız bir güvenlik stratejisi sunmayı hedeflemektedir.

Anahtar Sözcükler: Sayaç Mekanizması, Mobil Güvenlik, API, Tasdik, Doğrulama

ABSTRACT

COUNTER-BASED SECURITY MECHANISM BETWEEN MOBILE APPLICATION AND API SERVICES

Resul SİLAY

Düzce University
Graduate School, Department of Electrical-Electronics and Computer Engineering
Master Thesis
Supervisor: Assoc. Dr. Gülçin ERSÖZ DEMİR

February 2024, 72 pages

This study aims to develop an innovative security mechanism called ‘cTrust’, focusing on the widespread use of mobile devices and the security challenges it brings, to enhance secure communication between mobile applications and Application Programming Interface (API) services. The counter-based security mechanism provides an effective strategy for mobile applications to communicate securely with API services. The developed mechanism initiates authentication processes for mobile applications and generates a one-time token (cToken), offering a continuously changing authentication process. The counter information used in the authentication process is encrypted with a secret key and decrypted only by the API service, thereby enhancing the effectiveness of the one-time authentication mechanism. This strategy stands out as an effective solution to enhance the security of API services. The study aims to provide a secure interaction between mobile applications and API services by offering an alternative perspective to existing security approaches. The counter mechanism adds a unique security layer in addition to traditional security methods, creating a more resilient structure against attacks. In this context, it aims to enhance security between mobile applications and API services and provide an alternative security mechanism in digital communication. Consequently, the security of mobile applications and API services holds critical importance today. The cTrust counter-based security mechanism offers an effective solution to enhance security and holds the potential to comply with industry security standards. While aiming to enhance security between mobile applications and API services, this study also aims to provide an alternative and independent security strategy.

Keywords: Counter Mechanism, Mobile Security, API, Attestation, Authentication

1. GİRİŞ

Teknolojinin ilerlemesi ve internet altyapısındaki gelişmeler, mobil uygulama sektöründe dinamik bir büyümeyi beraberinde getirmiştir [1], [2], [3], [4], [5], [6]. 2022'de dünya genelinde 6,4 milyarı aşan akıllı telefon mobil ağ abonelik sayısı, 2028'e kadar 7,7 milyarı geçmesi beklenirken, bu artışta Çin, Hindistan ve Amerika Birleşik Devletleri gibi lider ülkelerin yüksek mobil abonelik sayıları etkili olmaktadır [7]. Günümüzde mobil uygulamalar; fintech, bankacılık, sağlık, eğitim, kamu, lojistik gibi çeşitli birçok alanda hizmet vermektedir [8], [9], [10], [11], [12], [13], [14], [15], [16]. Çeşitlenen kullanım alanlarıyla birlikte, mobil uygulamalar ve Uygulama Programlama Arabirimi (Application Programming Interface-API) servislerinin güvenliği giderek daha fazla önem kazanmaktadır [17]. Kullanıcıların finansal bilgileri, kişisel verileri ve hassas bilgileri konusundaki endişeler, güvenlik önlemlerinin daha da güçlendirilmesini gerektirmektedir. Şirketler mobil uygulamalarının güvenliğini sağlamak için çeşitli güvenlik önlemleri almaktadırlar. Fakat API servisleri ile mobil uygulama arasındaki doğrulama ve haberleşme güvenliği en kritik güvenlik konularının başında gelmektedir. Mobil uygulamaların veri kaynakları ile haberleşmesinde yaygın olarak Temsil Edilebilir Durum Transferi (Representational State Transfer-REST) API servisleri kullanılmaktadır [18], [19], [20]. API'lerin doğrulanmış gerçek mobil uygulamalar dışında klonlanmış [21] veya API servisleri kullanılarak oluşturulan sahte uygulamalar üzerinden erişilememesi önemli bir güvenlik kriteridir. API servislerin hizmet verdiği doğrulanmış uygulamalar dışındaki uygulamalar veya araçlar üzerinden API servisleri ile iletişim kurulması sahte uygulamaların oluşturulmasına, dolandırıcılığa veya Dağıtık Hizmet Engelleme (Distributed Denial of Service-DDoS) saldırıları ile ağın işlevsiz kalmasına sebebiyet vermektedir [22], [23], [24]. Sayaç tabanlı güvenlik mekanizması cTrust, mobil uygulamalar ile API servisleri arasında iletişim güvenliğini ele alan özgün bir sayaç tabanlı güvenlik mekanizmasıdır. Bu mekanizma, mobil uygulama ve API servisleri arasındaki doğrulama ve haberleşme süreçlerini güçlendirmeyi amaçlamaktadır. Sayaç tabanlı güvenlik mekanizması cTrust'ın temel amacı, API servisleri ile etkileşimde bulunan mobil uygulamaların doğrulama süreçlerini güçlendirmek, yetkisiz erişimlere karşı önlemler almak olarak belirlenmiştir. Bu bağlamda, cTrust, sayaç tabanlı bir

güvenlik mekanizması kullanarak, her iletişim başlatıldığında benzersiz bir anahtar oluşturarak güvenliği artırmayı amaçlar. Bu benzersiz anahtarlar, her iletişimde güncellenir ve sürekli değişerek güvenlik açıklarını minimize etmeye yönelik bir strateji sunar. Sayaç tabanlı güvenlik mekanizması cTrust'ın, mobil uygulama ve API servisleri arasındaki iletişimi korumak için sunduğu bu benzersiz güvenlik mekanizması, sahte uygulamaların ve yetkisiz erişim girişimlerinin engellenmesine yardımcı olmaktadır. Ayrıca, bu mekanizma, mobil uygulama ve API servis sağlayıcılarına, kullanıcıların finansal ve kişisel bilgilerini daha etkili bir şekilde koruma yeteneği sağlayarak güvenilir bir dijital deneyim sunma amacını taşır.

Tez çalışmasının bölümleri şu şekilde oluşturulmuştur: 2. Bölümde, mobil uygulamaların API servisleri ile iletişiminde güvenlik çözümlerine dair bilgiler verilmiştir. 3. Bölümde, önerilen güvenlik mekanizması açıklanmış ve mobil uygulama ile API servisleri arasındaki iletişimi nasıl güvence altına aldığı detaylandırılmıştır. 4. Bölümde, önerilen mekanizmanın Yazılım Geliştirme Kiti (Software Development Kit-SDK) API, uygulama API ve mobil SDK olmak üzere üç yapıdan oluştuğu ve hangi platformlarda kullanılabileceği açıklanmıştır. 5. Bölümde, testler başlığı altında mekanizmanın performansının değerlendirilmesi için yapılan Grafiksel Kullanıcı Arayüzü (Graphical User Interface-GUI) testlerinin sanal ve fiziksel cihazlar üzerindeki sonuçları verilmiştir. 6. Bölümde, bulgular ve tartışma başlığı altında elde edilen test sonuçlarına dayanarak mekanizmanın başarım performansını ve güvenilirliğini vurgulayan bulgular sunulmuş, ardından 7. Bölümde, sonuçlar ve öneriler başlığı altında çalışmanın genel değerlendirmesi ile birlikte mobil uygulama ve API servisleri arasındaki doğrulama süreçleri için öneriler yer almaktadır.

2. KAVRAMSAL ÇERÇEVE

API servislerine doğrulanmamış sahte veya klon uygulamalar üzerinden erişim sağlaması kritik bir güvenlik zafiyeti oluşturmaktadır. Doğrulanmış mobil uygulamalar dışında klon ve sahte uygulamaların API servisleri ile iletişimini engelleyecek güçlü bir güvenlik altyapısının oluşturulması, mevcut güvenlik tehditlerine karşı etkili bir savunma sağlamak açısından kritik bir öneme sahiptir.

2.1. API ANAHTARLARI VE KİMLİK DOĞRULAMA

Mobil uygulama ile API servisi arasındaki iletişimin güvenliğini sağlamak için API anahtarları ve kimlik doğrulama mekanizmaları kullanılır [25]. Mobil uygulama tarafından kullanılan API anahtarları güvenli bir şekilde saklanmalı ve paylaşılmamalıdır. Kimlik doğrulama süreçleri, güçlü parolalar ve çok faktörlü doğrulama gibi yöntemlerle desteklenmelidir.

2.2. TEK KULLANIMLIK ŞİFRE (OTP)

Tek Kullanımlık Şifre (One Time Password - OTP), dijital güvenliği artırmak için kullanılan ve genellikle bir uygulama veya Kısa Mesaj Servisi (Short Message Service – SMS) aracılığıyla iletilen, belirli bir işlemi veya girişi doğrulamak için kullanılan geçici şifrelerdir [26], [27]. Özellikle bankacılık işlemleri, çevrimiçi alışverişler ve diğer güvenlik gerektiren çevrimiçi platformlarda yaygın olarak kullanılır. Bu geçici şifre, kullanıcıların belirli bir süre içinde doğrulama sürecini tamamlamalarını sağlar ve sadece bir kez kullanılabilir. Bu da güvenlik açısından daha sağlam bir yöntemdir, çünkü şifrelerin tekrar kullanılma riski yoktur. OTP mobil, hesap güvenliğini artırırken yetkisiz erişim riskini azaltmaya yardımcı olur [28], [29].

2.3. ÇOK FAKTÖRLÜ KİMLİK DOĞRULAMA (MFA)

Çok Faktörlü Kimlik Doğrulama (Multi-Factor Authentication - MFA), hesapların güvenliğini artırmak için birden fazla doğrulama faktörünü gerektiren bir güvenlik yöntemidir. Bu yöntemde, geleneksel kullanıcı adı ve şifrenin yanı sıra kullanıcıdan ek bir doğrulama faktörü sağlaması istenir [30], [31], [32], [33]. Örneğin, Google

Authenticator gibi bir uygulama aracılığıyla dinamik doğrulama kodları kullanılabilir [34], [35].

2.4. TOKEN TABANLI GÜVENLİK YAKLAŞIMLARI

Token tabanlı güvenlik sistemleri, kullanıcının kimliğini doğrulamak ve yetkilendirmek için özel bir token kullanır. Bu tokenlar, sahte uygulamalardan veya klon uygulamalardan gelen istekleri filtreleyerek, sadece doğrulanmış ve yetkilendirilmiş mobil uygulamaların API servisi ile iletişim kurmasına izin verir [36], [37], [38]. Bu yaklaşım, mobil uygulamanın kimliğini doğrulamak ve güvenli bir şekilde API servisine erişim sağlamak için etkili bir yöntemdir.

2.5. GÜVENLİ BAĞLANTI PROTOKOLLERİ VE ŞİFRELEME

Mobil uygulama ekosistemi içerisinde, API servisleri ile güvenli bir iletişim, güvenli bağlantı protokollerinin kapsamlı bir kullanımını gerektirir. Özellikle Secure Hyper Text Transfer Protocol (HTTPS), bu protokoller arasında öne çıkan bir uygulama olarak, veri iletimi sırasında şifreleme kullanarak kullanıcı verilerinin ve hassas bilgilerin korunmasını sağlar [39], [40], [41], [42]. Güvenli Yuva Katmanı (Secure Sockets Layer - SSL) ve Aktarım Katmanı Güvenliği (Transport Layer Security - TLS) gibi güvenlik protokollerinin temel altyapısını oluşturan bu protokoller, istemci ve sunucu servisleri arasında güvenli veri alışverişini garanti eder [43], [44], [45], [46]. Bu güvenlik mekanizmaları, sahte uygulamaların API servislerine müdahalelerini engelleyerek, kullanıcı bilgilerinin güvenli bir biçimde iletilmesini temin eder. Mobil uygulama güvenliğindeki bu bağlam, güvenli bağlantı protokollerinin etkin kullanımıyla güçlenir ve endüstri standartlarına uygun bir güvenlik seviyesi sağlar.

2.6. MOBİL UYGULAMA GÜVENLİĞİ ÇÖZÜMLERİ

SafetyNet, Play Integrity API ve DeviceCheck gibi güvenlik mekanizmaları, mobil uygulama geliştiricilerinin kullanıcılarına daha güvenli bir deneyim sunmalarına yardımcı olan araçlardır.

SafetyNet, Android işletim sistemi üzerinde çalışan bir güvenlik hizmetidir. Temel amacı, cihazın bütünlüğünü korumak ve kötü amaçlı yazılımları tespit etmektir. SafetyNet,

cihazın root edilip cihaza en üst düzeyde erişim izinleri verilip verilmediğini ve diğer güvenlik önlemlerini değerlendirerek güvenilir bir çalışma ortamının sürdürülmesine katkıda bulunur. Uygulamalar, SafetyNet API'yı kullanarak cihazın güvenilirliğini dinamik olarak kontrol edebilirler [47].

Play Integrity API, Google tarafından sunulan bir API'dır ve uygulamanın doğru kaynaklardan indirilmesi süreçlerinin güvenliğine odaklanmıştır. API, uygulama paketlerinin doğrulanmış kaynaklardan yüklendiğini ve bütünlüklerinin korunduğu bilgilerini sağlamaktadır. Sahte uygulama paketlerini tespit eder ve önler, bu da kullanıcıların güvenli bir şekilde uygulamalarını güncellemelerini ve indirmelerini sağlar. Play Integrity API, Google Play hizmetleri ile entegre bir biçimde çalışarak güvenlik standartlarını yükseltir [48].

DeviceCheck, Apple'ın iOS işletim sistemi için geliştirdiği bir güvenlik hizmetidir. Cihazın benzersiz bir kimliği ile işaretlenmesini sağlayarak güvenli bir kimlik doğrulama mekanizması sunar. DeviceCheck, API çağrıları sırasında bu güvenli kimliği kullanarak doğrulama yapar ve kullanıcının özel bilgilerini paylaşmadan güvenli bir iletişim kanalı sağlar. Bu, uygulamaların güvenilir ve gizlilik odaklı bir şekilde kullanıcılarla etkileşimde bulunmasını mümkün kılar [49].

3. MATERYAL VE YÖNTEM

3.1. SOFTWARE DEVELOPMENT KIT (SDK)

SDK, bir yazılımın geliştirilmesini kolaylaştırmak ve hızlandırmak için kullanılan bir araç, kütüphane ve belge setidir. Geliştiricilere belirli bir platformda veya programlama dilinde uygulama oluşturmak için gerekli araçları sunar [50].

3.2. JSON WEB TOKEN (JWT)

JSON Web Token (JWT), modern mobil ve web uygulamalarında yaygın olarak kullanılan bir standarttır. Bu standart, bilgilerin güvenli bir şekilde JavaScript Object Notation (JSON) formatında paylaşılmasını sağlar ve temel olarak verileri dijital olarak imzalayarak güvenli bir taşıma mekanizması sunar. JWT genellikle kimlik doğrulama ve bilgi paylaşımı amacıyla kullanılarak, güvenli iletişim ve yetkilendirme süreçlerine katkı sağlar [51], [52], [53].

3.3. ŞİFRELEME YÖNTEMLERİ

Şifreleme, bilgi güvenliğinde kritik bir rol oynar ve hassas verilerin korunmasında temel bir araçtır [54]. Simetrik ve asimetrik şifreleme, bu alandaki iki anahtar yaklaşımıdır.

3.3.1. Simetrik Şifreleme

Simetrik şifreleme algoritmaları, aynı anahtarın hem şifreleme işlemi için hem de şifre çözme işlemi için kullanıldığı bir şifreleme türüdür; bu algoritmaların temel özelliği, veriyi şifrelerken kullanılan anahtarın, veriyi çözerken de aynı anahtarın kullanılmasıdır [55]. Bu anahtar genellikle 128, 192 veya 256 bit uzunluğunda olabilir. Simetrik şifreleme algoritmaları hızlıdır ve genellikle büyük veri akışlarını şifrelemek için tercih edilirler. Ancak anahtar yönetimi zorluğu nedeniyle, anahtarların güvenli bir şekilde paylaşılması ve saklanması gerekmektedir [56]. Gelişmiş Şifreleme Standardı (Advanced Encryption Standard - AES), simetrik anahtarlı bir şifreleme algoritmasıdır [57], [58]. 128, 192 veya 256 bit uzunluğunda anahtarlar kullanır [59].

3.3.2. Asimetrik Şifreleme

Asimetrik şifreleme algoritmaları ise, her kullanıcıya ait iki ayrı anahtarın kullanıldığı bir şifreleme türüdür [60]. Biri genel anahtar (public key) ve diğeri ise özel anahtar (private key) [61]. Genel anahtar, veriyi şifrelemek ve şifrelenmiş veriyi çözmek için kullanılırken, özel anahtar sadece sahibi tarafından saklanır ve şifrelenmiş veriyi çözmek için kullanılır [62]. Asimetrik şifreleme algoritmaları, anahtar yönetimini kolaylaştırır ve güvenliği artırır. Ancak simetrik algoritmalar kadar hızlı olmayabilirler [63]. Rivest-Shamir-Adleman (RSA), asimetrik anahtarlı bir şifreleme algoritmasıdır [64]. Genel anahtar ve özel anahtar olmak üzere iki anahtar kullanır [65]. RSA'nın genellikle 1024, 2048 veya 4096 bit uzunluğunda anahtarları desteklemektedir [66].

3.4. ÖNERİLEN MEKANİZMA

Önerilen sayaç tabanlı güvenlik mekanizması cTrust, mobil uygulamaların doğrulanmış kaynaklar dışında API servisleri ile haberleşmesinin önüne geçmeyi amaçlamaktadır. Mobil uygulama, API servisleri ile haberleşmede tek kullanımlık token bilgisi üzerinden doğrulama yapılmasını içermektedir. Tek kullanımlık token bilgisi (cToken), mobil uygulamalarda kullanılmak üzere geliştirilen mobil SDK ile SDK API arasındaki doğrulama süreçleri sonucunda üretilmektedir. Mobil uygulama çalışmaya başladığında, mobil uygulama içerisine entegre edilen mobil SDK mobil uygulamaya ait paket ismi, paket imzası gibi temel parametreleri şifreli olarak SDK API servisine ileterek ilk doğrulama sürecini başlatmaktadır. Doğrulamanın başarıyla gerçekleşmesi durumunda SDK API sunucularında başlatılan zaman bilgisi mobil SDK'ya iletilir. Mobil uygulama için üretilen özel bir değer, doğrulama sonucunda iletilen zaman bilgisine eklenir ve uygulama arka planında sayaç başlatılır. Başlatılan sayaç mekanizması, uygulamanın API servisleri ile haberleşmesinde doğrulamayı gerçekleştirmek için kullanılacak olan tek kullanımlık token (cToken) bilgisinin üretilmesi için o anki sayaç bilgisini (sToken) üretmektedir. Üretilen sToken bilgisi içerisinde sayaç bilgisi ve SDK API tarafından üretilen yalnızca bir kez kullanılabilen nonce değerini içermektedir [67]. Nonce terimi, genellikle güvenli iletişim süreçlerinde tekrarlanan saldırılara karşı koruma sağlamak amacıyla kullanılan, benzersiz ve rastgele bir sayı veya değeri ifade eder. Nonce, özellikle şifreleme işlemlerinde ve kimlik doğrulama süreçlerinde kullanılır [68]. Sayaç ve nonce bilgilerini içeren sToken bilgisi şifrelenerek mobil SDK üzerinden SDK API servislerine iletilir. İletilen sToken bilgisi SDK API servislerinde çözülür. Nonce bilgisi

