T.C.

TRAKYA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BLOKZİNCİRİ VE AKILLI SÖZLEŞMELER: GÜVENLİ BİR DİJİTAL SERTİFİKASYON UYGULAMASI GELİŞTİRİLMESİ

KEREM ATAŞEN

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Tez Danışmanı: Dr. Öğr. Üyesi Deniz TAŞKIN

EDİRNE-2019

KABUL VE ONAY SAYFASI

Kerem ATAŞEN' in hazırladığı "Blokzinciri ve Akıllı Sözleşmeler: Güvenli Bir Dijital Sertifikasyon Uygulaması Geliştirilmesi" başlıklı bu tez, tarafımızca okunmuş, kapsam ve niteliği açısından Bilgisayar Mühendisliği Anabilim Dalında bir Yüksek Lisans tezi olarak kabul edilmiştir.

Jüri Üyeleri (Ünvan, Ad, Soyad):	İmza
Dr. Öğr. Üyesi Deniz TAŞKIN	.....
Dr. Öğr. Üyesi Bora ASLAN	.....
Dr. Öğr. Üyesi Edip Serdar GÜNER	.....
Dr. Öğr. Üyesi Altan MESUT	.....
Dr. Öğr. Üyesi Tarık YERLİKAYA	.....

Tez Savunma Tarihi: 09/04/2019

Bu tezin Yüksek Lisans tezi olarak gerekli şartları sağladığını onaylarım.

Dr. Öğr. Üyesi Deniz TAŞKIN	Dr. Öğr. Üyesi Bora ASLAN
Tez Danışmanı	Tez Eş Danışmanı
İmza	İmza
.....	.....

Trakya Üniversitesi Fen Bilimleri Enstitüsü onayı

(Ünvan, Ad, Soyad)

Fen Bilimleri Enstitüsü Müdürü

Prof. Dr. Murat YURTCAN

T.Ü.FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ YÜKSEK LİSANS PROGRAMI

DOĞRULUK BEYANI

Trakya Üniversitesi Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında, tüm verilerin bilimsel ve akademik kurallar çerçevesinde elde edildiğini, kullanılan verilerde tahrifat yapılmadığını, tezin akademik ve etik kurallara uygun olarak yazıldığını, kullanılan tüm literatür bilgilerinin bilimsel normlara uygun bir şekilde kaynak gösterilerek ilgili tezde yer aldığını ve bu tezin tamamı ya da herhangi bir bölümünün daha önceden Trakya Üniversitesi ya da farklı bir üniversitede tez çalışması olarak sunulmadığını beyan ederim.

09/04/2019

Kerem ATAŞEN

imza

Yüksek Lisans Tezi

Blokzinciri ve Akıllı Sözleşmeler: Güvenilir Bir Dijital Sertifikasyon Uygulaması Geliştirilmesi

T.Ü Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

ÖZET

Blokzinciri teknolojisi ilk olarak 2008 yılında Satoshi Nakamoto'nun "Bitcoin: A Peer-to-Peer Electronic Cash System" isimli makalesiyle ortaya atılmış, klasik veritabanı teknolojilerinin merkeziyetçiliğine, müdahale edilebilirliğine ve değiştirilebilirliğine karşı çözüm olarak sunulmuştur. Genel kanının aksine blokzinciri ile bitcoin farklı kavramlardır. Blokzinciri bitcoin sisteminde kullanılan teknojidir ve birçok daha farklı kullanım alanı mevcuttur.

Blokzinciri merkeziyetçi olmayan yapısıyla varlığı daha eski olan geleneksel çözümlerinden daha güvenilir bir yapıya sahiptir. Sunduğu bu güvenilir yapı sayesinde kurumlar ve ticari kuruluşlar hitap ettiği kesimin güvenini sağlama alırken ulaşamadığı kesimlerin de güvenini kazanmaya başlamaktadırlar.

Bu tezde blokzinciri teknolojisi, bazı blokzinciri çözümlerinin sağladığı akıllı sözleşmeler ve sağlanan bu akıllı sözleşmelerle birlikte blokzinciri ağında birlikte çalışan merkezi olmayan uygulamalardan (DAPP-Decentralized Applications) bahsedilmiş ve güvenilir bir merkezi olmayan dijital sertifikasyon uygulaması geliştirilmiştir. Akıllı sözleşme koşan blokzinciri çözümlerinden Ethereum temel alınarak hareket edilmiştir. Lokalde çalışan bir özel(private) bir Ethereum blokzinciri ağı ile birlikte Dapp geliştirmek üzere tasarlanmış bir framework kullanılmıştır. Bir Ethereum ağıyla haberleşmek için kullanılmak üzere tarayıcı üzerinden çalışan bir light düğüm (node) kullanılmıştır.

Geliştirilen akıllı sözleşmeyi koşan Dapp uygulaması lokal özel Ethereum blokzinciri ağındaki bir adresten tarayıcıya aktarılan test bakiyesi ile finanse edilmiştir. Bir sözleşme oluşturma ve bu sözleşmeyi Dapp uygulamasından çağırmak için harcanan Ethereum miktarı ve zaman lokal blokzinciri ağının arayüzünden takip edilmiştir. İşverene, işe alacakları aday çalışanlarından gelecek diploma ve sertifikaların güvenilir bir şekilde saklandığı ve değişikliğe uğramadığı veya sahte olmadığı blokzinciri teknolojisi ve akıllı sözleşmelerle garanti edilmiştir.

Yıl : 2019

Sayfa Sayısı : 77

Anahtar Kelimeler : Blokzinciri, Akıllı Sözleşmeler, Ethereum, Dapp, Solidity, Ganache, Truffle, Metamask

Master's Thesis

Blockchain and Smart Contracts: Developing a Reliable Digital Certification Application

Trakya University Institute of Natural Sciences

Computer Engineering Department

ABSTRACT

Blockchain technology was first introduced in 2008 by Satoshi Nakamoto's "Bitcoin: A Peer-to-Peer Electronic Cash System" article and presented as a solution to the centralization, interoperability and interchangeability of classical database technologies. Blockchain is not a money laundering mechanism. On the contrary, it has a more reliable structure than the older ones with its non-centralized structure. Thanks to this reliable structure, institutions and commercial establishments ensure the confidence of the people they address and they start to gain the trust of the people they cannot reach.

In this thesis, the blockchain technology, smart contracts provided by some blockchain solutions and decentralized applications (DAPP) which work together in the blockchain network have been mentioned and a reliable decentralized web applicaiton has been developed. The Ethereum, which is one of the block chain platforms running smart contract, is used. A framework that is designed to develop Dapp and a private Ethereum blockchain network is used together in the personal computer. A light düğüm is used to communicate with an Ethereum network.

The Dapp app, which runs the developed smart contract, is financed by the test balance transferred to the browser from an address in the local private Ethereum blockchain network. The amount of Ethereum spent to create a contract and to recall the contract from the Dapp application was followed by the interface of the local blockchain network. The employer is guaranteed with blockchain technology and smart contracts where the certificates from the candidate employees to be hired are stored reliably, not modified or counterfeited.

Year : 2019

Number of Pages : 77

Keywords : Blockchain, Smart Contracts, Ethereum, Dapp, Solidity, Ganache, Truffle, Metamask

TEŞEKKÜR

Beni bugünlere getiren annem ve babam başta olmak üzere, tez çalışma konumun belirlenmesi ve tezimin hazırlanma sürecinde bilgi, tecrübe ve desteklerini benden esirgemeyen sayın danışman hocam Dr. Öğr. Üyesi Deniz TAŞKIN ve sayın eş danışman hocam Dr. Öğr. Üyesi Bora ASLAN'a sonsuz saygı ve teşekkürlerimi sunarım.

Son olarak tez çalışmam boyunca bana karşı anlayışlı olan, manevi desteklerini ve yardımlarını esirgemeyen eşim Gülçin ATAŞEN'e sonsuz sevgilerimle.