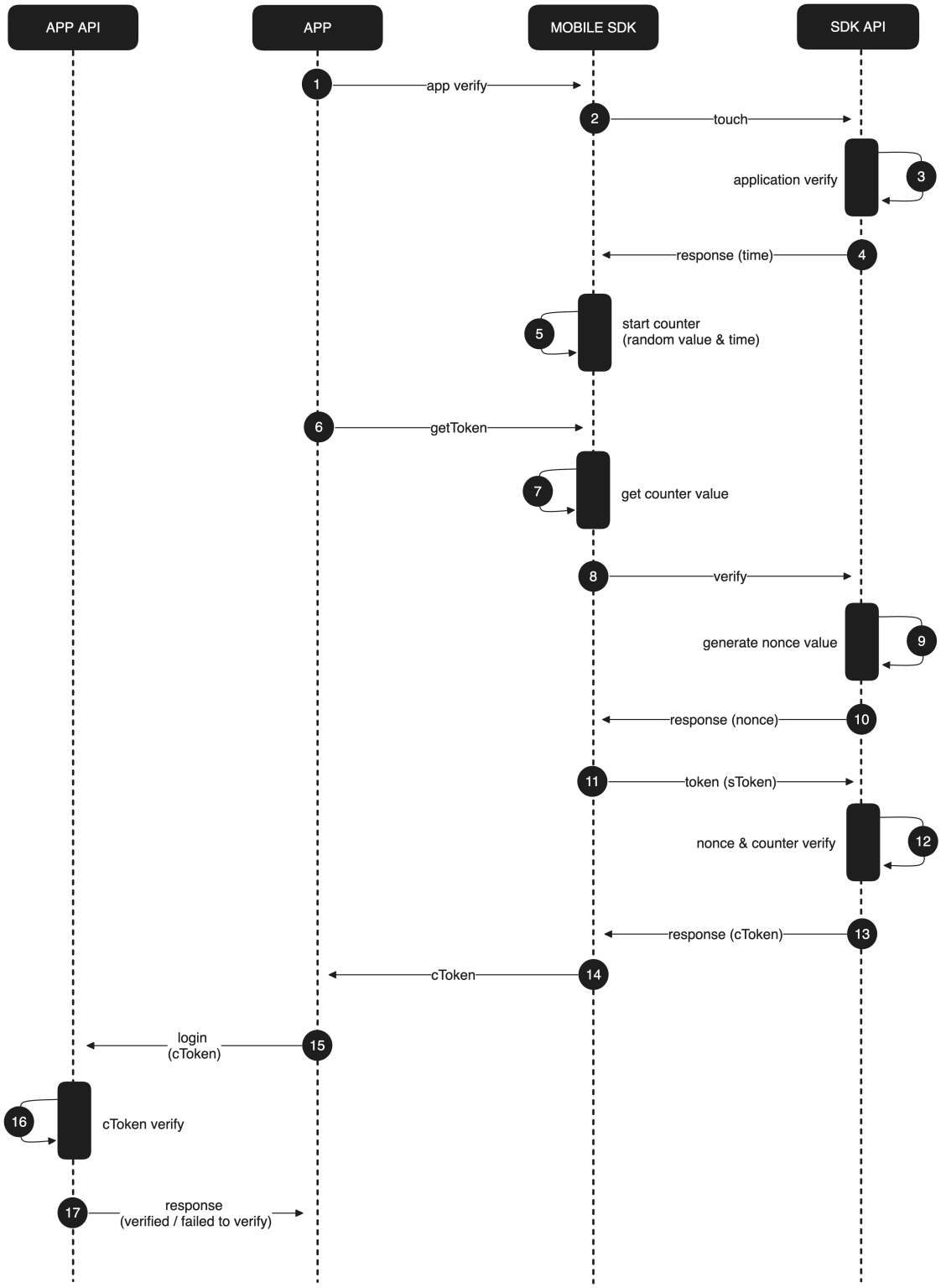
doğrulandıktan sonra sayaç bilgisi gerçek mobil uygulama için hesaplanan sayaç bilgisi ile karşılaştırılır. Sayaç bilgisi doğrulamasında iletişimden kaynaklı oluşabilecek zaman gecikmeleri için zaman aşımı belirlenmektedir. Sayaç bilgilerinin doğrulanması durumunda geçerli bir cToken üretilir. Doğrulamanın başarısız olması durumunda geçersiz bir cToken üretilir. Üretilen cToken bilgisinin geçerlilik süresi 3 saniye olarak belirlenmiştir. Ortadaki Adam (Man in The Middle - MITM) saldırıları, bilgi iletişiminde önemli bir tehdit oluşturan güvenlik açıklarından biridir [69]. Bu saldırı türü, bir saldırganın iletişimdeki iki taraf arasına gizlice girerek iletişimi izlemesine veya manipüle etmesine olanak tanır [70], [71], [72], [73].

Özellikle mobil uygulamalar ile API sunucuları arasındaki haberleşmede token bilgisinin geçerlilik süresi oldukça kritik bir öneme sahiptir. Örneğin, token'ın geçerlilik süresi çok kısa olduğunda, saldırganın token'ı ele geçirip zararlı faaliyetlerde bulunma süresi de sınırlı olacaktır. Bu, saldırganın potansiyel zararını azaltır ve güvenliği artırır.

Üretilen cToken bilgisinin geçerlilik süresi kontrol edildikten sonra geçerli bir token olup olmadığı bilgisi sadece API servisinde bulunan gizli anahtar aracılığı ile teyit edilebilmektedir. Bu sayede mobil uygulama cToken bilgisinin doğrulanmış olup olmadığı bilgisini teyit edemez ve kaba kuvvet yöntemler ile cToken bilgisinin üretilmesinin önüne geçilmiştir. Bu mekanizma, mobil uygulamaların gerçek üreticiler tarafından sunulan uygulamalar olduğunu tasdik etmeyi amaçlamıştır.

3.5. ÇALIŞMA YÖNTEMİ

Sayaç tabanlı güvenlik mekanizması cTrust, mobil uygulamalar ile API servisleri arasındaki haberleşmeyi sadece tasdik edilmiş gerçek uygulamalar aracılığı ile gerçekleştirmeyi amaçlayan bir çalışma prensibini takip etmektedir. Mobil uygulama tasdik süreçlerini gerçekleştiren mekanizmaya ait çalışma yöntemi Şekil 3.5.1'de verilmiştir.



Şekil 3.5.1. Sayaç Tabanlı Güvenlik Mekanizması - cTrust

Mekanizmanın çalışma prensibi:

1. Mobil SDK ile SDK API arasındaki doğrulama için, mobil SDK gizli anahtar ile şifrelenmiş cihaz bilgilerini içeren bir istek oluşturur ve bu isteği SDK API doğrulama servisine gönderir.

2. SDK API doğrulama servisi, gelen isteği alır ve gizli anahtar ile çözerek cihaz bilgilerini elde eder. Bu bilgiler uygulama kimliğini ve cihazın güvenilirliğini doğrulamak için kullanılır.
3. Doğrulama sonrasında, SDK API servisi, gizli anahtar ile şifrelenmiş zaman bilgisini mobil uygulamaya yanıt olarak gönderir.
4. Mobil uygulama, gelen zaman bilgisini gizli anahtar ile çözer. SDK içerisine yerleştirilen rastgele değer ile iletilen zaman bilgisi baz alınarak sayaç işlemi başlatılır.
5. Mobil uygulama, bu adımlardan sonra uygulama API servislerine erişmeden önce cToken bilgisi üretmesi gerekmektedir. Verify servisi, sayaç ve paket bilgileri kullanılarak oluşturulan sToken bilgisini iletir ve SDK API üzerinden bir kez kullanılan bir değer (nonce) bilgisi üretilir. Elde edilen nonce bilgisi ve sayaç bilgileri ile sToken oluşturulur. sToken bilgisi, cToken üretilmesi için SDK API servisine gönderilir.
6. SDK API servisi, mobil uygulamadan iletilen nonce değerini kontrol eder. Değer geçerli ise sayaç bilgisi sunucudaki sayaç bilgisi ile karşılaştırılır. Bu karşılaştırma sonucunda, sayaç bilgisi doğrulandığında geçerli bir cToken üretilir. Doğrulanmadığı durumda ise geçersiz bir cToken üretilerek mobil uygulamaya iletilir.
7. Üretilen cToken bilgisi mobil uygulamanın erişmek istediği API servis isteği içerisine eklenerek API servise gönderilir.
8. API servisine iletilen cToken bilgisi için ilk önce zaman geçerlilik kontrolü yapılır ve geçerli bir cToken ise gizli anahtar ile çözülerek geçerli bir cToken olup olmadığı kontrolü gerçekleştirilir.
9. Geçerli bir cToken olması durumunda mobil uygulamaya güvenilir ve ilgili servis görevini yerine getirir.
10. Geçersiz bir cToken olması durumunda API servis ile erişimi kesilir.

Bu mekanizma, mobil uygulama ile API servisi arasındaki güvenliği artırmak amacıyla gizli anahtar kullanımını ve tek kullanımlık erişim belirteci (cToken) kullanımını içermektedir [74]. Ayrıca, cihaz bilgileri ve sayaç bilgisi gibi unsurların şifrelenmesi, veri güvenliğini sağlamak için kritik bir önem taşır. Parametrelerin şifrelenmesi, iletişim sırasında hassas bilgilerin korunmasına katkıda bulunur. Herhangi bir doğrulama hatası

olması durumunda API servisinin otomatik olarak kullanıma kapatılması, güvenlik önlemlerini etkinleştirerek yetkisiz erişimi engeller.

Bu durum, sistemdeki hatalı bir durumu algılamak ve hızlı bir şekilde müdahale etmek için etkili bir güvenlik mekanizması sağlar. Bu sayede, potansiyel güvenlik tehditlerine karşı önleyici bir yaklaşım benimsenmiş olur ve sistem güvenliği güçlendirilir.

3.5.1. SDK API Mekanizması

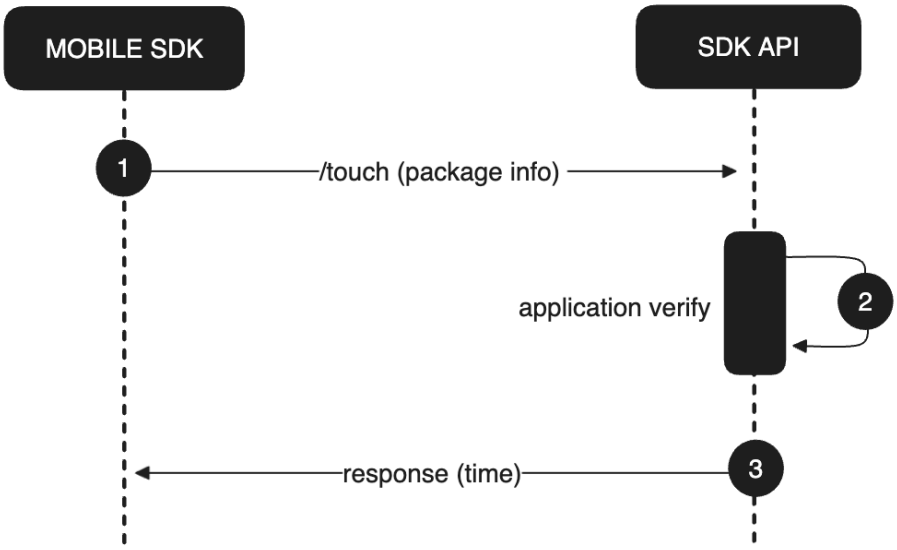
Mobil SDK, Çizelge 3.5.1'de belirtilen mobil uygulama bilgilerini gizli anahtar ile şifreleyerek oluşturduğu isteği SDK API doğrulama servisine gönderir.

Çizelge 3.5.1. Mobil Uygulama Bilgileri

Parametre Numarası	Parametre	Açıklama
1	PackageName	Mobil uygulama paket ismi.
2	Signature	Mobil uygulama bütünlüğünü içeren imza bilgisi.
3	CreatedDateTime	Mobil uygulama oluşturulma zamanı.
4	RandomValue	Mobil uygulama için oluşturulan rastgele değer.

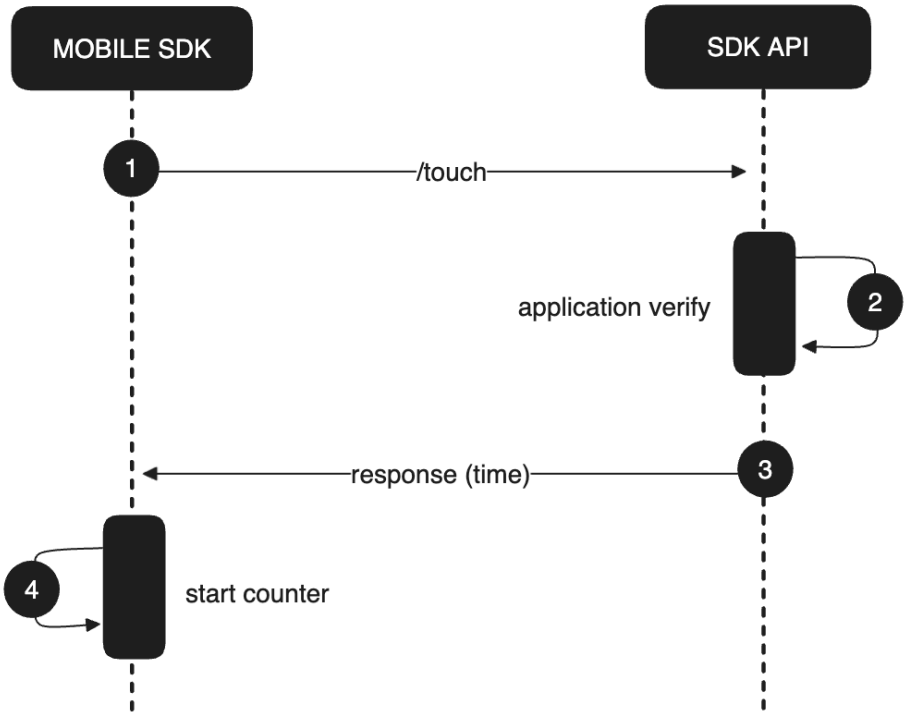
SDK API doğrulama servisi, mobil SDK'dan gelen isteği alır, gizli anahtar kullanarak cihaz bilgilerini çözer ve bu bilgilerle uygulama kimliğini veri tabanı üzerinden karşılaştırır.

Bu sayede cihaz güvenilirliğini kontrol eder. Doğrulama başarılı olduğu durumda, SDK API servisi gizli anahtar ile şifrelenmiş zaman bilgisini Şekil 3.5.2'de gösterildiği üzere mobil uygulama içerisindeki SDK'ya iletir. Zaman bilgisi, sayaç bilgisinin tüm uygulamalar için eş zamanlı olarak doğru şekilde başlatılması için iletilmektedir.



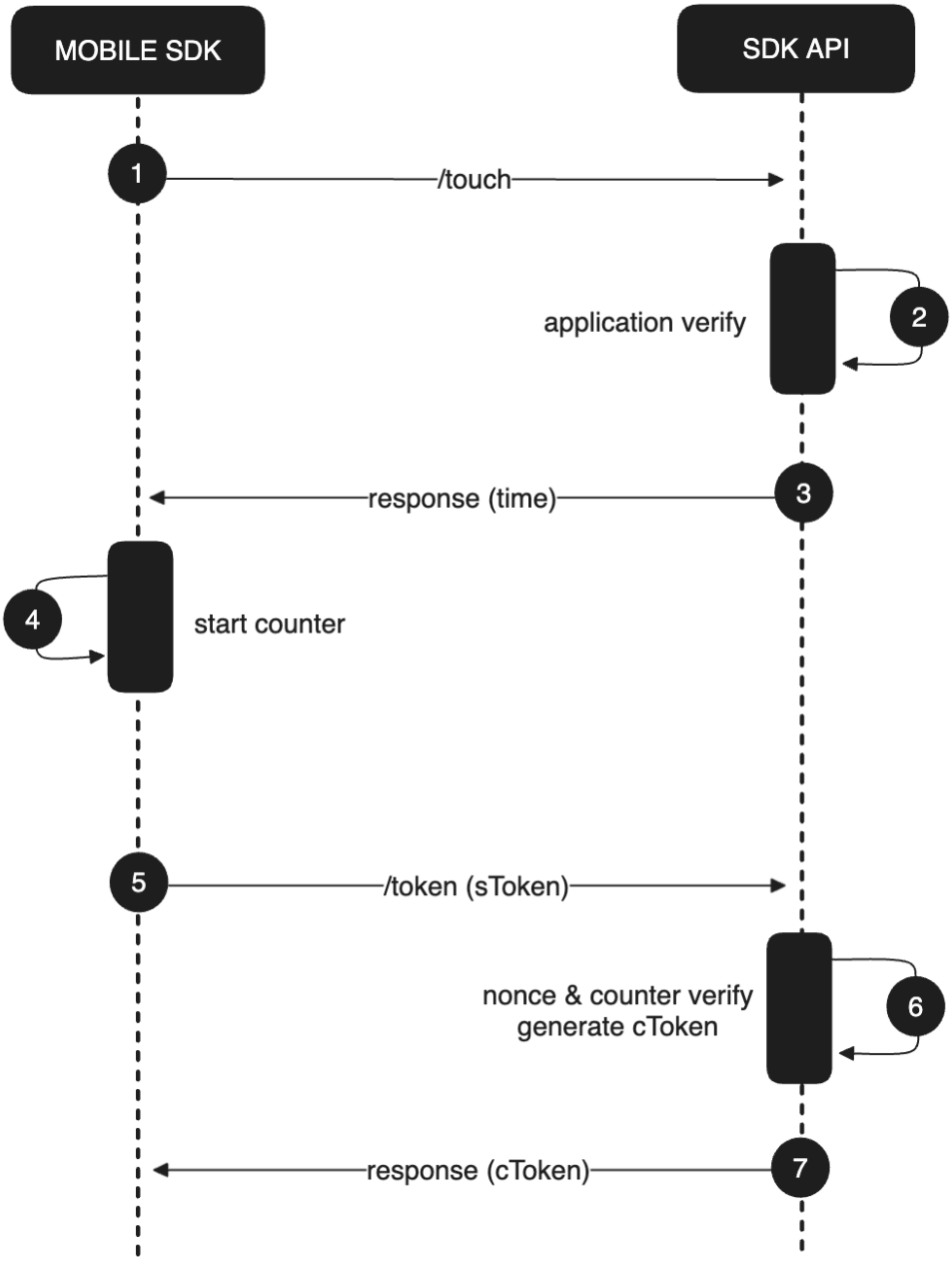
Şekil 3.5.2. Gizli Anahtar ile Şifrelenmiş Zaman Bilgisinin İletilmesi

Şekil 3.5.2'de gösterildiği üzere SDK API tarafından iletilen zaman bilgisi mobil uygulama içerisine entegre edilen SDK yapısına şifrelenmiş olarak iletilir. İletilen zaman bilgisi SDK üzerinden gizli anahtar bilgisi ile çözülmektedir.



Şekil 3.5.3. Mobil SDK Tarafında Yönetilen Sayaç Yapısı

Her bir mobil uygulama için üretilen rastgele sayısal bir değer ve zaman bilgisi ile sayaç mekanizması başlatılmaktadır. Şekil 3.5.3'de sayaç mekanizmasının başlatılmasına ait akış verilmiştir.



Şekil 3.5.4. Mobil SDK Üzerinden cToken Bilgisinin Elde Edilmesi

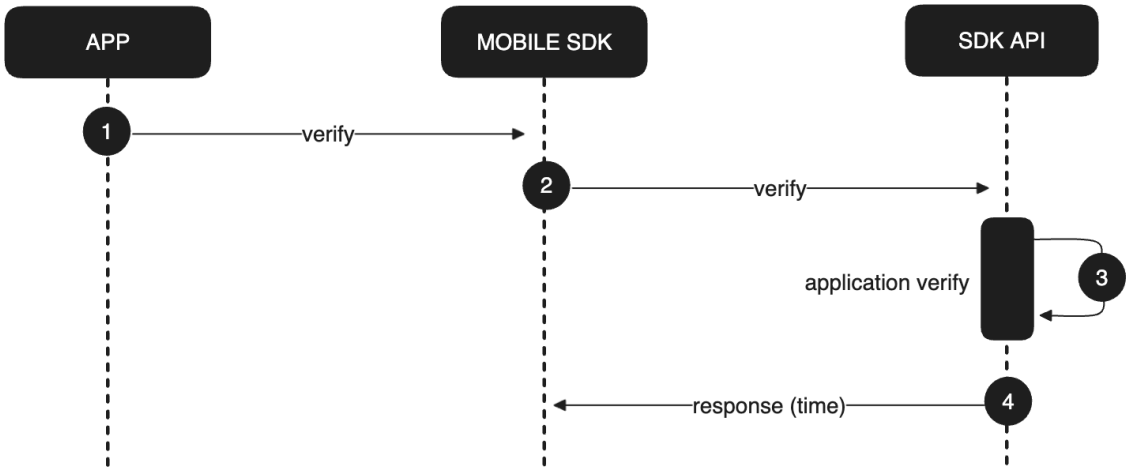
Mobil SDK, SDK API üzerinden bir kez kullanılan "nonce" değeri üretilir. Nonce değer ve sayaç bilgisi cToken bilgisinin üretilmesi için SDK API servisine gönderilmektedir. Mobil SDK üzerinden cToken üretme sürecinde, gecikme zamanının hesaplanması, zaman bilgisinin istek içeriğine eklenmesiyle sağlanır. Bu işlem, isteğin SDK API tarafına ulaştığı andaki zaman bilgisi ile istek aracılığı ile iletilen zaman bilgisi arasındaki farkın hesaplanmasıyla elde edilir. Gecikme zamanı, sayaç bilgisinin doğrulanma aşamasında dikkate alınarak, sayaç doğrulama sürecinde dikkate alınarak doğrulama işlemi gerçekleştirilir. Bu yaklaşım, doğrulama sürecinin, iletişim gecikmelerini göz

önünde bulundurarak daha güvenilir bir şekilde gerçekleştirilmesini sağlar. SDK API servisi, mobil SDK üzerinden iletilen nonce değerini kontrol eder. Nonce değer geçerli ise sayaç bilgisi sunucudaki sayaç bilgisi ile karşılaştırılır.