İÇİNDEKİLER

KABUL VE ONAY SAYFASI..... **Hata! Yer işareti tanımlanmamış.**
ÖZET.....iii
ABSTRACT.....iv
TEŞEKKÜR.....v
İÇİNDEKİLER.....vi
SİMGE VE KISALTMALAR DİZİNİ.....ix
ŞEKİLLER DİZİNİ.....xi
ÇİZELGELER DİZİNİ.....xii
BÖLÜM 1.....1
GİRİŞ.....1
BÖLÜM 2.....3
BLOKZİNCİRİ.....3
2.2. Blokzinciri Veri Yapıları ve Çalışma Mantığı.....4
2.3. Blokzinciri Ağ Türleri.....5
2.4. Blokzinciri ve Bağlı Listelerin Karşılaştırılması.....7
2.5. Blokzinciri ile Merkezi ve Dağıtık Veritabanı Sistemlerinin Karşılaştırılması.....8
2.6. Uzlaşı (Consensus) Algoritmaları.....8
2.6.1. Proof of Work.....8
2.6.2. Proof of Stake.....10
2.6.3. Delegated Proof of Stake.....11
2.6.4. Leased Proof of Stake.....12
2.6.5. Proof of Elapsed Time.....12
2.6.6. Practical Byzantine Fault Tolerance.....12
2.6.7. Simplified Byzantine Fault Tolerance.....13
2.6.8. Delegated Byzantine Fault Tolerance.....13
2.6.9. Directed Acyclic Graphs (DAG).....14
2.6.10. Proof-of-Activity.....15
2.6.11. Proof-of-Importance.....15
2.6.12. Proof-of-Capacity.....16
2.6.13. Proof-of-Burn.....16
2.6.14. Proof-of-Weight.....17
2.6.15. Uzlaşı Algoritmalarının Karşılaştırılması.....17
2.7. ICO, IPO, DAO ve DAICO Kavramları.....18
2.8. Blokzincirinin Dezavantajları.....19

2.9. Mevcut ve Gelecekteki Muhtemel Blokzinciri Uygulama Alanları 22
BÖLÜM 3 26
AKILLI SÖZLEŞMELER 26
3.1. Akıllı Sözleşme Koşabilen Blokzinciri Platformları 27
3.1.1. Ethereum 27
3.1.1.1. Ethereum Hesapları 28
3.1.1.2. Mesajlar ve İşlemler (Messages and Transactions) 31
3.1.1.3. Ethereumda Ücretler 33
3.1.1.4. Akıllı Sözleşme Kodunun Çalıştırılması 39
3.1.1.5. Ethereum Blok Yapısı 40
3.1.1.6. Ethereum Durum Geçiş Fonksiyonu 42
3.1.1.7. Ethereum Blokzinciri ve Madencilik 44
3.1.1.8. Endişelere Çözüm GHOST Protokolü 44
3.1.1.9. Hesaplama ve Turinge Tam Uyumluluk 46
3.1.1.10. Ethereum Para Birimi 47
3.1.1.11. Ethereum Token Standartları 47
3.1.1.12. Ethereum İstemci Türleri 51
3.1.1.13. Ethereum Ağları 52
3.1.2. Ethereum Dışındaki Diğer Platformlar 55
3.2. Ethereum Platformunda Koşan Akıllı Sözleşmelerde Kullanılabilen Programlama Dilleri 55
3.2.1. Solidity 55
3.2.2. Diğerler Programlama Dilleri 56
3.3. Akıllı Sözleşme Koşabilen Ethereum dışındaki diğer Platformlarda Kullanılan Programlama Dilleri 58
BÖLÜM 4 62
Merkezi Olmayan Uygulamalar (DAPP) 62
4.1. Truffle 63
4.2. Ganache 63
4.3. Metamask 64
4.4. Infura 64
BÖLÜM 5 65
BLOKZİNCİRİ TABANLI DİJİTAL SERTİFİKASYON UYGULAMASININ GELİŞTİRİLMESİ 65
5.1. Lokal Blokzincirinin Oluşturulması 65
5.2. Test Hesabın Metamask Light Düğümü Olarak Devreye Alınması 65
5.3. DAPP Geliştirilmesi 67

BÖLÜM 6 71
SONUÇLAR 71
KAYNAKLAR 72

SİMGE VE KISALTMALAR DİZİNİ

B2B	:	business-to business
PoW	:	Proof of Work
DDoS	:	Distributed Denial of Service
BTC	:	Bitcoin
BTG	:	Bitcoin Gold
ETC	:	Ethereum Classic
ETH	:	Ethereum
BCH	:	Bitcoin Cash
Dapp	:	Decentralized Applications
PoS	:	Proof of Stake
DPoS	:	Delegated Proof of Stake
LPoS	:	Leased Proof of Stake
PoET	:	Proof of Elapsed Time
pBFT	:	Practical Byzantine Fault Tolerance
DAG	:	Directed Acyclic Graphs
PoA	:	Proof-of-Activity
PoI	:	Proof-of-Importance
PoC:	:	Proof-of-Capacity
ASIC	:	Application Specific Integrated Circuits
PoB	:	Proof-of-Burn
PoWeight	:	Proof-of-Weight
SBFT	:	Simplified Byzantine Fault Tolerance
DBFT	:	Delegated Byzantine Fault Tolerance
ICO	:	Initial Coin Offering
IPO	:	Initial Public Offering
DAPP	:	Decentralized Applications
EVM	:	Ethereum Virtual Machine
RLP	:	Recursive Length Prefix
GHOST	:	Greedy Heaviest Observed Subtree

CLI	:	Command Line Interface
GUI	:	Graphical User Interface
LDL	:	Level Description Language
DAO	:	Decentralized Autonomous Organization
RPC	:	Remote Procedural Call
DAPP	:	Decentralized Applications

ŞEKİLLER DİZİNİ

Şekil 2.1. Blokzinciri Yapısı 3
Şekil 2.2. Merkle Tree 4
Şekil 3.1. Ethereum Blok Boyutları 28
Şekil 3.2. Ethereum Blok Oluşturulma Süreleri 29
Şekil 3.3. Ethereum Madencilerine Ödenen Ödüller 29
Şekil 4.1. Ganache Masaüstü Uygulaması ve Tanımlı Test Hesapları 63
Şekil 5.1. Ganache Üzerinden Blokzincirinin Görüntülenmesi 65
Şekil 5.2. Ganache-Metamask Entegrasyonu 67
Şekil 5.3. Üniversite Girişi İçin DAPP Arayüzü 67
Şekil 5.4. Diploma Yükleme ve İşlem Onaylama Ekranları 68
Şekil 5.5. Ganache Üzerinden Seçilen Blok Bilgisi Görüntüleme 68
Şekil 5.6. Diploması Kayıtlı Tüm Öğrencilerin Görüntülendiği Ekran 69
Şekil 5.7. Diploma Doğrulama Ekranı 70

ÇİZELGELER DİZİNİ

Çizelge 2.1. Bitcoin Enerji Tüketimi 9
Çizelge 2.2. Uzlaşı Algoritmalarının Karşılaştırılması 16
Çizelge 2.3. ICO-IPO Karşılaştırması 19
Çizelge 3.1. EVM Opcode Maliyet Grupları 32
Çizelge 3.2. EVM Opcode Gas Maliyetleri 34
Çizelge 3.3. Ethereum Para Birimleri 46