Sayaç bilgisinin doğrulanması ile geçerli bir cToken üretilir. Üretilen cToken bilgisinin elde edilmesine ait akış Şekil 3.5.4'de verilmiştir. Doğrulamanın gerçekleşemediği durumda ise, geçersiz bir cToken üretilerek mobil SDK'ya iletilir. Mobil SDK bu bilgiyi uygulama API servisi ile haberleşmede kullanmaktadır.

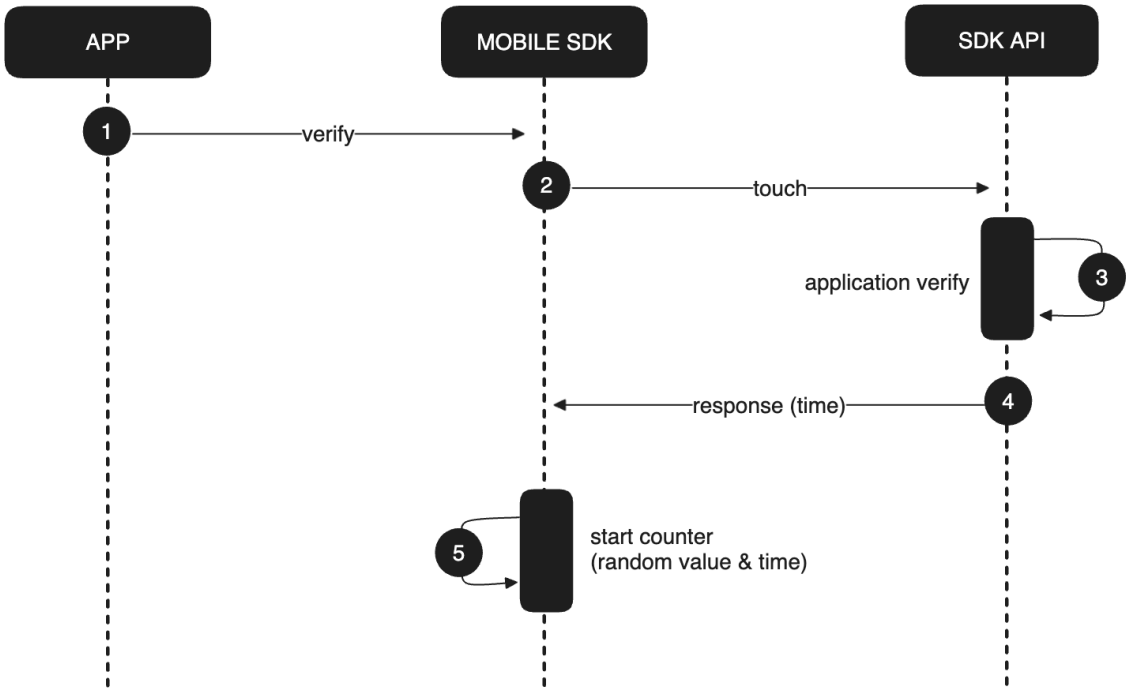
3.5.2. Mobil SDK Mekanizması

Mobil uygulamalar için sayaç mekanizmanın görevlerini yerine getiren bir SDK tasarlanmıştır. Mobil SDK, SDK API ile haberleşerek uygulama doğrulama ve API servisleri ile güvenli haberleşmede kullanılacak cToken bilgisini oluşturma işlemlerini gerçekleştirmektedir. Mobil uygulama Şekil 3.5.5'de gösterildiği gibi, uygulama doğrulama işlemini başlatır. Doğrulama işlemi tamamlandığında, sonuç mobil SDK'ya SDK API üzerinden iletilir.



Şekil 3.5.5. Mobil SDK Üzerinden Doğrulama Akışının Başlatılması

SDK API'dan gelen zaman bilgisi Mobil SDK'ya iletilir. Zaman bilgisi ve her bir uygulama için rastgele üretilen değer ile Şekil 3.5.6'da ifade edildiği üzere sayaç mekanizması başlatılır.

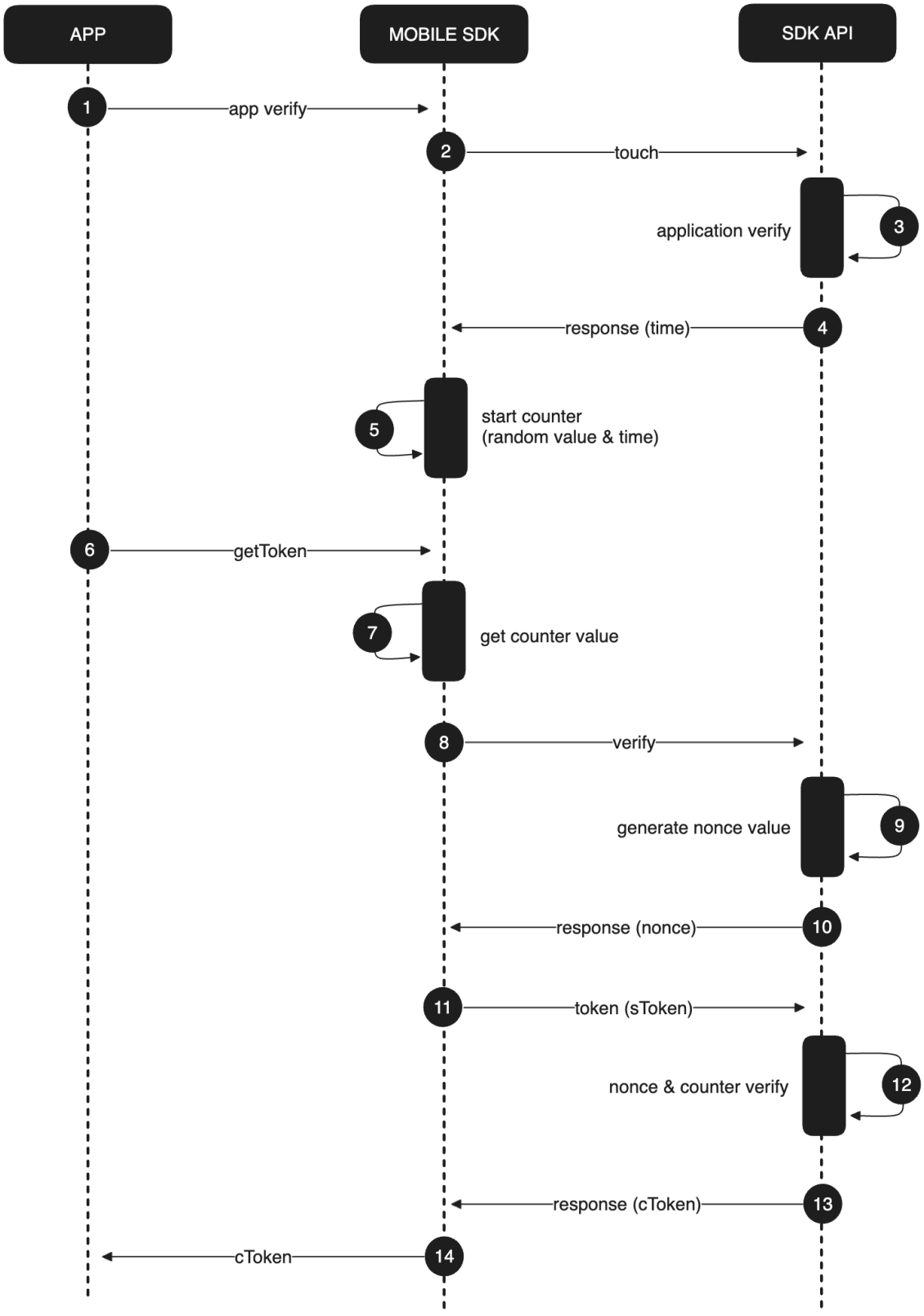


Şekil 3.5.6. Zaman Bilgisinin Mobil SDK İçerisindeki Counter Yapısına İletilmesi

Token (cToken) üretilmek istendiği durumda Şekil 3.5.7'de gösterildiği üzere mobil uygulama içerisine entegre edilen Mobile SDK, doğrulama sürecini başlatır. Bu süreç, uygulamanın doğrulama işlemlerini tamamlamasını sağlar ve ardından zaman bilgisini alarak sayaç başlatılmaktadır.

Mobil SDK, o anki sayaç bilgisini alır ve bu bilgiyi SDK API aracılığıyla sunucuya ileterek nonce değeri üretilir. Ardından, bu sayaç ve nonce değeri ile birlikte SDK API üzerinden token oluşturma süreci başlatılır. Eğer doğrulama başarılı bir şekilde gerçekleştirilirse, geçerli bir cToken üretilir. Ancak, doğrulama işlemi başarısız olursa, geçersiz bir cToken üretilmektedir.

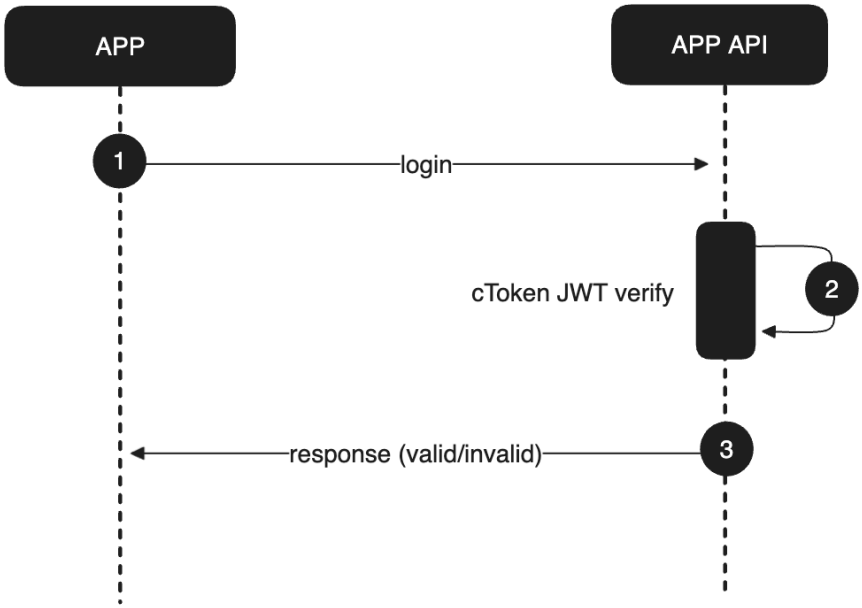
Bu süreç, iletişim güvenliğini sağlamak için kritik öneme sahiptir ve uygulamanın güvenliğini teminat altına alır. Her adımda doğru ve güvenli bir şekilde hareket edilmesi, kullanıcı verilerinin korunmasını ve yetkisiz erişimlerin engellenmesini sağlamaktadır.



Şekil 3.5.7. Nonce ve Counter Bilgisinin İletilmesi ve cToken Bilgisinin Üretilmesi

3.5.3. Uygulama API Mekanizması

Mobil uygulama, uygulama API servisleri ile haberleşmede kullanılmak üzere cToken bilgisini iletir. Mobil uygulama üzerinden iletilen cToken bilgisi sadece uygulama API tarafında bulunan gizli anahtar bilgisi ile çözülerek geçerlilik kontrolü yapılır. Şekil 3.5.8'de cToken bilgisinin uygulama API tarafında doğrulanma akışı verilmiştir. Üretilen cToken bilgisinin geçerlilik süresi SDK API tarafında ayarlanabilmektedir. Token (cToken) bilgisi için geçerlilik süresi anahtar bilgisinin yalnızca bir kez kullanılmasını sağlamaktadır.



Şekil 3.5.8. Uygulama API Haberleşmesinde cToken Bilgisinin Kullanılması

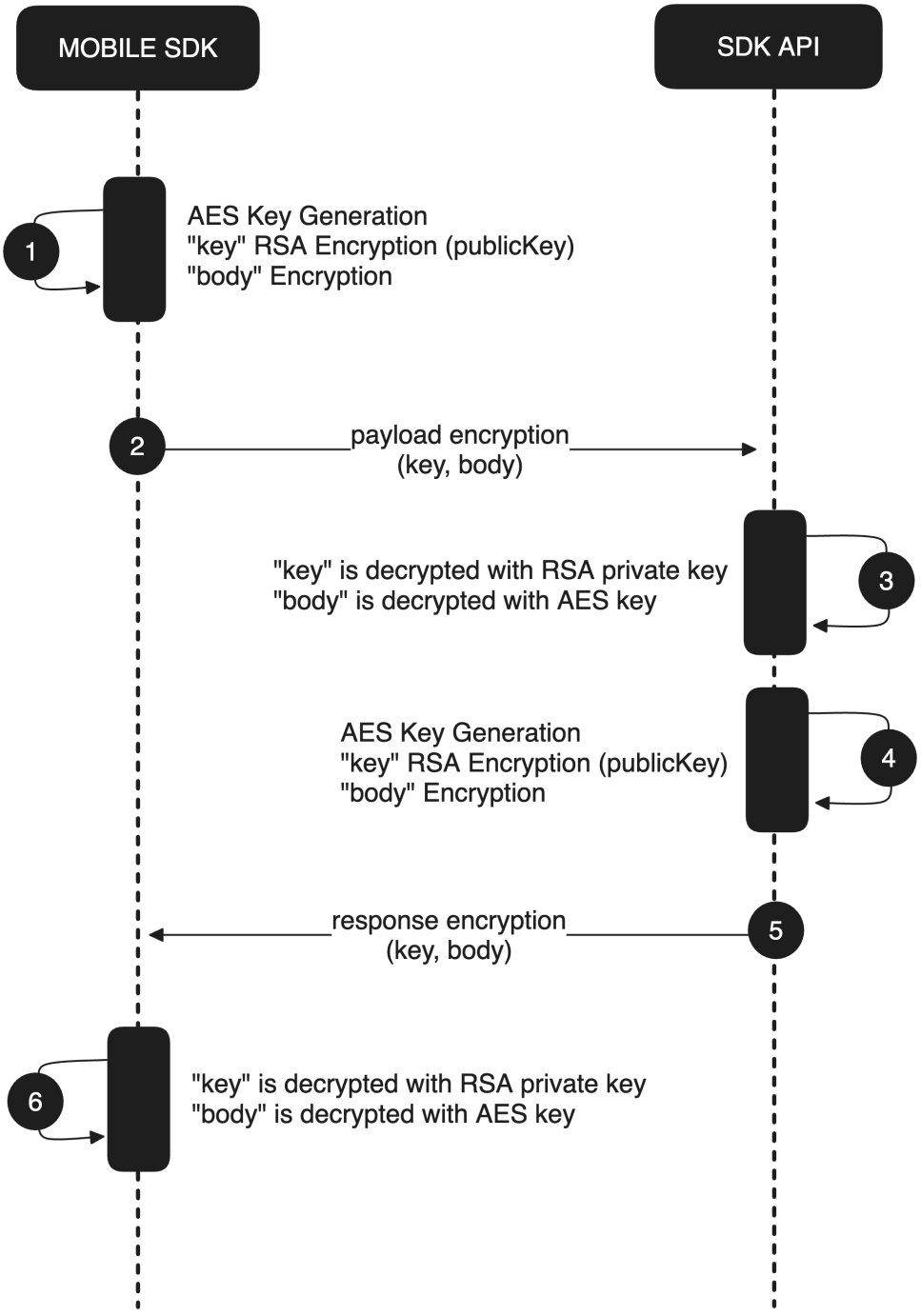
Şekil 3.5.8'da ifade edildiği üzere oluşturulan cToken bilgisi, uygulama API ile iletişimde kritik bir parametredir ve doğru kaynaklarla bilgi paylaşımında önemli bir rol oynamaktadır. Veri tabanındaki bilgilerin hizmete sunulması sırasında, yalnızca doğrulanmış kaynaklarla iletişim kurmak, veri güvenliğini sağlamak açısından hayati öneme sahiptir. Doğrulanmış kaynaklar, güvenli bir iletişim ortamı oluşturarak bu sürecin güvenilirliğini artırır.

3.6. MEKANİZMA GÜVENLİĞİ

Mobil SDK ve SDK API arasındaki iletişimdeki veri güvenliği, simetrik anahtarlı şifreleme algoritması olan AES ve asimetrik anahtarlı şifreleme algoritması olan RSA kombinasyonuyla sağlanmaktadır [75]. AES, iletişim sırasında verilerin güvenliğini

sağlamak için kullanılan simetrik anahtarlı şifreleme algoritmasıdır [76]. Şifreleme ve çözme işlemlerinde aynı anahtar kullanılır, bu nedenle güvenli bir şekilde anahtarın paylaşılması gerekmektedir. Bu gereksinimi karşılamak için her iletişim esnasında yeni bir AES anahtarı üretilir. Üretilen AES (256 bit) anahtarları, RSA algoritması kullanılarak güvenli bir şekilde karşı tarafa iletilir. RSA, açık ve özel anahtar çiftlerine dayalı bir asimetrik anahtarlı şifreleme algoritmasıdır [77]. AES anahtarları, RSA ile şifrelenerek iletilir ve sadece alıcı tarafın özel anahtarı ile çözülebilir. Bu sayede anahtarların güvenliği sağlanır ve veri iletişimi güvence altına alınır.

Bu yaklaşım, güvenilir bir iletişim sağlamak için simetrik ve asimetrik şifreleme yöntemlerinin avantajlarını birleştirir. AES, verilerin hızlı ve etkin bir şekilde şifrelenmesini sağlar, RSA ise anahtarların güvenli bir şekilde paylaşılmasını sağlar. Sonuç olarak, Mobil SDK ve SDK API arasındaki bilgi alışverişi, AES ve RSA algoritmalarının birleşimiyle güvenli bir şekilde gerçekleştirilir. Sayaç tabanlı güvenlik mekanizması cTrust, sürekli değişen bir doğrulama süreci sunarak güvenli bir iletişim ortamında tek kullanımlık doğrulama imkânı sağlar. Bu süreçte, her iletişimde yeni bir AES anahtarı üretilir ve bu anahtarlar RSA algoritması kullanılarak güvenli bir şekilde karşı tarafa iletilir. Bu sayede, iletişimdeki güvenlik sağlanır. Sayaç bilgisi, mobil uygulama tarafından hesaplanır ve AES algoritması kullanılarak şifrelenir. Şifrelenmiş sayaç bilgisi, SDK API servisi tarafından şifreli bir şekilde alınır ve alıcının özel anahtarı kullanılarak çözülür. Böylece, alıcı sayaç bilgisini doğrular ve iletişim sürecinde güvenliği sağlamış olur. Bu yaklaşım, AES'in hızlı ve etkili veri şifreleme yeteneklerini RSA'nın anahtar değişimindeki güvenlik sağlama özellikleriyle birleştirerek güvenli iletişim sağlar. Bu mekanizma, güvenli haberleşme için gizli anahtarları kullanarak verilerin güvenliğini artırır ve tek kullanımlık doğrulama süreciyle ek bir katman sağlar. Mobil SDK ve SDK API arasındaki haberleşmede uygulanan güvenlik önlemlerini içeren akış Şekil 3.6.1'de verilmiştir.



Şekil 3.6.1. Mobil SDK ve SDK API Haberleşmesinde Kullanılan Şifreleme Yapısı

4. UYGULAMA

Önerilen sayaç tabanlı güvenlik mekanizması SDK API, uygulama API ve mobil SDK olmak üzere üç yapıdan oluşmaktadır. Mobil uygulamaya entegre edilen mobil SDK ve doğrulama ve güvenlik anahtarı oluşturulmasını sağlayan SDK API mekanizmalarından meydana gelmektedir. Uygulama kısmında önerilen sayaç mekanizmasının mobil SDK tarafında Android platformu üzerine geliştirmiştir. API kısmında ise Spring Boot ile geliştirme sağlanmıştır. Sayaç tabanlı güvenlik mekanizması cTrust, platform bağımsız Windows, Mac OS X, Linux, HarmonyOS, Android ve iOS gibi platformlar için uygulanabilir bir mekanizmadır.

4.1. API Mekanizmasının Tasarlanması

Bu bölümde, sayaç tabanlı güvenlik mekanizması cTrust'ın API mekanizmalarına ait detaylar verilmiştir.