BÖLÜM 1

GİRİŞ

İlk olarak 2008 yılında Satoshi Nakamoto'nun "Bitcoin: A Peer-to-Peer Cash System" isimle makalesiyle ortaya çıkan blokzinciri teknolojisi günümüzde kendine finans, bankacılık, sağlık sektörü ve tedarik zinciri gibi güven teminatı gerektiren, takip edilebilirliği ve şeffaflığı yüksek olması istenen alanlarda yer bulmaya başlamıştır. Geçmişten günümüze iletişim ve alışveriş halinde olan kişilerin birbirlerine güven duymamalarından kaynaklanan güven teminatçısı üçüncü taraflar ortaya çıkmıştır. Üçüncü taraflardan ücret karşılığında alınan güven teminatı tarafların giderlerinin artmasına neden olurken gerçekleşecek işlemlerin istenilen hızlarda gerçekleşmemesi de ücret-fayda-zaman bağlamında hizmet satın alanları memnun etmemektedir. Bunlara ek olarak güven sağlayıcı üçüncü tarafların veya doğrudan hizmet sunan tarafların bu hizmetlerle ilgili verileri tekelinde bulundurmaları bu veriye olan güvenin sorgulanmasına neden olmaktadır.

Günümüzde sahte transkript düzenleyerek burs başvurusunda bulunan öğrencilerin yanı sıra yıllardır öğretmenlik, hekimlik veya diğer bazı meslekleri sürdürdükleri halde diplomaları gerçek olmayan insanların var oldukları görülmektedir. Bu durum kişilerin yeterliliklerini belgeledikleri diplomaların kontrol edilebileceği bir mekanizma olmamasından değil bu mekanizmaya daha sonra art niyetli bir şekilde müdahale olup olmadığının bilinememesinden kaynaklıdır. Bu durum geriye dönük takip edilebilirliği yüksek, verilerin tekelleşmediği merkeziyetçi olmayan bir sisteme duyulan gereksinimi ortaya koymaktadır.

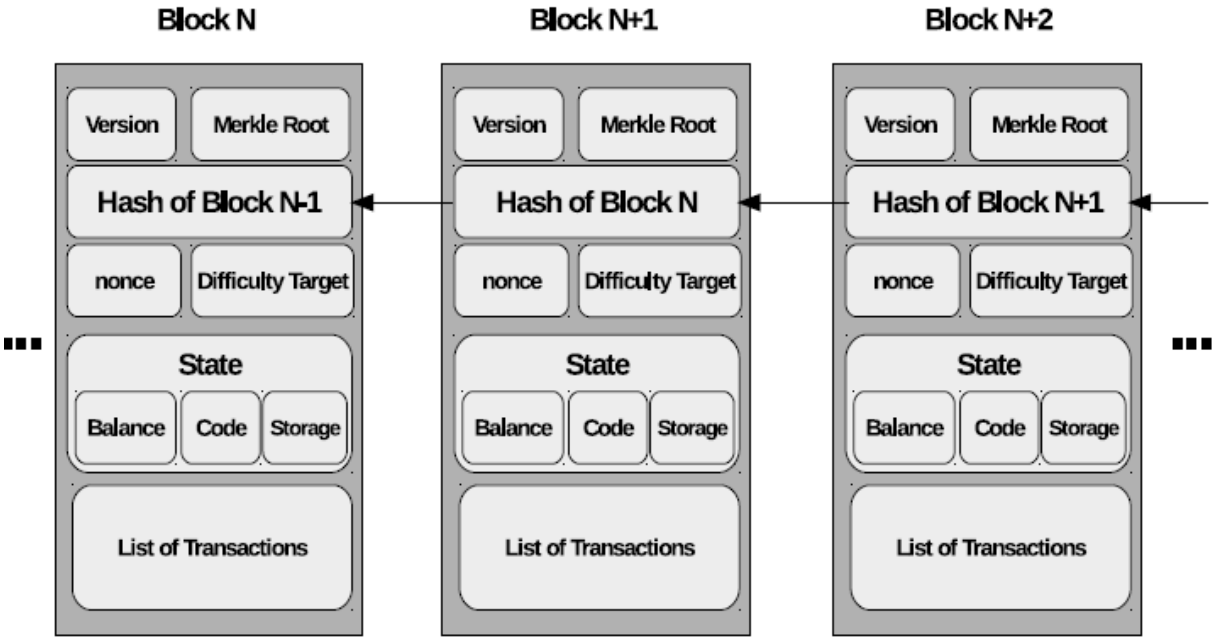
Bu tez çalışmasının ikinci bölümünde blokzinciri teknolojisi kapsamlı bir şekilde ele alınmıştır. Bunu takip eden bölümde ise akıllı sözleşmelere ek olarak akıllı sözleşmelerle ilgili platformlar, akıllı sözleşmeler için kullanılabilen programlama dilleri tez çalışmasında kullanılanları kapsamlı ve diğerleri yüzeysel olacak şekilde ele alınmıştır. Dördüncü bölümde uygulama geliştirilmesi için kullanılan teknolojiler açıklanmıştır. Beşinci bölümde önceki bölümlerdeki bilgilerin ışığında yukarıda bahsedilen eksikliklere ve yanlışlıklara çözüm olarak akıllı sözleme koşabilen blokzinciri tabanlı merkezi bir sunucusu olmayan bir uygulama geliştirilmiştir. Blokzincirinin ve akıllı sözleşmelerin değerlendirildiği başka Türkçe yayınlar mevcutken bu tez çalışmasında blokzincirinin ve akıllı sözleşmelerin değerlendirilmesine ek olarak, Ethereum tabanlı merkezi sunucusu olmayan ve akıllı sözleşme

koşan bir uygulama geliştirilmiştir. Literatür incelendiğinde ülkemizde tüm bu bilgi ve uygulamayı bir arada bulunduran Türkçe bir çalışmaya rastlanmamıştır. Bu yüzden daha sonra yapılacak çalışmalara Türkçe önemli bir kaynak olacağı düşünülmektedir.

BÖLÜM 2

BLOKZİNCİRİ

Blokzinciri, şifrelenmiş işlemleri içeren blokların birbirlerine kriptografik olarak zincirlenerek oluşturdukları dağıtık, merkeziyetçilikten uzak bir veri kayıt sistemi veya bir bloklar zinciri olarak tanımlanabilir. Blokzinciri teknolojisinin bir merkeze bağlı olmadan işlem yapması demek hizmeti talep eden kullanıcılar ile hizmeti sunan kullanıcılar arasında üçüncü bir şahsın olmaması demektir. Blokzincirinin dağıtık bir veri kayıt sistemi olması demek blokzinciri ağında bulunan bütün düğümlerle aynı işlem geçmişine sahip blokların tutulması demektir. Blokzincirine veritabanı demek doğru olmaz çünkü klasik veritabanı teknolojilerinde veri eklemenin yanısıra veri güncelleme ve silme işlemleri yapılabilirken blokzincirine kaydedilen veri bir daha asla değiştirilemez ve silinemez. Verinin değiştirilemez (immutable) olması bu sisteme duyulan güvenin haklılığını ortaya koymaktadır. Basit bir blokzinciri yapısı Şekil 2.1.' deki gibi ifade edilebilir:



Şekil 2.1. Blokzinciri Yapısı (Dorri, Kanhere, Jurdak & Gauravaram, 2017)

Bu bölümde blokzinciri teknolojisi ile ilgili kavram, teknoloji ve yöntemlere yer verilmiştir.

2.2. Blokzinciri Veri Yapıları ve Çalışma Mantığı

Blokzincirinde bulunan blokların yapısını oluşturan temel kavramlar aşağıdaki gibi ifade edilebilir.

i. **index**: Bloğun blokzincirdeki konumunu gösteren veridir. Örneğin her blokzincirinin ilk bloğunun index bilgisi 0'dır.

ii. **hash**: Bloktaki verilerin bir hash fonksiyonuna sokulduktan sonra elde edilen özet sonuçtur.

iii. **previousHash**: Kendisinden bir önceki bloğun hash değerini ifade eder.

iv. **timesamp**: Bloğun yaratıldığı zamanı gösteren zaman damgasıdır.

v. **nonce**: Madencilik işlerinde kullanılan 32 veya 64 bitlik tamsayılardır.