4.1.1. SDK API Mekanizması

SDK API mekanizmasının tasarımının temel unsurlarına odaklanarak, kullanılan teknolojilerden yazılım mimarisine kadar genel bir bakış sunulmuştur. Bu tasarım, SDK API'nın güçlü temelini oluşturan programlama dilleri, bağımlılıklar, yazılım mimarisi bilgileri verilmiştir. SDK API mekanizmasının, Mobil SDK ile olan etkileşimi sırasında kullanılan protokoller, veri alışverişi yöntemleri ve güvenlik önlemleri detaylı bir şekilde ele alınmıştır.

4.1.1.1. Kullanılan Teknolojiler

SDK API'nın geliştirilmesinde Kotlin programlama dilinin tercih edilmesinin temel avantajlarından biri dilin okunabilir ve anlaşılır bir sözdizimine sahip olmasıdır. Kotlin, kısa ve açık ifadelerle yazılan kodlarıyla geliştiricilere işlerini hızlı ve etkili bir şekilde yürütme imkânı sunar [78]. Bu dilin seçilmesi, SDK API'nın geniş bir kullanıcı kitlesi tarafından rahatça benimsenmesini ve yaygın olarak kullanılabilir olmasını hedeflemektedir. Proje gereksinimlerini karşılamak amacıyla, Spring Boot çerçevesi tercih edilmiştir. Bu seçim, projenin geliştirilmesinde sağladığı avantajlar ve sunulan özellikler göz önüne alınarak yapılmıştır. Spring Boot'un esnek mimarisi, geniş kapsamlı kütüphaneleri ve hızlı uygulama geliştirme süreci, projenin ihtiyaçlarına uygun bir çözüm

sunmaktadır. Ayrıca, Spring Boot'un sunduğu güçlü ve modüler özellikler, projenin başarılı bir şekilde geliştirilmesine ve sürdürülmesine katkı sağlamaktadır. Bu çerçeve, projenin karmaşıklığına uygun bir altyapı sunarak, geliştiricilere etkili bir şekilde çalışma imkânı tanımaktadır [79]. SDK API'da veri tabanı olarak MySQL tercih edilmiştir. MySQL'in açık kaynak olması, ücretsiz olarak kullanılabilmesi, hızlı performansı, geniş kullanıcı tabanı ve topluluk desteği, projenin gereksinimlerine uygunluğunu sağlamıştır. Ayrıca, MySQL'in güvenilirlik, dayanıklılık ve çeşitli uygulama entegrasyonu gibi özellikleri, projenin veri tabanı tercihinin arkasındaki temel nedenleri oluşturmuştur [80]. Bu seçim, projenin etkili bir şekilde geliştirilmesi ve yönetilmesine katkı sağlamaktadır.

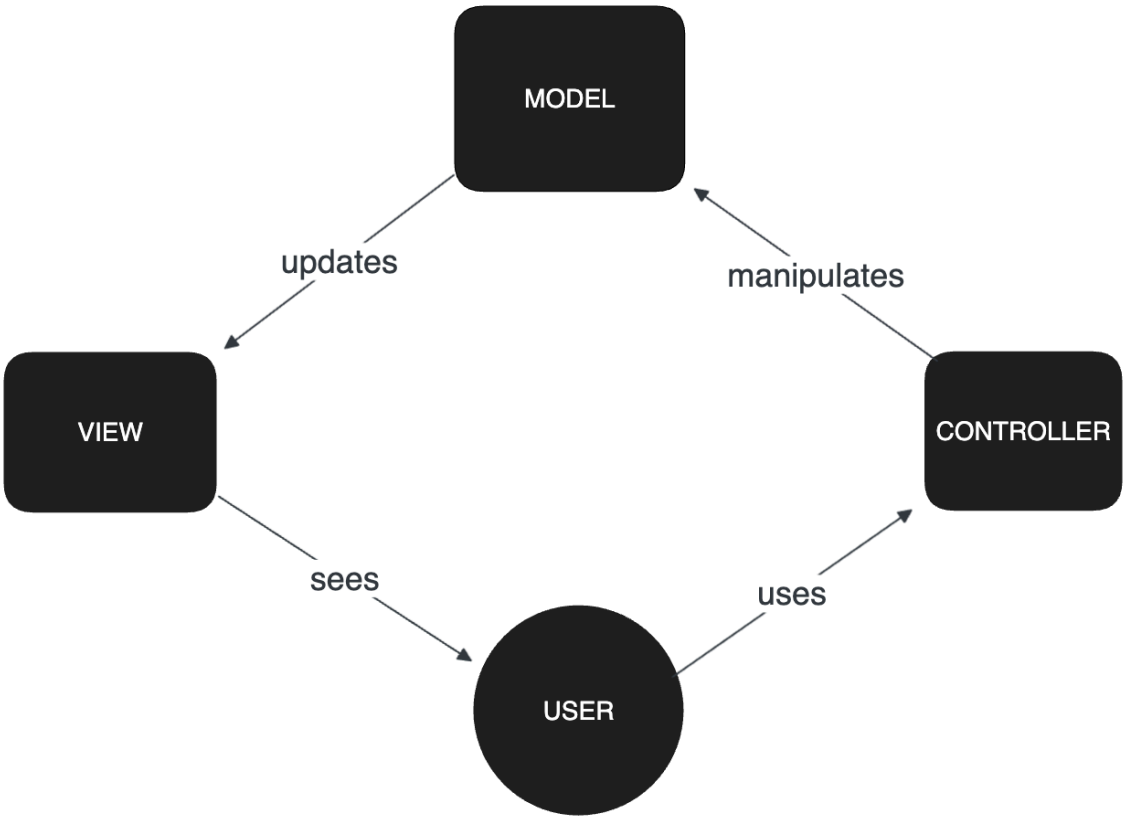
SDK API yapısında kullanılan teknolojiler Çizelge 4.1.1'de verilmiştir. Bağımlılıklar açısından, geliştirme sürecinde simetrik şifreleme algoritması olarak AES algoritması kullanılmıştır. AES anahtar bilgisinin güvenli bir şekilde iletilmesi için RSA asimetrik şifreleme algoritması kullanılmıştır.

Çizelge 4.1.1. SDK API Mekanizmasının Uygulanmasında Kullanılan Teknolojiler

Parametre Numarası	**Parametre**	**Açıklama**
1	Programlama Dili	Kotlin
2	Çerçeve	Spring Boot
3	Veri Tabanı	MySQL

4.1.1.2. Yazılım Mimarisi

SDK API yapısında tasarım deseni olarak Şekil 4.1.1'de ifade edilen Model-View-Controller (MVC) tasarım deseni kullanılmıştır. Bu desen, yazılım bileşenlerini Model, view ve controller olmak üzere üç temel bileşen altında düzenleyerek, projenin modülerliğini, bakımını ve test edilebilirliğini artırmayı amaçlar [81].



Şekil 4.1.1. MVC Tasarım Deseni

Model, SDK API içindeki veri yapıları ve işlemleri temsil eder. Veri manipülasyonları, doğrulama süreçleri ve iş kuralları bu katmanda yer alır. Bu sayede, veri işleme süreçleri bağımsız bir şekilde yönetilebilir.

View, SDK'nın kullanıcı ara yüzü ile doğrudan ilgili değildir; SDK API'nın durumu hakkında bilgi sağlar. Bu katman, SDK API'nın iç çalışma detaylarına karışmadan dış sistemlerle etkileşime geçebilir.

Controller, SDK API'nın temel kontrol mekanizmalarını içerir. Gelen talepleri işler, veri manipülasyonunu yönetir ve uygun sonuçları üretir. Controller, SDK API'nın iç mantığını dış dünya ile etkileşime geçirir.

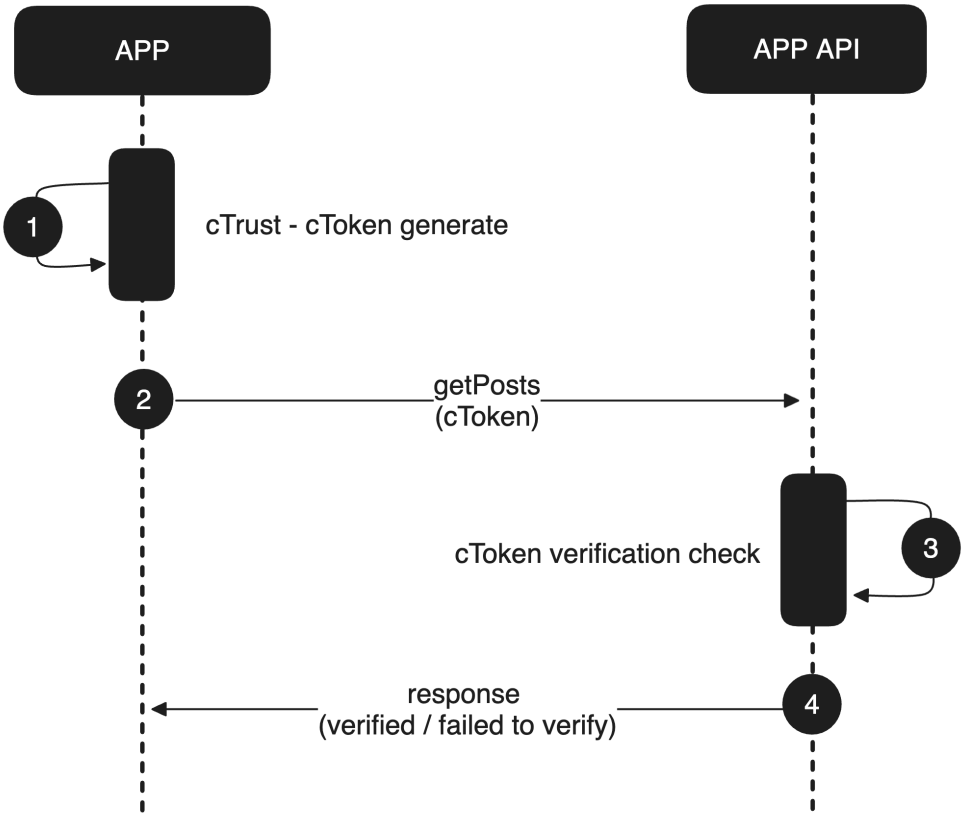
MVC tasarım deseni, SDK API'nın parçalara bölünmüş ve birbirinden bağımsız bileşenlerden oluşan bir yapısını sağlar. Bu, geliştirme sürecinde esneklik ve genişletile bilirlik sunar. MVC'nin benimsenmesi, yazılımın daha anlaşılır olmasını ve ek geliştirmelerin daha sorunsuz bir şekilde entegre edilebilmesi için kullanılmıştır.

4.1.1.3. Güvenlik Önlemleri

SDK API ve mobil SDK arasındaki haberleşme AES ve RSA algoritmaları kullanılarak sağlanmıştır, Token üretme ve doğrulama süreçleri JWT standartlara uygun ve güvenli bir şekilde gerçekleştirilir. Ayrıca, güvenlik önlemleri kapsamında kullanılan nonce değeri, tekrarlanan kullanımları engelleyerek güvenlik standartlarını yükseltir.

4.1.2. Uygulama API Mekanizması

Mobil uygulama üzerinden iletilen cToken bilgisi uygulama tarafından doğrulanması istenen servis üzerinden uygulama API tarafına iletilir. Uygulama API gönderilen cToken bilgisini gizli anahtarı ile çözer ve geçerliliğini kontrol eder. Gönderilen cToken bilgisi geçerli ise uygulama doğrulanır ve servisin görevi yerine getirilir.



Şekil 4.1.2. API Haberleşmesinde cToken Bilgisinin Kullanımı

Şekil 4.1.2'de ifade edildiği üzere geçersiz bir cToken gönderildiğinde uygulamanın doğrulanamadığı bilgisi mobil uygulamaya iletilir.

4.2. Mobil SDK Mekanizmasının Tasarlanması

Mobil Uygulama içerisine entegre edilen, Mobil SDK mekanizmasının detaylı tasarım unsurları ele alınmıştır. Özellikle doğrulama, token oluşturma ve sayaç mekanizmasının aktifleştirilmesi gibi kilit aksiyonlara odaklanarak, Mobil SDK'nın genel yapısı ve işlevselliği ile ilgili bilgiler verilmiştir. Önerilen mekanizma, Android işletim sistemi için uygulanmıştır. Mekanizma diğer platformlar içinde uygulanabilir bir yapıyı içermektedir.

4.2.1. Android İşletim Sistemi

Android, Google tarafından geliştirilen bir mobil işletim sistemidir ve günümüzde milyonlarca cihazda yaygın bir şekilde kullanılmaktadır [82]. Android işletim sistemine ait yazılım yığını şu şekildedir.

Linux Çekirdeği: Android'in temelini oluşturan Linux çekirdeği, donanım kaynaklarını etkin bir şekilde yönetmekten sorumludur. Sürücüleri sağlar, güvenlik önlemlerini uygular ve temel sistem işlevselliğini gerçekleştirir. Linux tabanlı mimari, güçlü çoklu görev desteği ve güvenilir bir çekirdek yapısı sunar [31].

Android Runtime (ART/Dalvik): Android uygulamaları genellikle Kotlin ve Java dilinde yazılır ve bytecode olarak derlenir. ART, bu bytecode doğrudan çalıştıran bir Sanal Makine (Virtual Machine – VM) sistemidir. Dalvik VM, önceki sürümlerde kullanılmıştır, ancak ART, daha yeni sürümlerde performans ve optimizasyon avantajlarıyla standart hale gelmiştir [32].

Çekirdek Kütüphaneler: Android işletim sistemi, temel Java kütüphanelerini içerir. Bu kütüphaneler, veri yapıları işlemleri, dosya işlemleri, ağ operasyonları gibi temel işlevselliği destekler. Yüksek performanslı ve güvenilir kütüphane seti, uygulama geliştiricilerine güçlü bir temel sağlar.

Uygulama Çerçeveleri ve Kütüphaneler: Android, geliştiricilere uygulama oluşturmak için çeşitli çerçeveler ve kütüphaneler sunar. Grafik işleme, veri tabanı işlemleri, ağ bağlantıları gibi temel işlevselliği içeren bu araçlar, hızlı ve etkili uygulama geliştirme süreçlerini destekler.

Sistem Hizmetleri: Arka planda çalışan sistem hizmetleri, işlem yönetimi, hafıza yönetimi, ağ yönetimi gibi temel sistem görevlerini gerçekleştirir. Bu hizmetler, kullanıcının etkileşimde bulunmasa bile arka planda çalışarak sistemin dayanıklılık ve performansını sağlar.

Donanım Soyutlama Katmanı (Hardware Abstraction Level - HAL): Donanım ile yazılım arasında bir ara yüz sağlar. Donanım özelliklerinin, sürücülerin ve diğer donanım bağımlı bileşenlerin soyutlanması, farklı donanım konfigürasyonlarına kolay entegrasyonu mümkün kılar [33].

Uygulama Katmanı: Kullanıcıların doğrudan etkileşimde bulunduğu uygulamalar bu katmanda bulunur. Farklı uygulama türleri, kullanıcı ara yüzleri, oyunlar ve özel uygulamalar, Android'in sunduğu zengin uygulama çerçeveleri üzerine inşa edilir.

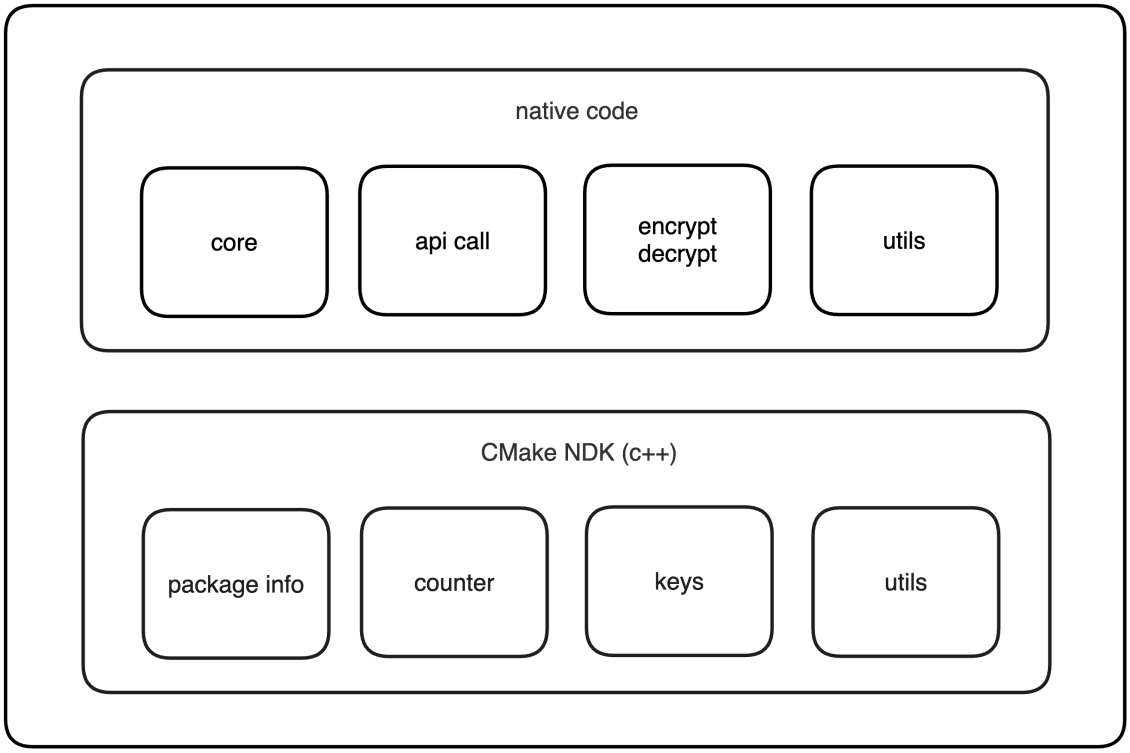
Android işletim sistemine ait yazılım yığını EK 1 içerisindeki Şekil 9.1.1.1'de verilmiştir. Android'in bu temel özellikleri, mobil cihazlarda geniş bir kullanıcı kitlesine hitap etmesini sağlar. Açık kaynak yapısı ve geniş uygulama çerçevesi, geliştiricilere esneklik sağlar ve çeşitli cihazlarda çalışabilen zengin uygulama ekosistemini destekler [83].

4.2.2. Android NDK

Android Yerel Geliştirme Kiti (Native Development Kit - NDK), Android uygulamalarını geliştirmek için kullanılan bir araç setidir. Bu araç seti, geliştiricilere Kotlin ve Java tabanlı Android uygulamalarının yanı sıra C veya C++ gibi düşük seviyeli dillerle yazılmış kodları entegre etme imkânı sunar. Java'nın performans sınırlamalarını aşmak ve özellikle sistem düzeyinde işlemler gerçekleştirmek isteyen geliştiricilere hitap eder. Android NDK'nın temel amacı, grafik işleme, ses işleme, matematiksel operasyonlar gibi hesaplama yoğunluğu yüksek görevlerde daha etkili bir performans elde etmektir. Bu sayede, oyun geliştiricileri, ses işleme uygulamaları ve diğer performans önemli uygulama türleri için daha iyi kontrol ve hız sağlamak mümkün olur. NDK'nın kullanımı, Java tabanlı Android SDK'dan farklıdır ve genellikle özel bir bağlantı katmanı, Java Yerel Arayüzü (Java Native Interface – JNI) kullanılarak gerçekleştirilir. Bu sayede, Java ve C/C++ kodları birbirine entegre edilebilir. Ancak, NDK kullanmak, genellikle performans avantajlarına rağmen, geliştirme sürecini karmaşıklaştırabilir ve bakım maliyetini artırabilir. Bu nedenle, NDK genellikle özel durumlar için tercih edilir ve çoğu Android uygulaması Java veya Kotlin gibi yüksek seviyeli dillerle geliştirilmeye devam eder. SDK mekanizması içerisinde sayaç mekanizması, anahtar bilgileri gibi aksiyonlar NDK'nın sunmuş olduğu güvenlik özellikleri sebebiyle C++ kullanılarak gerçekleştirilmiştir [84].

4.2.3. Mobil SDK

Sayaç mekanizması, Android işletim sistemi ile uyumlu çalışacak şekilde tasarlanmış bir SDK yapısı haline getirilmiştir. Android uygulamalarında kullanılmak üzere Kotlin dili kullanılarak geliştirilen bir SDK'dır. SDK, anahtar bilgilerini güvenli bir şekilde saklamak ve sayaç mekanizmasını etkili bir şekilde işletmek için Android NDK kullanılarak C++ diliyle geliştirilmiştir. Bu sayede, yüksek performans elde edilirken aynı zamanda güvenlik ve verimlilik de sağlanmaktadır. SDK'nın Android uygulamalarına entegrasyonu kolaydır, bu da geliştiricilere esneklik ve kullanım kolaylığı sunmaktadır.



Şekil 4.2.1. Mobil SDK Yapısı

Şekil 4.2.1'de ifade edildiği üzere mobil SDK'nın temel yapısı, iki ana bloktan oluşmaktadır: Native blok ve C++ kodlarını içeren NDK kısmıdır. Kotlin ile yazılan dört bölüm, SDK'nın ana işlevselliğini sağlamaktadır.

Core alanı, mobil uygulamanın SDK ile etkileşimini düzenleyen bir arayüz sunar. Temel iletişim noktasını oluşturarak, SDK'nın kullanımını kolaylaştırır. API Call alanı, SDK ile SDK API arasındaki iletişimi yöneten bu bölüm, farklı bileşenler arasında veri akışını düzenler. API çağrılarını yönetir ve uygulama içinde veri transferini kolaylaştırır. Encrypt/Decrypt alanı, AES, RSA ve Base64 algoritmalarına dayalı şifreleme ve şifre çözme işlemlerini gerçekleştirir. Güvenli iletişimi destekleyerek verilerin korunmasını

sağlar. Utils alanı, SDK içinde kullanılan loglama gibi yardımcı işlemleri gerçekleştiren bu bölüm, genel kullanım kolaylığına katkıda bulunur.

NDK tarafındaki C++ ile geliştirilen dört bölüm bulunmatakdır. Bunlar; Package info alanı, paket ismi, imza bilgisi gibi paketle ilgili bilgileri içerir. Bu bilgiler, paketin tanımlanması ve güvenliği için önemlidir. Counter, sayaç bilgisini hesaplayan bu bölüm, thread işlemlerini içerir. Keys, iletişimin şifrelenmesinde kullanılan anahtar ve diğer şifreleme fonksiyonları içermektedir. Utils, zaman hesaplamaları ve değişken dönüşümlerini içeren bu bölüm, genel yardımcı işlevsel fonksiyonları içermektedir.

Bu yapı, Kotlin ile yazılan SDK ara yüzünü ve C++ ile yazılan alt yapıyı birleştirerek güvenli bir SDK yapısı oluşturmaktadır.

4.2.3.1. Android Arşivi

Android uygulamasının dağıtım ve yüklenme süreçlerini yönetmek için kullanılan bir dosya türüdür. Android arşiv (Android Archive – AAR) dosyaları, Android Studio gibi geliştirme araçları tarafından kullanılan özel bir arşiv formatındadır. AAR dosyaları genellikle bir Android kütüphanesini içerir. Android uygulama geliştiricileri, bu tür kütüphaneleri projelerine ekleyerek, hazır çözümleri kullanabilir ve kendi uygulamalarını geliştirmek için bu kütüphanelerden yararlanabilirler. AAR dosyaları, bir kütüphanenin kodunu, kaynak dosyalarını ve diğer bağımlılıkları içerir. Bu dosyalar, Android uygulama geliştirme sürecinde modülerlik ve tekrar kullanılabilirlik sağlamak için önemlidir. AAR dosyaları, bir Android projesine kolayca entegre edilebilir ve projenin bağımlılıklarını yönetmek için kullanılır [85].

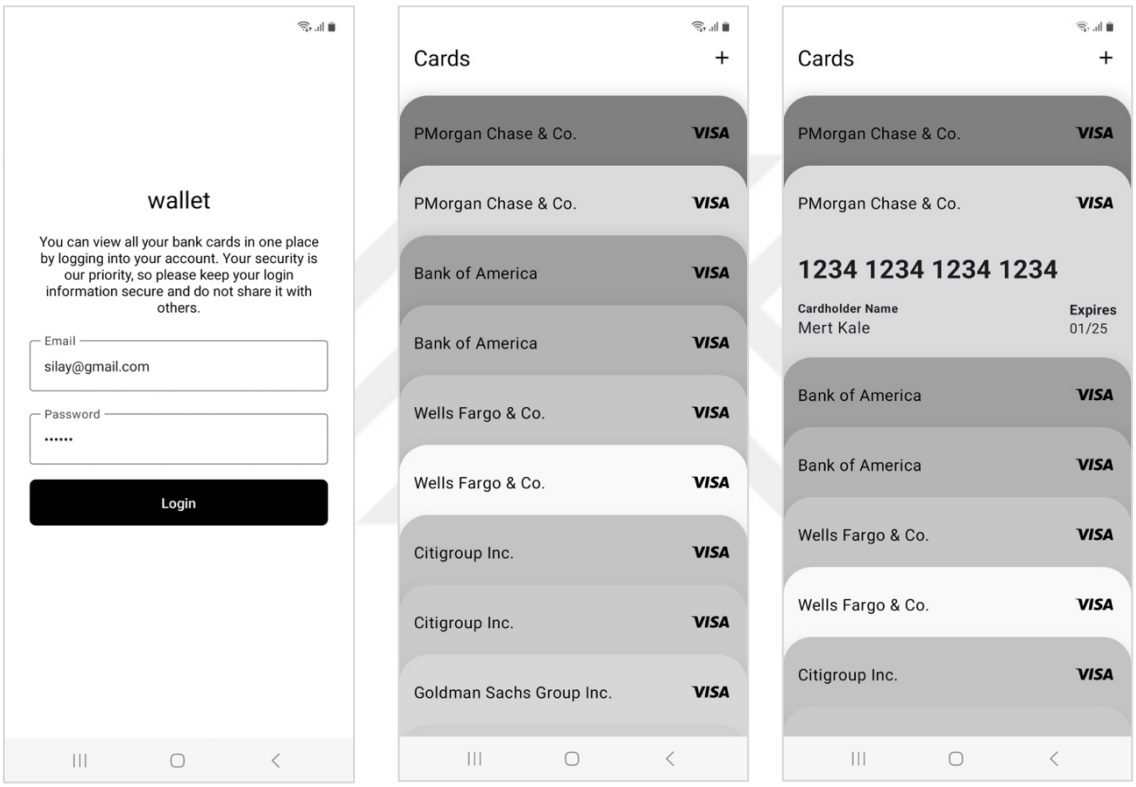
4.2.3.2. Kullanılan Teknolojiler

Mobil SDK yapısı Kotlin ve C++ kullanılarak geliştirilmiştir. Şifreleme için AES ve RSA algoritmaları kullanılmıştır. Mobil SDK dağıtımı için AAR yapısı kullanılmıştır.

4.3. Mobil Uygulama Örneği

Bu bölümde, sayaç tabanlı güvenlik mekanizması cTrust'ın entegre edildiği Wallet mobil uygulamasının geliştirilmesi ve işleyişi incelenmiştir. Wallet mobil uygulaması Android işletim sistemi için geliştirilen, kullanıcıların kart bilgilerini güvenli bir şekilde görüntüleyebildiği bir mobil uygulamadır. Wallet mobil uygulaması ile önerilen sayaç tabanlı güvenlik mekanizması cTrust'ın testleri gerçekleştirilmiştir. Mobil uygulamaya

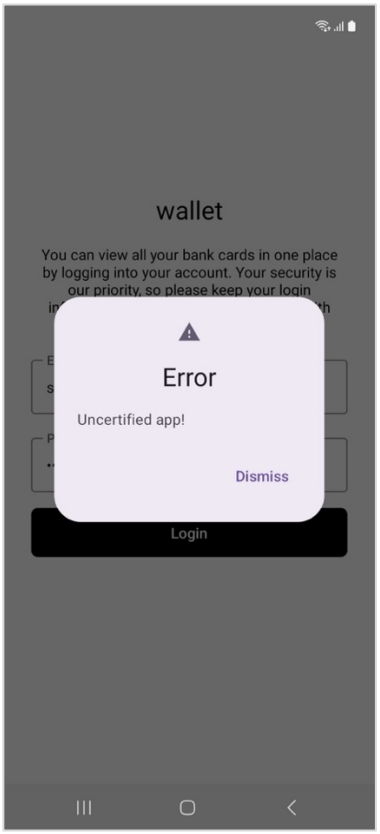
ait ekran görüntüleri Şekil 4.3.1'de verilmiştir. Mobil uygulama üzerinden gerçekleştirilen kullanıcı girişi ve kart listeleme işlemleri, cToken doğrulaması ile sağlanmaktadır. Kullanıcı girişi yapmak isteyen kullanıcılar, e-posta adresi ve şifrelerini girdikten sonra cTrust mobil SDK aracılığıyla cToken bilgisi üretilir. Kullanıcı bilgileri ve üretilen cToken bilgisi, kullanıcı girişi API servisine iletilir. Mobil uygulama tarafından uygulama API servisine iletilen cToken (JWT) bilgisinin geçerlilik süresi kontrol edilir. Geçerlilik süresi devam ediyor ise gizli anahtar ile cToken bilgisi kontrol edilir. Doğrulama işlemi başarılı olduğu durumda servis hizmetini sunmaya devam eder.



a) b) c)

Şekil 4.3.1. Wallet Mobil Uygulaması Arayüzü a) Kullanıcı Girişi b) Kart Listesi c) Kart Detayı

Doğrulamanın başarısız olduğu durumda mobil uygulama ile API arasındaki haberleşme kesilmektedir. Sayaç tabanlı güvenlik mekanizması cTrust'ın entegre edildiği Wallet mobil uygulamasında doğrulamanın başarısız olduğu durumda kullanıcıya gösterilen bilgilendirme uyarısı Şekil 4.3.2'de verilmiştir.

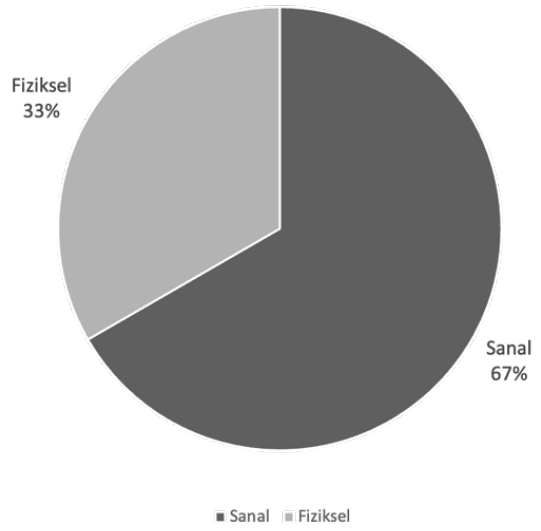


Şekil 4.3.2. Doğrulamanın Başarısız Olduğu Durumda Gösterilen Bilgilendirme Ekranı

Sayaç tabanlı güvenlik mekanizması cTrust ile tasdik edilmemiş uygulamaların API servisleri ile haberleşmesinin önüne geçilmesi sağlanmaktadır.

5. TESTLER

Grafiksel Kullanıcı Arayüz (Graphical User Interface - GUI) testleri bir bilgisayar programının veya uygulamanın grafiksel kullanıcı ara yüzünün doğruluğunu, işlevselliğini ve kullanılabilirliğini değerlendirmek için yapılan testlerdir [86], [87]. Bu testler, kullanıcıların programı etkileşimli olarak kullanırken karşılaşabilecekleri sorunları belirlemek ve gidermek için gerçekleştirilir [88]. Mobil uygulama için kodlanan GUI testleri ile birlikte mobil SDK, SDK API ve uygulama API aktivite kayıtları ve süreçleri incelenmiştir. Testler, mekanizmanın performansını hem fiziksel hem de sanal cihazlarda değerlendirmek amacıyla kurgulanmıştır. Gerçekleştirilen testler ile mekanizmanın gerçek dünya ve sanal ortam koşullarında nasıl tepki verdiğini anlamak ve işlevselliğini doğrulamak için değerlendirme sürecinde kullanılmak üzere çıktılar elde edilmiştir.



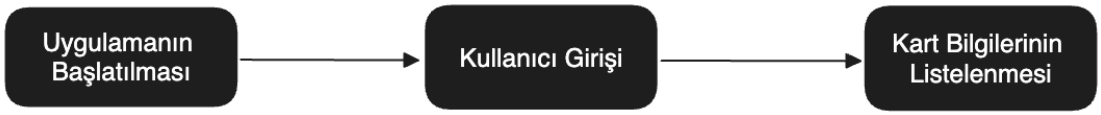
Şekil 5.1. Testlerde Kullanılan Fiziksel ve Sanal Cihaz Oranları

Test senaryolarının gerçekleştirildiği Android işletim sistemine sahip fiziksel ve sanal cihazların birbirine olan oranları Şekil 5.1'de verilmiştir. Fiziksel cihazlardan elde edilen test sonuçları, mekanizmanın kullanıcı deneyimi üzerindeki etkilerini belirleme noktasında önemli bir role sahiptir. Sanal cihazlardaki performans ölçümleri ise, mekanizmanın geniş bir kullanım yelpazesi ve çeşitli senaryolara nasıl adapte olduğunu

değerlendirmede önemli bir perspektif sunmaktadır. Test sonuçları, mekanizmanın güvenilirliği ve etkinliği üzerinde bilimsel bir değerlendirme sunarak, mekanizmanın performansını belirlemede önemli bir rol oynamaktadır. Elde edilen bu veriler, mekanizmanın gerçek dünya ve sanal ortam koşullarındaki işlevselliği hakkında kapsamlı bir iç görü sağlamakla kalmayacak, aynı zamanda mekanizmanın temel hedeflerine ulaşmada elde ettiği başarıları vurgulamaktadır. Test sonuçları, mekanizmanın tasarım ve uygulama süreçlerindeki güçlü yönleri ve potansiyel geliştirmelerin belirlenmesi konusunda önemli bir kaynak teşkil etmektedir.

5.1. TEST SENARYOLARI

GUI testleri ile sanal ve fiziksel cihazlar üzerinde testler gerçekleştirilmiştir. Gerçekleştirilen ara yüz testleri için test senaryoları Şekil 5.1.1'de verilmiştir. Uygulamanın başlatılması, kullanıcı girişi ve kartların listelenmesi senaryoları EK 4'de verilen ara yüz test kodları ile gerçekleştirilmiştir. Bu sayede kullanıcının başarıyla giriş yapıp yapmadığı, kart bilgilerinin uygulama API üzerinden başarıyla getirilip getirilmediği senaryoları test edilmiştir.



Şekil 5.1.1. Mobil Uygulama Ara Yüz Test Akışı

Mobil SDK ile SDK API arasındaki haberleşme güvenliği, uygulama doğrulama akışı, sayaç bilgisinin doğrulanması ve üretilen cToken bilgisinin uygulama API tarafından doğrulanması gibi temel işlevlerin başarı durumlarını ölçmek için Çizelge 5.1.1'de belirtilen test senaryoları dikkate alınmıştır. Bu senaryolar, sayaç tabanlı güvenlik mekanizması cTrust'ın performansını değerlendirmek için seçilmiştir.

Test senaryoları, sayaç mekanizmasının işleyişini ve güvenlik sağlama yeteneğini göstermek amacıyla belirlenmiştir. Bu senaryolar, iletişim sırasında verilerin manipülasyona karşı korunduğunu ve doğru şekilde işlendiğini doğrulamak için seçilmiştir.

Bu senaryoların başarılı bir şekilde gerçekleştirilmesi, sayaç tabanlı güvenlik mekanizması cTrust'ın etkinliğini ve güvenilirliğini göstermektedir.

Çizelge 5.1.1. Mekanizma İçin Belirlenen Test Senaryoları

No.	Test Açıklaması	Parametreler	Beklenen Sonuç	Başarı Durumu
1	Parametre Şifreleme ve Çözme	Gizli Anahtar, Şifrelenen Parametreler	Parametrelerin güvenli bir şekilde şifrelendiği ve çözüldüğü kontrol edilecek.	[Başarılı / Başarısız]
2	Doğrulama Mekanizması	Şifrelenmiş Parametreler, Gizli Anahtar ile Şifrelenmiş Sayaç Bilgisi	Doğrulama mekanizmasının çalıştığı ve güvenli bir şekilde doğrulandığı kontrol edilecek.	[Başarılı / Başarısız]
3	Sayaç Bilgisi İletimi	Şifrelenmiş Sayaç Bilgisi	Sayaç bilgisinin güvenli bir şekilde iletilip alındığı kontrol edilecek.	[Başarılı / Başarısız]
4	Sayaç Bilgisi Eşleşmesi	Gönderilen Sayaç Bilgisi, Sunucuda Bulunan Sayaç Bilgisi	Gönderilen ve sunucuda bulunan sayaç bilgilerinin eşleştiği kontrol edilecek.	[Başarılı / Başarısız]
5	Bilgi Akışı	Doğrulanmış Sayaç Bilgisi	Doğrulanmış sayaç bilgisi ile güvenli bilgi akışının gerçekleştiği kontrol edilecek.	[Başarılı / Başarısız]
6	Bilgi Akışının Kesilmesi	Doğrulanmamış veya Eşleşmeyen Sayaç Bilgisi	Doğrulanmamış veya eşleşmeyen sayaç bilgisi durumunda bilgi akışının engellendiği kontrol edilecek.	[Başarılı / Başarısız]

5.2. SANAL CİHAZ TESTLERİ

Mekanizmanın performansının değerlendirilmesi amacıyla gerçekleştirilen sanal cihaz testlerine odaklanmıştır. Bu testler, farklı tipteki sanal cihazlar üzerinde mekanizmanın işlevselliğini, uyumluluğunu ve performansını değerlendirme amacını taşımaktadır.

Çizelge 5.2.1. Testlerde Kullanılan Sanal Cihazlar

Cihaz No.	Cihaz	Cihaz Tipi	Android API	ABI	RAM
1	Pixel 7	Telefon	32	arm64-v8a	1536 MB
2	Pixel 7 Pro	Telefon	31	arm64-v8a	1536 MB
3	Pixel 6a	Telefon	30	arm64-v8a	1536 MB
4	Pixel 6 Pro	Telefon	29	x86_64	1536 MB
5	Pixel 5	Telefon	28	x86_64	1536 MB
6	Pixel 4	Telefon	32	arm64-v8a	1536 MB
7	Pixel 4a	Telefon	30	armeabi-v7a	1536 MB
8	Pixel 3	Telefon	28	armeabi-v7a	1536 MB
9	Nexus 5	Telefon	32	arm64-v8a	1536 MB
10	Nexus 5	Telefon	30	arm64-v8a	1536 MB
11	Nexus 5	Telefon	29	arm64-v8a	1536 MB
12	Nexus 4	Telefon	33	arm64-v8a	1536 MB
13	Nexus 4	Telefon	32	arm64-v8a	1536 MB
14	Nexus 6	Telefon	34	arm64-v8a	1536 MB
15	Nexus 6P	Telefon	34	arm64-v8a	1024 MB
16	Pixel	Telefon	28	arm64-v8a	1536 MB
17	Pixel 2	Telefon	30	arm64-v8a	1024 MB
18	Pixel 2	Telefon	31	arm64-v8a	1024 MB
19	Nexus 7	Tablet	30	x86_64	1536 MB
20	Nexus 10	Tablet	28	arm64-v8a	1536 MB

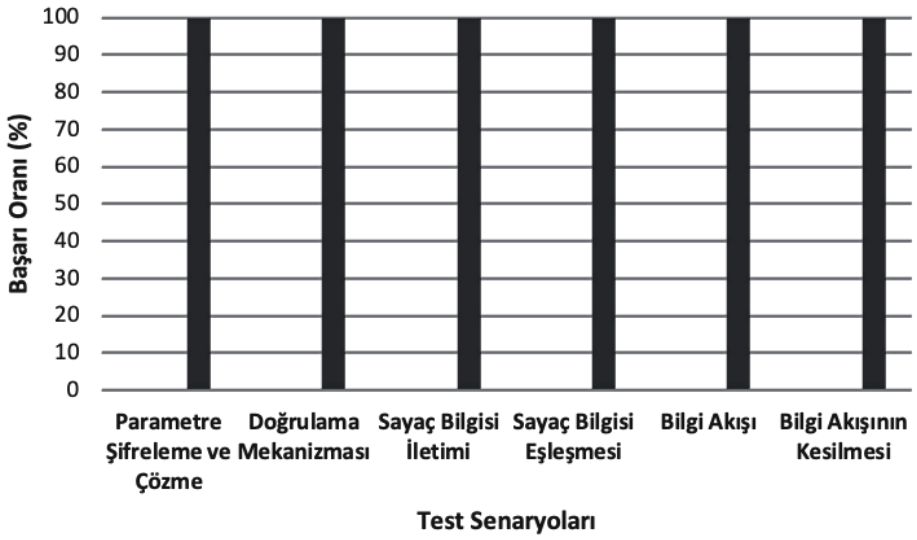
Farklı Uygulama İkili Arabirimi (Application Binary Interface – ABI), Android API ve Rasgele Erişim Belleği (Random Access Memory – RAM) kapasitelerine sahip sanal cihazlarda yapılan testler mekanizmanın geniş bir kullanıcı kitlesi tarafından kullanılabilirliği açısından kritik bir değerlendirme sunmaktadır. Bu bölümde paylaşılan test sonuçları, mekanizmanın çeşitli sanal cihazlarda nasıl performans gösterdiğini ve kullanıcı deneyimi üzerindeki etkilerini anlamak adına perspektif sağlamaktadır. Çizelge 5.2.1'de verilen farklı ABI ve RAM boyutlarına sahip sanal cihazlar üzerinde önerilen mekanizma testleri gerçekleştirilmiştir. Android işletim sisteminin farklı API sürümleri

üzerindeki davranışlarının incelenmesi için Android API sürümleri üzerinde test senaryoları gerçekleştirilmiştir.