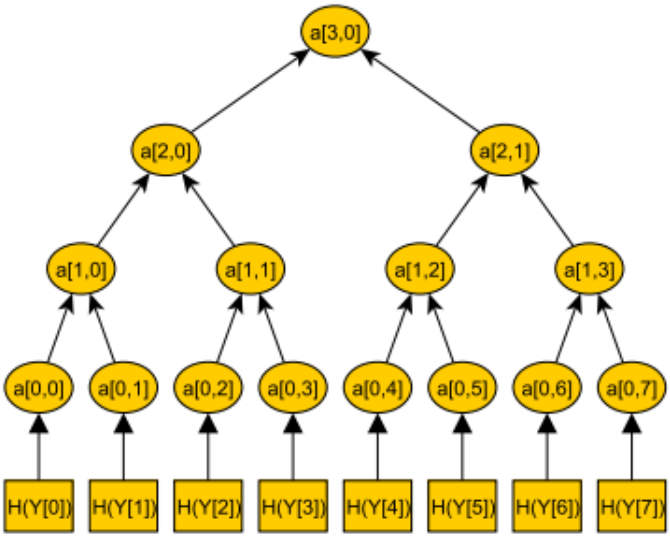
vi. **numTx**: Blok içindeki toplam işlem (transaction) sayısıdır.

vii. **İşlemler(transactions)**: Bloktaki bütün işlemlerin bulunduğu bir dizidir. İşlemlerde aşağıdaki bilgiler bulunur:

Hash: bir transaction içindeki verinin, transaction çıktılarının ve girdi verilerinin özetidir.

Type: transactionun tipidir. 3 farklı tipi vardır. Bunlardan Coin Base Transaction kriptopara tokeni sağlama mekanizması için gerçekleşen işlemleri ifade eder. Fee Transaction, transaction sahibinin madencilere ödediği ödüllerin işlemlerini ifade eder. Regular transaction ise token sahipliğinin transfer edilmesi için gerçekleştirilen işlemleri ifade eder.

viii. **Merkle Tree:** Bloktaki her işlemin hash değerleri Merkle ağacının son yapraklarını oluşturmaktadır. Bütün bu yaprakların ikili gruplar olarak hashlenmiş hali bu yaprakların ata düğümlerini (parent) oluşturur. İkili gruplar halinde hash değerlerinin hesaplanıp bir üst seviyede yarı sayıda düğüm oluşturma işlemi ağacın son düğümü olan Merkle Kök düğümü oluşana kadar devam eder.



Şekil 2.2. Merkle Tree (Becker, 2008)

2.3. Blokzinciri Ağ Türleri

Blokzincir projeleri kullanım senaryolarındaki iletişim ve uzlaşı tercihlerine göre türlere ayrılmaktadır.

2.3.1. Açık (Public) Blokzinciri Ağları

Bu tür ağlarda dileyen herkes verileri okuyabilir, işlemleri doğrulayabilir, uzlaşı algoritmasına göre zincire eklenecek bloğu oluşturabilir. Bu tür bir ağ çeşidine katılımın yüksek olması beklenir. Fazla sayıda düğümden oluşan bir blokzinciri ağı ise onay ve doğrulama mekanizması olarak işlev görecek düğümlerin artması ve merkeziyetçiliğin gerçekleşmesinin imkansızlaşması ile birlikte yüksek güvenilirlikli bir hale gelmiş demektir. Açık blokzinciri ağları iki alt gruba ayrılabilir: (Usta & Doğantekin, 2017)

2.3.1.1. Bütünüyle İzin Gerektirmeyen Açık Blokzinciri Ağları

Eğer yeni bir düğümün bir açık blokzincir ağına dahil olduktan sonra bu ağda bulunan verileri okumak ve ağda kullanılan mevcut uzlaşı algoritmasına göre işlemleri onaylamak ve yeni bloklar yaratmak için izin alması gerekmiyorsa bu blokzincir ağı bir bütünüyle izin gerektirmeyen blokzincir ağıdır. Bu tür ağlara katılım çok olacağı için verilerin eş örneklerinin tutulduğu düğüm sayısı da buna doğru orantılı olarak çok olacaktır. Bu blokzincir ağının daha güvenilir hale gelmesi anlamına gelir. Örneğin Bitcoin blokzincir ağı bir bütünüyle izin gerektirmeyen açık blokzincir ağıdır (Usta & Doğantekin, 2017).