Çizelge 5.2.2. Sanal Cihazlar Üzerindeki Test Sonuçları

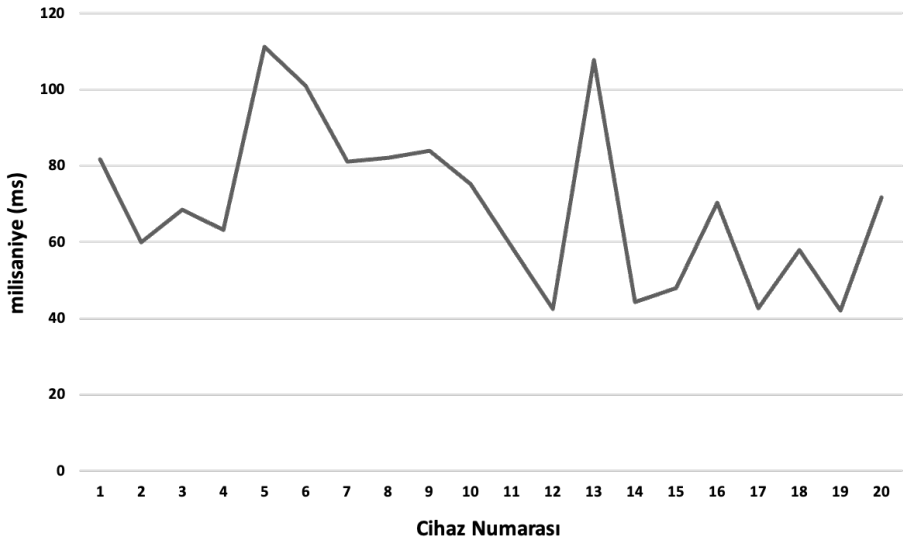
Cihaz No.	Parametre Şifreleme ve Çözme (%)	Doğrulama Mekanizması (%)	Sayaç Bilgisi İletimi (%)	Sayaç Bilgisi Eşleşmesi (%)	Bilgi Akışı (%)	Bilgi Akışı Kesilmesi (%)	Gecikme Zamanı (ms)
1	%100	%100	%100	%100	%100	%100	82 ms
2	%100	%100	%100	%100	%100	%100	60 ms
3	%100	%100	%100	%100	%100	%100	69 ms
4	%100	%100	%100	%100	%100	%100	63 ms
5	%100	%100	%100	%100	%100	%100	111 ms
6	%100	%100	%100	%100	%100	%100	101 ms
7	%100	%100	%100	%100	%100	%100	81 ms
8	%100	%100	%100	%100	%100	%100	82 ms
9	%100	%100	%100	%100	%100	%100	84 ms
10	%100	%100	%100	%100	%100	%100	75 ms
11	%100	%100	%100	%100	%100	%100	59 ms
12	%100	%100	%100	%100	%100	%100	42 ms
13	%100	%100	%100	%100	%100	%100	108 ms
14	%100	%100	%100	%100	%100	%100	44 ms
15	%100	%100	%100	%100	%100	%100	48 ms
16	%100	%100	%100	%100	%100	%100	70 ms
17	%100	%100	%100	%100	%100	%100	43 ms
18	%100	%100	%100	%100	%100	%100	58 ms
19	%100	%100	%100	%100	%100	%100	42 ms
20	%100	%100	%100	%100	%100	%100	72 ms

Farklı Android cihazları, çeşitli Merkezi İşlem Birimi (Central Process Unit – CPU) ve talimat setlerini destekler. Her bir CPU ve talimat seti kombinasyonu, kendi Uygulama İkili Arabirimi ne (ABI) sahiptir. ABI, bir uygulamanın çalıştığı platforma özgü düşük seviyeli kodlama ve derleme bilgilerini içerir. Bu, her cihazın farklı donanım özelliklerine uygun olarak optimize edilmiş uygulamaların çalıştırılabilmesini sağlamaktadır.



Şekil 5.2.1. Sanal Cihaz Başarı Oranları

Çizelge 5.1.1'de mekanizma için belirlenen test senaryoları Çizelge 5.2.1'de verilen sanal cihazlar üzerinde gerçekleştirilmiştir. Sanal cihazların başarı oranları ve mobil SDK ile SDK API arasındaki ortalama gecikme zamanları Çizelge 5.2.2'de verilmiştir. Sanal cihazların, Çizelge 5.1.1'de belirtilen test senaryolarına ait yüzdelik başarı oranları Şekil 5.2.1'de verilmiştir.



Şekil 5.2.2. Sanal Cihaz Gecikme Zamanları

Yerel ağ üzerinde çalıştırılan API servisleri üzerinde yapılan testler sonucunda, her bir sanal cihaz için 100 kez çalıştırılan test senaryolarının sonuçları değerlendirilmiştir. Test senaryolarının neticesinde sanal cihazlar için elde edilen ortalama gecikme zamanları Şekil 5.2.2'de verilmiştir.

5.3. FİZİKSEL CİHAZ TESTLERİ

Bu bölümde, mekanizma performansının değerlendirilmesi için fiziksel cihaz testlerine odaklanılmıştır. Fiziksel cihazlar, mekanizmanın günlük kullanım senaryolarında nasıl davrandığını anlamak adına hayati bir öneme sahiptir. Bu testler, mekanizma deneyimini optimize etme, güvenilirlik açısından değerlendirme ve fiziksel cihazlarda potansiyel performans geliştirmelerini belirleme amacını taşımaktadır.

Gerçek dünya kullanım koşullarında, sayaç tabanlı güvenlik mekanizması cTrust'ın pratikteki performansını ve karşılaştığı zorlukları anlamak için gerçekleştirilen fiziksel cihaz testlerinin sonuçları incelenmiştir. Bu testler, mekanizmanın etkinliğini değerlendirmenin yanı sıra, mekanizmanın gerçek dünya senaryolarında nasıl performans gösterdiğini anlamak için de önemlidir.

Çizelge 5.3.1. Testlerde Kullanılan Fiziksel Cihazlar

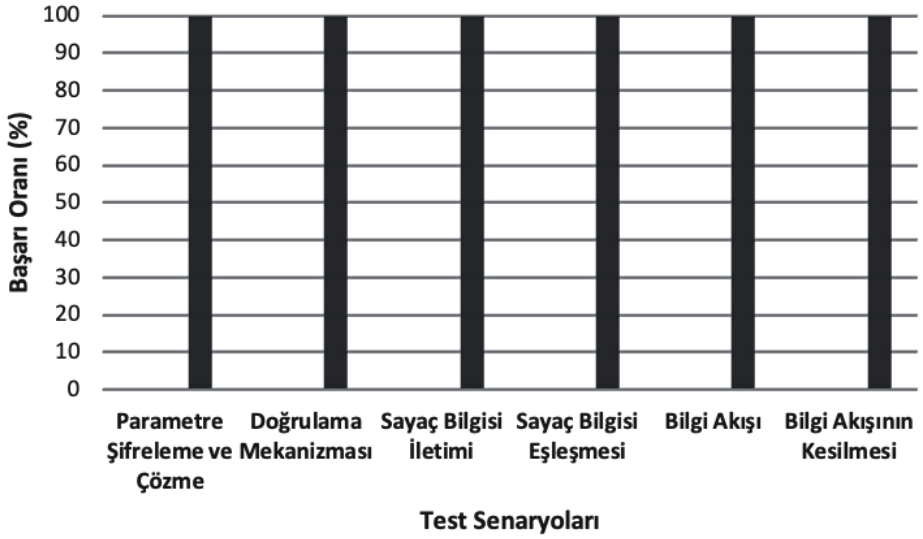
Cihaz No.	Cihaz Modeli	Cihaz Tipi	Android API	RAM	Wi-Fi	Ağ Desteği
1	HONOR 9X	Telefon	28	4 GB	Wi-Fi 5	4.5G
2	Samsung A53	Telefon	31	8 GB	Wi-Fi 5	5G
3	Asus ZenFone 5 Laser	Telefon	27	4 GB	Wi-Fi 4	4.5G
4	Samsung A52	Telefon	30	8 GB	Wi-Fi 5	4.5G
5	Samsung A54	Telefon	33	8 GB	Wi-Fi 6	5G
6	Xiaomi Mi 11 Lite	Telefon	30	8 GB	Wi-Fi 6	5G
7	Xiaomi Redmi Note 13	Telefon	33	8 GB	Wi-Fi 5	4.5G
8	Huawei P20 Pro	Telefon	27	8 GB	Wi-Fi 5	4.5G
9	Xiaomi 12 Lite	Telefon	32	8 GB	Wi-Fi 6	5G
10	Samsung Galaxy S23 FE	Telefon	33	8 GB	Wi-Fi 6E	5G

Çizelge 5.3.1'de verilen fiziksel cihazlar üzerinde Çizelge 5.1.1'de verilen test senaryolarının başarı oranları EK 4'de verilen ara yüz test kodları kullanılarak gerçekleştirilmiştir.

Çizelge 5.3.2. Fiziksel Cihazlar Üzerindeki Test Sonuçları

Cihaz No.	Parametre Şifreleme ve Çözme (%)	Doğrulama Mekanizması (%)	Sayaç Bilgisi İletimi (%)	Sayaç Bilgisi Eşleşmesi (%)	Bilgi Akışı (%)	Bilgi Akışının Kesilmesi (%)	Gecikme Zamanı (ms)
1	%100	%100	%100	%100	%100	%100	272 ms
2	%100	%100	%100	%100	%100	%100	287 ms
3	%100	%100	%100	%100	%100	%100	290 ms
4	%100	%100	%100	%100	%100	%100	349 ms
5	%100	%100	%100	%100	%100	%100	291 ms
6	%100	%100	%100	%100	%100	%100	291 ms
7	%100	%100	%100	%100	%100	%100	283 ms
8	%100	%100	%100	%100	%100	%100	305 ms
9	%100	%100	%100	%100	%100	%100	312 ms
10	%100	%100	%100	%100	%100	%100	288 ms

Fiziksel cihazlar üzerinde gerçekleştirilen test senaryolarına ait başarı oranları ve mobil SDK ile SDK API arasındaki ortalama gecikme zamanları Çizelge 5.3.2'de verilmiştir.



Şekil 5.3.1. Fiziksel Cihaz Başarı Oranları

Fiziksel cihazların, Çizelge 5.1.1'de belirtilen test senaryolarına ait yüzdelik başarı oranları Şekil 5.3.1'de verilmiştir.

6. BULGULAR VE TARTIŞMA

Elde edilen bulgular, önerilen mekanizmanın sanal ve fiziksel cihazlar üzerinde başarılı bir şekilde test senaryolarını %100 başarı oranıyla tamamladığını göstermektedir. Her bir sanal cihaz için 100 kez test senaryosu çalıştırılmış ve API servisleri ile haberleşme sırasında yaşanan ortalama zaman gecikmeleri verilmiştir. Bu bulgular, mekanizmanın hem sanal hem de fiziksel cihazlar üzerinde güvenilir bir performans sergilediğini ortaya koymaktadır. Sanal cihaz test sonuçları, mekanizmanın farklı Android API sürümleri ve bellek kapasiteleriyle uyumlu olduğunu göstermektedir. Fiziksel cihaz test sonuçları ise mekanizmanın gerçek dünya kullanım koşullarında güvenilir ve tutarlı bir performans gösterdiğini doğrulamaktadır. Bu başarıların yanı sıra, mekanizmanın ileride daha geniş bir cihaz yelpazesi ve farklı kullanım senaryolarında test edilmesinin önemi vurgulanmaktadır. Bu şekilde, mekanizmanın potansiyel zorlukları ve performans sorunları daha detaylı bir şekilde ele alınabilir ve çözümlenebilir.

6.1. SAYAÇ MEKANİZMASI VE GÜVENLİK ETKİNLİĞİ

Sanal ve fiziksel cihazlar üzerinde yapılan testler sonucunda elde edilen bulgular, cTrust'ın mobil uygulama ve API servisi arasında güvenli iletişim kurma yeteneğini başarıyla kanıtlamıştır. Sürekli değişen doğrulama süreci, güvenlik katmanını güçlendirirken aynı zamanda tek kullanımlık doğrulama imkânı sunarak iletişimin güvenliğini sağlamaktadır.

6.2. MEKANİZMA HABERLEŞME GÜVENLİĞİ

Haberleşme güvenliği açısından, RSA ve AES algoritmalarının birlikte kullanılması büyük önem taşımaktadır. Bu algoritmaların birlikte kullanılması, doğrulama süreçlerinde veri güvenliğini arttırır. İletişim sırasında sayaç bilgisini şifreleyerek güvenli bir iletim sağlanır ve böylece yetkisiz erişim engellenir.

6.3. ALTERNATİF GÜVENLİK STRATEJİSİ VE BAĞIMSIZ YAKLAŞIM

Sayaç tabanlı güvenlik mekanizması cTrust, mevcut güvenlik yöntemlerine alternatif bir yaklaşım sunar. Google, Apple ve diğer üçüncü platformlara bağımlı olmadan uygulanabilen bağımsız bir yapı sunar. Bu, mobil uygulamaların belirli bir platforma bağımlı olmadan güvenli bir şekilde korunabileceği anlamına gelir. Geleneksel olarak, birçok güvenlik çözümü belirli bir platformun altyapısına veya API'larına dayanırken, cTrust'ın bağımsız yapısı, farklı platformlarda aynı güvenlik standartlarını sağlar. Bu durum, kuruluşlara ve mobil uygulama geliştiricilerine daha fazla esneklik ve kontrol imkânı sunar. Çünkü cTrust'ın bağımsızlığı, güvenlik açısından üçüncü taraf platformlara bağımlı olma zorunluluğunu ortadan kaldırır. Bu sayede, mobil uygulama geliştiricileri mobil uygulamalarını farklı platformlara kolayca yayınlarken güvenlik standartlarını koruyabilirler.

7. SONUÇLAR VE ÖNERİLER

7.1. SONUÇLAR

Bu çalışma, mobil uygulamalar ile API servisleri arasındaki güvenli iletişimi sağlamak amacıyla çift taraflı sayaç mekanizmasına dayanan bir yapı üzerine kurulmuştur. Bu mekanizmanın uygulanması ve genel yapılabilirliğinin değerlendirilmesi amacıyla öncelikle Android işletim sistemi üzerinde gerçekleştirilmiştir. Ancak, belirtmek gerekir ki, sayaç tabanlı güvenlik mekanizması cTrust sadece Android işletim sistemi ile sınırlı değildir; aynı zamanda iOS, HarmonyOS, Windows, Linux gibi çeşitli platformlarda çalışan uygulamalar ve API servisleri arasındaki iletişimi güvence altına almak ve uygulamaları tasdik etmek için de kullanılabilmektedir. Elde edilen sonuçlar ve öne çıkan unsurlar aşağıda verilmiştir.

Tek kullanımlık token bilgisi: Çalışma, API haberleşmelerinde sürekli değişen ve bir daha kullanılamayan token bilgisinin üretilmesini sağlayarak güvenlik katmanını güçlendirmiştir. Bu yaklaşım, yetkisiz erişim girişimlerini önlemekte ve her iletişimde yeni bir güvenlik token'ı kullanılmasını sağlamaktadır.

Bağımsız doğrulama mekanizmaları: API servisleri ile mobil uygulamalar arasındaki doğrulama mekanizmaları, büyük teknoloji firmalarının SafetyNet, Play Integrity ve DeviceCheck gibi kütüphanelerine bağımlı olmayacak şekilde tasarlanmıştır. Bu sayede kurumlar ve bireyler, kendi sunucuları üzerinde bağımsız doğrulama mekanizmalarını uygulayabilirler.

Bağımsız yapılar için uygunluk: Çalışmanın öne çıkan özelliklerinden biri, kurumların ve kişilerin kendi sunucuları üzerinde kurgulayabilecekleri bir yapı olmasıdır. Bu, bağımlılığı ortadan kaldırarak daha özelleştirilebilir ve güvenli bir doğrulama süreci sağlamaktadır.

Google Play ve App Store dışındaki uygulamalar için çözüm: Çalışma, Google Play ve App Store mobil uygulama mağazaları dışında, özellikle askeri ve istihbarat gibi kritik alanlarda geliştirilen uygulamalar için doğrulama mekanizmalarını bağımsız hale getirme amacını taşımaktadır. Bu tür özel uygulamalar için daha güvenli ve kontrol edilebilir bir doğrulama süreci sağlamayı amaçlamaktadır.

Bu sonuçlar, mobil uygulama ile API servisleri arasındaki iletişim güvenliğini artırmak amacıyla sayaç mekanizmasının etkili bir strateji olduğunu göstermektedir. Sayaç mekanizması, bağımsız doğrulama ve tek kullanımlık erişim belirteci yapılarıyla birleşerek mobil uygulama ve API servisleri arasındaki güvenli iletişimi sağlamak için sunulmuş açık kaynak bir mekanizmadır.

7.2. ÖNERİLER

Mobil uygulama ve API servisleri arasındaki güvenli iletişimin sürdürülebilirliği için aşağıdaki öneriler dikkate alınmalıdır.

Güvenlik standartlarının sürekli izlenmesi ve güncellenmesi: Kullanılan güvenlik standartları, sürekli olarak gözden geçirilmeli ve güncellenmelidir. Yeni tehditler ve zayıf noktalar belirlendikçe, güvenlik önlemleri ve standartlar buna göre güncellenmelidir.

Doğrulama ve yetkilendirme süreçlerinin değerlendirilmesi ve güncellenmesi: Mobil uygulama ve API servisleri arasındaki doğrulama ve yetkilendirme süreçleri, periyodik olarak değerlendirilmeli ve güncellenmelidir. Bu, her iki tarafın daha güçlü ve güvenli doğrulama süreçlerine sahip olmalarını sağlayacaktır.

Mobil uygulama güvenliği araştırmalarının devam ettirilmesi: Mobil uygulama güvenliği konusundaki araştırmalara devam edilmeli ve yeni güvenlik tehditlerine karşı etkili önlemler geliştirmek amacıyla sürekli olarak yenilikçi çözümler araştırılmalıdır. Bu, mevcut güvenlik açıklarının tespit edilmesi ve kapatılması için önemlidir.

Önerilen mekanizmanın sürekli iyileştirilmesi ve uyarlanması: Önerilen mekanizma mobil uygulama ile API servisleri arasındaki haberleşmenin güvenli hale getirilmesi ve mobil uygulamaların doğrulanması süreçlerini barındırmaktadır. Bu süreçler üzerinde yapılan değişiklikler ve güncellemeler, farklı doğrulama süreçlerinin elde edilmesine ve mekanizmanın sürekli olarak iyileştirilmesine olanak tanır.

Bu önerilerin uygulanması, mobil uygulama ve API servisleri arasındaki iletişimin güvenliğini artırabilir ve güncel güvenlik tehditlerine karşı daha etkili önlemler alınmasına olanak sağlayabilir.

8. KAYNAKLAR

[1] F. M. Asmare, ve L. G. Ayalew, "Security challenges in the transition to 4G mobile systems in developing countries", *Cogent Engineering*, c. 10, sayı 1, ss.1-14, 2023.

[2] M. Banafaa, I. Shayea, J. Din, M. H. Azmi, A. Alashbi, Y. I. Daradkeh, ve A. Alhammadi, "6G Mobile communication technology: requirements, targets, applications, challenges, advantages, and opportunities", *Alexandria Engineering Journal*, c. 64, ss. 245-274, 2023.

[3] P. Satyanarayana, G. Diwakar, B. V. Subbayamma, N. V. Phani Sai Kumar, M. Arun, ve S. Gopalakrishnan, "Comparative analysis of new meta-heuristic-variants for privacy preservation in wireless mobile adhoc networks for IoT applications", *Computer Communications*, c. 198, ss. 262-281, 2023.

[4] E. T. Mihret, ve G. Haile, "4G, 5G, 6G, 7G and future mobile technologies", *American Journal of Computer Science and Information Technology*, c. 9, sayı 2, ss. 1-7, 2021.

[5] P. Weichbroth, ve Ł. Łysik, "Mobile security: threats and best practices", *Mobile Information Systems*, c. 2020, ss. 1-15, 2020.

[6] J. Agrawal, R. Patel, P. Mor, P. Dubey, ve J. M. Keller, "Evolution of mobile communication network: from 1G to 4G", *International Journal of Multidisciplinary and Current Research*, c. 3, ss. 1100-1103, 2015.

[7] P. Taylor, "Number of smartphone mobile network subscriptions worldwide from 2016 to 2022, with forecasts from 2023 to 2028", Statista. Erişim: 14 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[8] M. M. Baig, H. GholamHosseini, ve M. J. Connolly, "Mobile healthcare applications: system design review, critical issues and challenges", *Australasian Physical and Engineering Sciences in Medicine*, c. 38, sayı 1, ss. 23-38, 2015.

[9] A. A. Shaikh, H. Alamoudi, M. Alharthi, ve R. Glavee-Geo, "Advances in mobile financial services: a review of the literature and future research directions", *International Journal of Bank Marketing*, c. 41, sayı 1, ss. 1-33, 2023.

[10] S. Lee, "Evaluation of mobile application in user's perspective: case of P2P lending apps in FinTech industry", *KSII Transactions on Internet and Information Systems*, c. 11, sayı 2, ss. 1105-1117, 2017.

[11] A. S. Drigas, ve P. Angelidakis, "Mobile applications within education: An overview of application paradigms in specific categories", *International Journal of Interactive Mobile Technologies*, c. 11, sayı 4, ss.17-29, 2017.

[12] S. Bidin, ve A. A. Ziden, "Adoption and application of mobile learning in the education industry", *Procedia - Social and Behavioral Sciences*, c. 90, ss. 720-729, 2013.

[13] D. H. Byun, H. N. Yang, ve D. S. Chung, "Evaluation of mobile applications usability of logistics in life startups", *Sustainability*, c. 12, sayı 21, ss. 1-17, 2020.

[14] X. Li, X. Zhao, W. A. Xu, ve W. Pu, "Measuring ease of use of mobile applications in e-commerce retailing from the perspective of consumer online shopping behaviour patterns", *Journal of Retailing and Consumer Services*, c. 55, 2020.

[15] A. F. Utami, I. A. Ekaputra, A. Japutra, ve S. Van Doorn, "The role of interactivity on customer engagement in mobile e-commerce applications", *International Journal of Market Research*, c. 64, sayı 2, ss. 1-23, 2022.

[16] J. Dorcic, J. Komsic, ve S. Markovic, "Mobile technologies and applications towards smart tourism- state of the art", *Tourism Review*, c. 74, sayı 1, ss. 82-103, 2019.

[17] "OWASP API Security Project", OWASP Foundation. Erişim: 14 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://owasp.org/www-project-api-security/

[18] R. T. Fielding, ve R. N. Taylor, "Principled design of the modern web architecture", *ACM Transactions on Internet Technology*, c. 2, sayı 2, ss. 115-150, 2002.

[19] C. Riva, ve M. Laitkorpi, "Designing web-based mobile services with REST", *Lecture Notes in Computer*, c. 4907, 2009.

[20] A. Belkhir, M. Abdellatif, R. Tighilt, N. Moha, Y. G. Gueheneuc, ve E. Beaudry, "An observational study on the state of REST API uses in android mobile applications", *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, ss.66-75, 2019.

[21] K. Chen, P. Liu, ve Y. Zhang, "Achieving accuracy and scalability simultaneously in detecting application clones on Android markets", *ICSE '14: 36th International Conference on Software Engineering*, ss. 175-186, 2014.

[22] Y. Xiang, W. Zhou, ve M. Chowdhury, "A survey of active and passive defence mechanisms against DDoS attacks", *Deakin University, School of Information Technology*, ss.30-40, 2004.

[23] P. Farina, E. Cambiaso, G. Papaleo, ve M. Aiello, "Understanding DDoS attacks from mobile devices", *2015 International Conference on Future Internet of Things and Cloud,* ss. 614-619, 2015.

[24] H. Li, C. Yang, L. Wang, N. Ansari, D. Tang, X. Huang, Z. Xu, ve D. Hu, "A cooperative defense framework against application-level DDoS attacks on mobile edge computing services", *IEEE Transactions on Mobile Computing*, c. 22, sayı 1, ss. 1-18, 2023.

[25] "API Keys", Swagger. Erişim: 18 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://swagger.io/docs/specification/authentication/api-keys/

[26] A. D. Rubin, "Independent one-time passwords", *Computing Systems*, c. 9, sayı 1, ss.15-27, 1996.

[27] M. H. Eldefrawy, K. Alghathbar, ve M. K. Khan, "OTP-based two-factor authentication using mobile phones", *2011 8th International Conference on Information Technology: New Generations,* ss. 327-331, 2011.

[28] F. Cheng, "Security attack safe mobile and cloud-based one-time password tokens using rubbing encryption algorithm", *Mobile Networks and Applications*, c. 16, ss. 304-336, 2011.

[29] M. H. Eldefrawy, M. K. Khan, K. Alghathbar, T. H. Kim, ve H. Elkamchouchi, "Mobile one-time passwords: Two-factor authentication using mobile phones", *Security and Communication Networks*, c. 5, sayı 5, ss. 508-516, 2012.

[30] "Çok Faktörlü Kimlik Doğrulama nedir? - MFA'ya Ayrıntılı Bakış - AWS", Amazon AWS. Erişim: 06 Şubat 2024. [Çevrimiçi]. Erişim adresi: https://aws.amazon.com/tr/what-is/mfa/

[31] S. P. Otta, S. Panda, M. Gupta, ve C. Hota, "A Systematic Survey of Multi-Factor Authentication for Cloud Infrastructure", *Future Internet*, c. 15, sayı 4, ss. 1-20, 2023.

[32] T. Suleski, M. Ahmed, W. Yang, ve E. Wang, "A review of multi-factor authentication in the internet of healthcare things", *Digital Health*, c. 9, ss.1-20, 2023.

[33] K. Zaky, ve D. H. Saxe, "Multi-factor Authentication", *IDPro Body of Knowledge*, c. 1, sayı 10, 2022.

[34] I. Chenchev, A. Aleksieva-Petrova, ve M. Petrov, "Authentication mechanisms and classification: a literature survey", *Lecture Notes in Networks and Systems*, c. 285, 2021.

[35] A. Al Abdulwahid, N. Clarke, I. Stengel, S. Furnell, ve C. Reich, "The current use of authentication technologies: An investigative review", *2015 International Conference on Cloud Computing (ICCC)*, ss. 1-8, 2015.

[36] D. Hardt, "RFC 6749 - The OAuth 2.0 Authorization Framework", Internet Engineering Task Force (IETF). Erişim: 18 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://datatracker.ietf.org/doc/html/rfc6749

[37] “OpenID Connect Discovery”, Swagger. Erişim: 18 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://swagger.io/docs/specification/authentication/openid-connect-discovery/

[38] “Bearer Authentication”, Swagger. Erişim: 18 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://swagger.io/docs/specification/authentication/bearer-authentication/

[39] D. Naylor, A. Finamorey, I. Leontiadisz, Y. Grunenbergerz, M. Mellia, M. Munafòy, K. Papagiannakiz, ve P. Steenkiste, “The cost of the ‘s’ in HTTPS”, *10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, ss. 133-140, 2014.

[40] M. Husák, M. Čermák, T. Jirsík, ve P. Čeleda, “HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting”, *EURASIP Journal on Information Security*, c. 2016, ss.1-14, 2016.

[41] A. P. Singh, ve M. Singh, “A comparative review of malware analysis and detection in HTTPs traffic”, *International Journal of Computing and Digital Systems*, c. 10, sayı 1, ss.111-123, 2021.

[42] X. Wei, ve M. Wolf, “A Survey on HTTPS implementation by android apps: Issues and countermeasures”, *Applied Computing and Informatics*, c. 13, sayı 2, ss. 101-117, 2017.

[43] T. Dierks, ve E. Rescorla, “RFC 5246 - The transport layer security (TLS) protocol - Version 1.2”, Network Working Group, IETF. Erişim: 18 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://datatracker.ietf.org/doc/html/rfc5246

[44] Y. Guo, Z. Cao, W. Yang, ve G. Xiong, “A measurement and security analysis of SSL/TLS deployment in mobile applications”, *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, c. 209, 2018.

[45] F. J. Ramírez-López, Á. J. Varela-Vaca, J. Ropero, J. Luque, ve A. Carrasco, “A framework to secure the development and auditing of SSL pinning in mobile applications: The case of android devices”, *Entropy*, c. 21, sayı 12, ss.1-19, 2019.

[46] S. Kiljan, K. Simoens, D. De Cock, M. Van Eekelen, ve H. Vranken, “A survey of authentication and communications security in online banking”, *ACM Computing Surveys*, c. 49, sayı 4, ss. 1-35, 2016.

[47] “Protect against security threats with SafetyNet”, Android Developers. Erişim: 14 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://developer.android.com/privacy-and-security/safetynet

[48] “Overview of the Play Integrity API”, Android Developers. Erişim: 14 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://developer.android.com/google/play/integrity/overview

[49] “DeviceCheck”, Apple Developer. Erişim: 14 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://developer.apple.com/documentation/devicecheck/

[50] “SDK vs. API: What’s the Difference”, IBM Cloud Education. Erişim: 21 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://www.ibm.com/blog/sdk-vs-api/

[51] M. Jones, J. Bradley, ve N. Sakimura, “JSON Web Token (JWT)”, *Internet Engineering Task Force (IETF)*, 2015.

[52] S. Ahmed, ve Q. Mahmood, “An authentication based scheme for applications using JSON web token”, *22nd International Multitopic Conference (INMIC)*, ss.1-6, 2019.

[53] E. Rushdy, W. Khedr, ve N. Salah, “Framework to secure the OAuth 2.0 and JSON web token for rest API”, *Journal of Theoretical and Applied Information Technology*, c. 99, sayı 9, ss. 2144-2161, 2021.

[54] F. Maqsood, M. Ahmed, M. M. Ali, ve M. A. Shah, “Cryptography: A comparative analysis for modern techniques”, *International Journal of Advanced Computer Science and Applications*, c. 8, sayı 6, ss. 442-448, 2017.

[55] G. Singh, “A study of encryption algorithms (RSA, DES, 3DES and AES) for information security”, *International Journal of Computer Applications*, c. 67, sayı 19, ss. 33-38, 2013.

[56] J. Thakur, ve N. Kumar, “DES, AES and Blowfish: Symmetric key cryptography algorithms simulation based performance analysis”, *International Journal of Emerging Technology and Advanced Engineering*, c. 1, sayı 2, 2011.

[57] P. Mahajan, ve A. Sachdeva, “A study of encryption algorithms AES, DES and RSA for security”, *Global Journal of Computer Science and Technology*, c. 13, sayı 15, ss. 15-22, 2013.

[58] A. M. Abdullah, “Advanced encryption standard (AES) algorithm to encrypt and decrypt data”, *Cryptography and Network Security*, 2017.

[59] J. V. Shanta, “Evaluating the performance of symmetric key algorithms: AES (advanced encryption standard) and DES (data encryption standard)”, *IJCEM International Journal of Computational Engineering & Management*, c. 15, sayı 4, ss. 43-49, 2012.

[60] G. J. Simmons, “Symmetric and asymmetric encryption”, *ACM Computing Surveys (CSUR)*, c. 11, sayı 4, ss. 305-330, 1979.

[61] M. Agrawal, ve P. Mishra, “A comparative survey on symmetric key encryption techniques”, *International Journal on Computer Science and Engineering*, c. 4, sayı 5, ss. 877-882, 2012.

[62] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini, ve Y. Khamayseh, "Comprehensive study of symmetric key and asymmetric key encryption algorithms", *2017 International Conference on Engineering and Technology (ICET)*, ss. 1-7. 2017.

[63] Q. Zhang, "An overview and analysis of hybrid encryption: the combination of symmetric encryption and asymmetric encryption", *2021 2nd international conference on computing and data science (CDS)*, ss. 616-622, 2021.

[64] S. Singh, S. K. Maakar, ve S. Kumar, "A performance analysis of DES and RSA cryptography", *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, c. 2, sayı 3, ss. 418-423, 2013.

[65] H. T. Sihotang, S. Efendi, E. M. Zamzami, ve H. Mawengkang, "Design and implementation of Rivest Shamir Adleman's (RSA) cryptography algorithm in text file data security", *Journal of Physics: Conference Series*, c. 1641, ss.1-8, 2020.

[66] P. Patil, P. Narayankar, D. G. Narayan, ve S. M. Meena, "A comprehensive evaluation of cryptographic algorithms: DES, 3DES, AES, RSA and Blowfish", *Procedia Computer Science*, c. 78, ss. 617-624, 2016.

[67] O. Rodriguez, "Boost the security of your app with the nonce field of the Play Integrity API", Android Developers Blog. Erişim: 21 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://android-developers.googleblog.com/2022/05/boost-security-of-your-app-with-nonce.html

[68] "What is a Nonce? - Cryptographic Nonce from SearchSecurity", Techtarget. Erişim: 06 Şubat 2024. [Çevrimiçi]. Erişim adresi: https://www.techtarget.com/searchsecurity/definition/nonce

[69] Z. Čekerevac, Z. Dvorak, L. Prigoda, ve P. Čekerevac, "Internet of Things and The Man-In-The-Middle Attacks- Security and Economic Risks", *MEST Journal*, c. 5, sayı 2, ss.15-25, 2017.

[70] M. Conti, N. Dragoni, ve V. Lesyk, "A Survey of Man in the Middle Attacks", *IEEE Communications Surveys and Tutorials*, c. 18, sayı 3, ss. 2027-2051, 2016.

[71] M. A. Al-Shareeda, ve S. Manickam, "Man-in-the-middle attacks in mobile ad hoc networks (MANETs): Analysis and evaluation", *Symmetry*, c. 14, sayı 8, ss.1-11, 2022.

[72] A. Canteaut, M. Naya-Plasencia, ve B. Vayssière, "Sieve-in-the-middle: Improved MITM attacks", *Lecture Notes in Computer Science*, c. 8042, 2013.

[73] I. Dacosta, M. Ahamad, ve P. Traynor, "Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties", *Lecture Notes in Computer Science*, c. 7459, 2012.

[74] “The Complete Guide to API Security”, frontegg. Erişim: 14 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://frontegg.com/wp-content/uploads/2022/04/The-Complete-Guide-to-API-Security.pdf

[75] X. Zhou, ve X. Tang, “Research and implementation of RSA algorithm for encryption and decryption”, *6th International Forum on Strategic Technology (IFOST)*, ss. 1118-1121, 2011.

[76] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, ve E. Roback, “Report on the Development of the Advanced Encryption Standard (AES)”, *Journal of Research of the National Institute of Standards and Technology*, c. 106, sayı 3, ss. 511-577, 2001.

[77] N. Qi, W. Wei, J. Zhang, W. Wang, J. Zhao, J. Li, P. Shen, X. Yin, X. Xiao, ve J Hu, “Analysis and research of the RSA algorithm”, *Information Technology Journal*, c. 12, sayı 9, ss. 1818-1824, 2013.

[78] “Android’s Kotlin-first approach”, Android Developers. Erişim: 19 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://developer.android.com/kotlin/first

[79] “Spring Boot”, Spring. Erişim: 14 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://spring.io/projects/spring-boot/

[80] “Why MySQL?”, MySQL. Erişim: 19 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://www.mysql.com/why-mysql/

[81] A. Sunardi ve Suharjito, “MVC architecture: A comparative study between laravel framework and slim framework in freelancer project monitoring system web based”, *Procedia Computer Science*, c. 157, ss. 134-141, 2019.

[82] “Mobile OS market share worldwide 2009-2023”, Statista Research Department. Erişim: 16 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[83] “Platform architecture”, Android Developers. Erişim: 14 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://developer.android.com/guide/platform

[84] “Get started with the NDK”, Android Developers. Erişim: 14 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://developer.android.com/ndk/guides

[85] “Create an Android library”, Android Developers. Erişim: 14 Ocak 2024. [Çevrimiçi]. Erişim adresi: https://developer.android.com/studio/projects/android-library

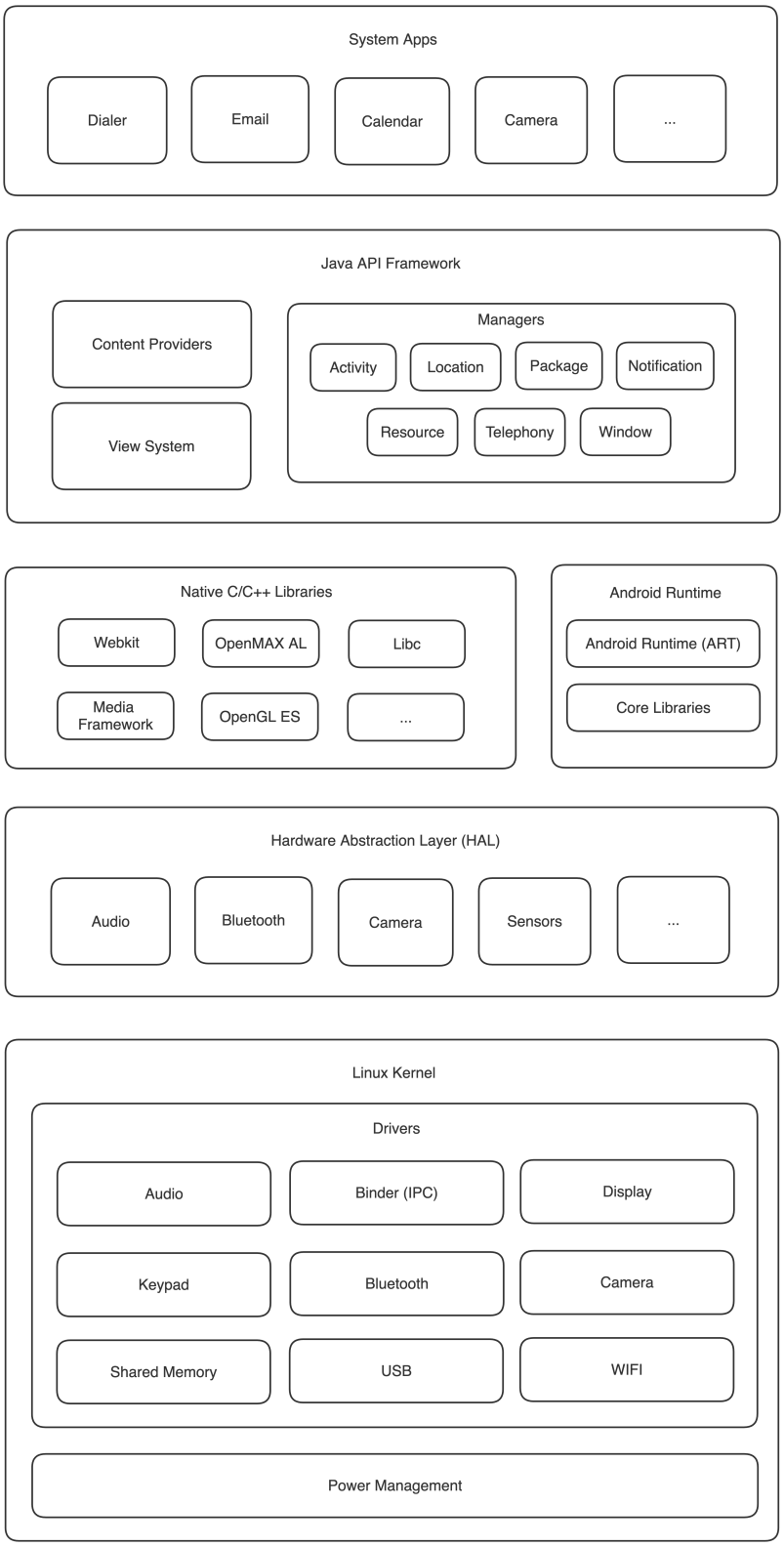
[86] C. Zhang, H. Cheng, E. Tang, X. Chen, L. Bu, ve X. Li, “Sketch-guided GUI test generation for mobile applications”, *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ss. 38-43, 2017.

[87] I. C. Morgado, ve A. C. R. Paiva, “The iMPAcT tool: Testing UI patterns on mobile applications”, *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ss. 876-881, 2015.

[88] K. S. Said, L. Nie, A. A. Ajibode, ve X. Zhou, “GUI testing for mobile applications: Objectives, approaches and challenges”, *12th Asia-Pacific Symposium on Internetware*, ss. 51-60, 2020.

9. EKLER

9.1. EK 1: ANDROID YAZILIM YIĞINI



Şekil 9.1.1. Android Yazılım Yığını

9.2. EK 2: SDK API İÇERİSİNDE KULLANILAN KRİPTOLOJİ İŞLEMLERİNE AİT KODLAR

```
import com.ctrust.api.security.manager.AESManager
import com.ctrust.api.security.manager.RSAManager
import com.fasterxml.jackson.module.kotlin.jacksonObjectMapper
import org.springframework.beans.factory.annotation.Value
import org.springframework.stereotype.Service
import java.util.*

@Service
object EncryptionHelper {
    @Value("\${ctrust.properties.api.payload.private.secret.key}")
    var payloadPrivateKey: String = "KEY"

    inline fun <reified T> encryptionToPayload(key: String, payload: T): Pair<String, String> {
        val aesKey = AESManager.generateAESKey()
        val data = jacksonObjectMapper().writeValueAsString(payload)
        val body = AESManager.encrypt(secretKey = aesKey, data = data)
        val encryptionKey = RSAManager.encrypt(key = key, data = aesKey)
        return Pair(encryptionKey, body)
    }

    inline fun <reified T> decryptToPayload(privateKey: String = payloadPrivateKey, key: String, value: String): T {
        val aesKey = RSAManager.decrypt(key = privateKey, data = key)
        val decryptionBody = AESManager.decrypt(secretKey = aesKey, data = value)
```

```
        return jacksonObjectMapper().readValue(decryptionBody, T::class.java)
    }

    inline fun <reified T> decryptToken(privateKey: String, key: String, value: String): T {
        val aesKey = RSAManager.decrypt(key = privateKey, data = key)
        val decryptionBody = AESManager.decrypt(secretKey = aesKey, data = value)
        return jacksonObjectMapper().readValue(decryptionBody, T::class.java)
    }

    inline fun <reified T> encryptionToToken(secretKey: String, model: T): String {
        val data = jacksonObjectMapper().writeValueAsString(model)
        return AESManager.encrypt(secretKey = secretKey, data = data)
    }

    fun generateSecretKey(): String = UUID.randomUUID().toString()
}

import com.ctrust.common.ext.decodeFromBase64
import com.ctrust.common.ext.encodeToBase64
import java.security.Key
import javax.crypto.Cipher
import javax.crypto.KeyGenerator
import javax.crypto.SecretKey
import javax.crypto.spec.IvParameterSpec
import javax.crypto.spec.SecretKeySpec
```

```
object AESManager {
    private const val algorithm = "AES"
    private const val transformation = "AES/CBC/PKCS5Padding"

    fun encrypt(data: String, secretKey: String): String {
        val cipher = Cipher.getInstance(transformation)
        cipher.init(Cipher.ENCRYPT_MODE, generateKey(key = secretKey), getParameterIV())
        val encryptedValue = cipher.doFinal(data.toByteArray())
        return encryptedValue.encodeToBase64()
    }

    fun decrypt(data: String, secretKey: String): String {
        val cipher = Cipher.getInstance(transformation)
        cipher.init(Cipher.DECRYPT_MODE, generateKey(key = secretKey), getParameterIV())
        val decryptedValue = cipher.doFinal(data.decodeFromBase64())
        return String(decryptedValue)
    }

    fun generateAESKey(): String {
        KeyGenerator.getInstance("AES").apply {
            init(256)
        }.run {
            convertTo32ByteString(generateKey()).run {
                return this
            }
        }
```

```
    }

    private fun getParameterIV(): IvParameterSpec = IvParameterSpec(ByteArray(16))

    private fun generateKey(key: String): Key = SecretKeySpec(key.toByteArray(), "AES")

    private fun convertTo32ByteString(aesKey: SecretKey): String {
        return aesKey.encoded.encodeToBase64().substring(0, 32)
    }
}

import org.springframework.stereotype.Service
import java.security.*
import java.security.spec.MGF1ParameterSpec
import java.security.spec.PKCS8EncodedKeySpec
import java.security.spec.X509EncodedKeySpec
import java.util.*
import javax.crypto.Cipher
import javax.crypto.spec.OAEPParameterSpec
import javax.crypto.spec.PSource

@Service
object RSAManager {
    fun generateKeyPair(): KeyPair {
        val keyPairGenerator = KeyPairGenerator.getInstance("RSA")
        keyPairGenerator.initialize(2048)
```

```
        return keyPairGenerator.generateKeyPair()
    }

    fun encrypt(key: String, data: String): String {
        val encryptCipher = Cipher.getInstance("RSA/ECB/OAEPPadding")
        encryptCipher.init(Cipher.ENCRYPT_MODE, stringToPublicKey(key), getParameterSpec())
        val encryptedValue = encryptCipher.doFinal(data.toByteArray())
        return Base64.getEncoder().encodeToString(encryptedValue)
    }

    fun decrypt(key: String, data: String): String {
        val decryptCipher = Cipher.getInstance("RSA/ECB/OAEPPadding")
        decryptCipher.init(Cipher.DECRYPT_MODE, stringToPrivateKey(key), getParameterSpec())
        return String(decryptCipher.doFinal(Base64.getDecoder().decode(data)), Charsets.UTF_8)
    }

    private fun stringToPublicKey(publicKeyString: String?): PublicKey {
        val publicKeyBytes: ByteArray = Base64.getDecoder().decode(publicKeyString)
        val keyFactory: KeyFactory = KeyFactory.getInstance("RSA")
        val keySpec = X509EncodedKeySpec(publicKeyBytes)
        return keyFactory.generatePublic(keySpec)
    }

    private fun stringToPrivateKey(privateKeyString: String?): PrivateKey {
```

```
        val privateKeyBytes: ByteArray = Base64.getDecoder().decode(privateKeyString)
        val keyFactory: KeyFactory = KeyFactory.getInstance("RSA")
        val keySpec = PKCS8EncodedKeySpec(privateKeyBytes)
        return keyFactory.generatePrivate(keySpec)
    }

    fun publicKeyToString(publicKey: PublicKey): String {
        val publicKeyBytes = publicKey.encoded
        return Base64.getEncoder().encodeToString(publicKeyBytes)
    }

    fun privateKeyToString(privateKey: PrivateKey): String {
        val privateKeyBytes = privateKey.encoded
        return Base64.getEncoder().encodeToString(privateKeyBytes)
    }

    private fun getParameterSpec() =
        OAEPParameterSpec("SHA-256", "MGF1", MGF1ParameterSpec.SHA256,
PSource.PSpecified.DEFAULT)
}

import com.ctrust.api.security.type.Algorithm
import com.ctrust.api.security.manager.HashManager
import org.springframework.stereotype.Service

@Service
```

```
class HashHelper(
    private val hashManager: HashManager
) {
    fun getAppUUID(packageName: String, signature: String): String {
        return hashManager.hash(
            data = "$packageName:$signature",
            algorithm = Algorithm.SHA256
        )
    }
}

import com.ctrust.api.security.type.Algorithm
import org.springframework.stereotype.Service
import java.nio.charset.StandardCharsets
import java.security.MessageDigest

@Service
class HashManager {
    fun hash(data: String, algorithm: Algorithm): String {
        return try {
            val md = MessageDigest.getInstance(algorithm.hash)
            md.update(data.toByteArray(StandardCharsets.UTF_8))
            val bytes = md.digest()
            val sb = StringBuilder()
            for (aByte in bytes) {
                sb.append(String.format("%02x", aByte))
```

```
            }
            sb.toString()
        } catch (e: Exception) {
            e.printStackTrace()
            ""
        }
    }
}
```

9.3. EK 3: UYGULAMA API - CTOKEN BİLGİSİNİ DOĞRULAMA İŞLEMİNİ GERÇEKLEŞTİREN KODLAR

```
import com.auth0.jwt.JWT
import com.auth0.jwt.algorithms.Algorithm
import com.auth0.jwt.exceptions.TokenExpiredException


object JwtHelper {
    fun validateToken(secretKey: String, token: String): Pair<Boolean, Any?> {
        return try {
            val algorithm = Algorithm.HMAC256(secretKey)
            val verifier = JWT.require(algorithm).build()
            val verify = verifier.verify(token)
            Pair(true, verify)
        } catch (e: TokenExpiredException) {
            e.printStackTrace()
            Pair(false, "TokenExpiredException")
        } catch (e: Exception) {
            e.printStackTrace()
            Pair(false, "Exception")
        }
    }
}


object CtrustVerify {
    private const val CTRUST_APP_SECRET_KEY = "X"
    fun isValidCToken(cToken: String): Boolean {
```

```
        val validateToken = JwtHelper.validateToken(
            secretKey = CTRUST_APP_SECRET_KEY,
            token = cToken
        )
        return validateToken.first
    }
}
```

9.4. EK 4: MOBİL SDK SAYAÇ MEKANİZMASINDA KULLANILAN KODLAR

```
#ifndef TIMER_H
#define TIMER_H
#ifdef __cplusplus
extern "C" {
#endif
void startCounter();
void stopCounter();
void setCounterValue(long time);
long getCounterValue();
#ifdef __cplusplus
}
#endif
#endif // TIMER_H

#include <pthread.h>
#include <unistd.h>
#include <stdio.h>
static long counter = 0;
static pthread_t counterThread;
static pthread_mutex_t counterMutex = PTHREAD_MUTEX_INITIALIZER;
static int stopFlag = 0;

void *incrementCounterThread(void *arg) {
    while (!stopFlag) {
        sleep(1);
```

```
        pthread_mutex_lock(&counterMutex);
        counter += 1000;
        pthread_mutex_unlock(&counterMutex);
    }
    return NULL;
}

void startCounter() {
    stopFlag = 0;
    pthread_create(&counterThread, NULL, incrementCounterThread, NULL);
}

void stopCounter() {
    stopFlag = 1;
    pthread_join(counterThread, NULL);
}

long getCounterValue() {
    pthread_mutex_lock(&counterMutex);
    long value = counter;
    pthread_mutex_unlock(&counterMutex);
    return value;
}

void setCounterValue(long newValue) {
    pthread_mutex_lock(&counterMutex);
```

```
    counter = newValue;
    pthread_mutex_unlock(&counterMutex);
}

// ctrust.c
#include <jni.h>
#include <string>
#include "util/timer/timer.h"

// region 'Time'
extern "C" JNIEXPORT void JNICALL
Java_com_ctrust_ctrust_Sensitive_setTime(JNIEnv *env, __attribute__((unused)) jobject
instance, jlong time) {
    stopCounter();
    setCounterValue((long) time);
    startCounter();
}

extern "C" JNIEXPORT jlong JNICALL
Java_com_ctrust_ctrust_Sensitive_getTime(JNIEnv *env, __attribute__((unused)) jobject
instance) {
    return getCounterValue();
}
// endregion

// region 'Utils'
jbyteArray stringToBytes(JNIEnv *env, jobject instance, jstring inputString) {
```

```
    const char *inputChars = env->GetStringUTFChars(inputString, 0);
    jsize inputLength = env->GetStringUTFLength(inputString);
    jbyteArray byteArray = env->NewByteArray(inputLength);
    env->SetByteArrayRegion(byteArray, 0, inputLength, reinterpret_cast<const jbyte
*>(inputChars));
    env->ReleaseStringUTFChars(inputString, inputChars);
    return byteArray;
}

jstring bytesToString(JNIEnv *env, jobject instance, jbyteArray byteArray) {
    jsize length = env->GetArrayLength(byteArray);
    jbyte *buffer = new jbyte[length];
    env->GetByteArrayRegion(byteArray, 0, length, buffer);
    const char *charArray = reinterpret_cast<const char *>(buffer);
    jstring result = env->NewStringUTF(charArray);
    delete[] buffer;
    return result;
}

jlong getCurrentTimeMillis() {
    return static_cast<jlong>(std::chrono::duration_cast<std::chrono::milliseconds>(
        std::chrono::system_clock::now().time_since_epoch()
    ).count());
}
// endregion
```

```
// region 'Check'
const char *getPackageName(JNIEnv *env, jobject context) {
    jclass android_content_Context = env->GetObjectClass(context);
    jmethodID midGetPackageName = env->GetMethodID(android_content_Context,
"getPackageName", "()Ljava/lang/String;");
    auto packageName = (jstring) env->CallObjectMethod(context, midGetPackageName);
    return env->GetStringUTFChars(packageName, nullptr);
}

const char *getSignature(JNIEnv *env, jobject context) {
    jclass native_context = env->GetObjectClass(context);
    jmethodID methodID_func = env->GetMethodID(native_context, "getPackageManager",
"()Landroid/content/pm/PackageManager;");
    jobject package_manager = env->CallObjectMethod(context, methodID_func);
    jclass pm_clazz = env->GetObjectClass(package_manager);
    jmethodID methodId_pm = env->GetMethodID(pm_clazz, "getPackageInfo",
"(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;");

    jmethodID methodID_packageName = env->GetMethodID(native_context,
"getPackageName", "()Ljava/lang/String;");
    auto name_str = static_cast<jstring>(env->CallObjectMethod(context,
methodID_packageName));
    jobject package_info = env->CallObjectMethod(package_manager, methodId_pm,
name_str, 64);
    jclass pi_clazz = env->GetObjectClass(package_info);

    jfieldID fieldID_signatures = env->GetFieldID(pi_clazz, "signatures",
"[Landroid/content/pm/Signature;");
```

```
    jobject signatureObject = env->GetObjectField(package_info, fieldID_signatures);
    auto signatures = reinterpret_cast<jobjectArray>(signatureObject);

    jobject signature = env->GetObjectArrayElement(signatures, 0);
    jclass s_clazz = env->GetObjectClass(signature);
    jmethodID hashCodeMethod = env->GetMethodID(s_clazz, "hashCode", "()I");

    if (hashCodeMethod == nullptr) {
        return nullptr;
    }

    char buf[64];
    jint hashCode = env->CallIntMethod(signature, hashCodeMethod);
    sprintf(buf, "%d", hashCode);
    jstring signatureValue = env->NewStringUTF(buf);
    return env->GetStringUTFChars(signatureValue, nullptr);
}

jboolean check(JNIEnv *env, jobject context, const char *packageName, const char
*signature) {
    const char *_packageName = getPackageName(env, context);
    const char *_signature = getSignature(env, context);
    if (strcmp(packageName, _packageName) == 0) {
        if (strcmp(signature, _signature) == 0) {
            return true;
        }
```

```
    }
    return false;
}
// endregion

// region 'Const'
const char *getAppPackageName(JNIEnv *env) {
    jstring d = env->NewStringUTF("com.ctrust.app");
    return env->GetStringUTFChars(d, nullptr);
}

const char *getAppSignature(JNIEnv *env) {
    jstring d = env->NewStringUTF("APP_SIGNATURE");
    return env->GetStringUTFChars(d, nullptr);
}
// endregion

// region 'Package'
extern "C" JNIEXPORT jstring JNICALL
Java_com_ctrust_ctrust_Sensitive_getPackageName(JNIEnv *env, __attribute__((unused))
jobject instance, jobject context) {
    return env->NewStringUTF(getPackageName(env, context));
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_ctrust_ctrust_Sensitive_getSignature(JNIEnv *env, __attribute__((unused)) jobject
instance, jobject context) {
```

```
    return env->NewStringUTF(getSignature(env, context));
}
jlong getRandomValue(JNIEnv *env) {
    return 307;
}
jlong getCreatedAtTime(JNIEnv *env) {
    return 1700376640358; /* APP CREATED TIME */
}
// endregion

// region 'App'
extern "C" JNIEXPORT jstring JNICALL
Java_com_ctrust_ctrust_Sensitive_getAppPackageName(JNIEnv *env,
__attribute__((unused)) jobject instance) {
    std::string value = "APP_PACKAGE_NAME";
    return env->NewStringUTF(value.c_str());
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_ctrust_ctrust_Sensitive_getAppSignature(JNIEnv *env, __attribute__((unused))
jobject instance) {
    std::string value = "APP_SIGNATURE";
    return env->NewStringUTF(value.c_str());
}

extern "C" JNIEXPORT jlong JNICALL
```

```
Java_com_ctrust_ctrust_Sensitive_getCounter(JNIEnv *env, __attribute__((unused)) jobject
instance, jobject context) {
    const char *_packageName = getAppPackageName(env);
    const char *_signature = getAppSignature(env);
    if (!check(env, context, _packageName, _signature)) {
        return 0;
    }
    return getCounterValue() - (getCreatedAtTime(env) + getRandomValue(env));
}
// endregion

// region 'Public & Private Keys'
extern "C" JNIEXPORT jstring JNICALL
Java_com_ctrust_ctrust_Sensitive_getBodyPublicKey(JNIEnv *env, __attribute__((unused))
jobject instance) {
    const char *output = "KEY";
    return env->NewStringUTF(output);
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_ctrust_ctrust_Sensitive_getRequestPublicKey(JNIEnv *env,
__attribute__((unused)) jobject instance) {
    const char *output = "KEY";
    return env->NewStringUTF(output);
}

extern "C" JNIEXPORT jstring JNICALL
```

```
Java_com_ctrust_ctrust_Sensitive_getResponsePrivateKey(JNIEnv *env,
__attribute__((unused)) jobject instance) {
    const char *output = "KEY";
    return env->NewStringUTF(output);
}
// endregion
```

9.5. EK 5: MOBİL UYGULAMA ARA YÜZ TEST KODLARI

```
import androidx.activity.compose.setContent
import androidx.compose.ui.test.assertIsDisplayed
import androidx.compose.ui.test.hasClickAction
import androidx.compose.ui.test.hasText
import androidx.compose.ui.test.junit4.createAndroidComposeRule
import androidx.compose.ui.test.onNodeWithText
import androidx.compose.ui.test.performClick
import androidx.hilt.navigation.compose.hiltViewModel
import com.ctrust.app.IS_TEST
import com.ctrust.app.screen.card.CardScreen
import com.ctrust.app.screen.login.LoginScreen
import com.ctrust.app.screen.main.MainActivity
import dagger.hilt.android.testing.HiltAndroidRule
import dagger.hilt.android.testing.HiltAndroidTest
import org.junit.Before
import org.junit.Rule
import org.junit.Test


@HiltAndroidTest
class CTrustAppUiTest {
    @get:Rule
    var hiltRule = HiltAndroidRule(this)
    @get:Rule
    val composeTestRule = createAndroidComposeRule<MainActivity>()
```

```
    @Before
    fun init() {
        hiltRule.inject()
    }

    @Test
    fun uiFlowTest() {
        IS_TEST = true
        composeTestRule.activity.setContent {
            LoginScreen(viewModel = hiltViewModel()) {
                composeTestRule.activity.setContent {
                    CardScreen(viewModel = hiltViewModel())
                }
            }
        }
        composeTestRule.waitForIdle()
        val buttonLogin = hasText("Login") and hasClickAction()
        composeTestRule.onNode(buttonLogin).performClick()
        composeTestRule.waitUntilTimeout(15000)
        composeTestRule.onNodeWithText("Cards", useUnmergedTree =
true).assertIsDisplayed()
    }
}
```

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Resul SİLAY

Yabancı Dili : İngilizce

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Yüksek Lisans	Elektrik-Elektronik ve Bilgisayar Mühendisliği	Düzce Üniversitesi	2024
Lisans	Bilgisayar Mühendisliği	Kastamonu Üniversitesi	2019
Lise	Bilişim Teknolojileri / Veri Tabanı Programcılığı	Keçiören İMKB Teknik ve Endüstri Meslek Lisesi	2013

TEZDEN ÇIKAN BİLDİRİ

Silay, R., Ersöz Demir, G. (2023). cTrust - Mobil Uygulama ile API Servisleri Arasında Eş Zamanlı Koruma Mekanizması. 2. Uluslararası Mühendislik ve Fen Bilimleri Kongresi Bildiri Kitabı. 16-17 Aralık. İstanbul: Güven Plus Grup Danışmanlık A.Ş. Yayınları, 577-585.

YAYINLAR

Silay, R., Karacı, A. (2021). Patlayıcı Etki Analizi Simülasyon Yazılımının Geliştirilmesi ve Basınç Dalgası Parametrelerinin Derin Öğrenme ile Tahmin Edilmesi. Düzce Üniversitesi Bilim ve Teknoloji Dergisi, 9(6), 303-315.