2.3.1.2. Kısmen İzin Gerektirmeyen Açık Blokzinciri Ağları

Eğer yeni bir düğümün bir açık blokzincir ağına dahil olduktan sonra bu ağda bulunan verileri okumak için izin alması gerekmiyorsa fakat bu ağdaki mevcut uzlaşı algoritmasına göre işlemleri onaylayıp yeni bloklar eklemek için izin alması gerekiyorsa bu blokzincir ağı bir kısmen izin gerektirmeyen açık blokzincir ağıdır. Örneğin Ethereum blokzincir ağı bir kısmen izin gerektirmeyen blokzincir ağıdır (Usta & Doğantekin, 2017).

2.3.2. Özel (Private) Blokzinciri Ağları

Şirketler, kamu kurumları veya organizasyonlar açık blokzinciri ağında verilerinin herkes tarafından izlenmesini istemeyebilirler. Böyle durumlar için geliştirilmiş olan özel blokzinciri ağındaki verileri okumak için ağın sahibinin veya onun belirlediği kuralların izni gereklidir (Usta & Doğantekin, 2017). Özel blokzinciri ağları iki alt gruba ayrılabilir:

2.3.2.1. Kısmen İzin Gerektiren Özel Blokzinciri Ağları

Eğer yeni bir düğüm bir özel blokzinciri ağına dahil olduktan sonra bu ağda bulunan verileri okumak için izin alması gerekiyorsa fakat ağa dahil olan her düğümün bu ağda kullanılan mevcut uzlaşı algoritmasına göre işlemleri onaylayıp yeni bloklar yaratmasına izin veriliyorsa bu blokzinciri ağı kısmen izin gerektiren özel blokzinciri ağıdır (Usta & Doğantekin, 2017).

2.3.2.2. Bütünüyle İzin Gerektiren Özel Blokzinciri Ağları

Eğer yeni bir düğümün bir özel blokzincir ağına dahil olduktan sonra bu ağda bulunan verileri okumak ve ağda kullanılan mevcut uzlaşı algoritmasına göre işlemleri onaylayıp yeni bloklar eklemek için izin alması gerekiyorsa bu blokzincir ağına bütünüyle izin gerektiren özel blokzincir ağıdır (Usta & Doğantekin, 2017).

2.3.3. Konsorsiyum (Consortium) Blokzinciri Ağları

Kısmen özel olan blokzinciri ağlarıdır. İşlemleri doğrulama yetkisini internet bağlantısı olan herkese vermek veya tamamen bir şirketin tekeline bırakmak yerine ideal sayıda düğüm bu işlemi gerçekleştirmek için seçilir. Özel blokzinciri ağlarıyla çok sayıda ortak avantaj içeren konsorsiyum blokzinciri ağları özel blokzinciri ağlarında olduğu gibi tek bir kurumun altında değil seçilmiş düğümlerden oluşan bir grubun liderliğinde çalışır. Bu platform şirketler arası işbirliği için biçilmiş kaftandır denilebilir (Zheng, Xie, Dai, & Wang, 2016).

2.3.4. Yarı özel (Semiprivate) Blokzinciri Ağları

Açık blokzinciri ağlarında herkesin ağa katılmasının ve özel blokzinciri ağlarında ağa katılım isteklerinin sıkıca denetlenmesinin aksine yarı-özel blokzinciri tabanlı uygulamaları koşan bir şirket tarafından hakeden kullanıcılara erişim yetkisi verilir. Bu tür ağlarda genellikle B2B (business-to-business) kullanıcıları hedeflenmektedir (Radanović & Likić, 2018). ABD'nin Chicago eyaletinde bulunan Cook bölgesi arazi mülk gibi sahiplik değişmelerinin gerçekleşebileceği varlıkları takip etmek ve güvenilir bir şekilde kayıt altında tutmak için Velox.re işbirliği ile bu tür blokzinciri ağı kullanan pilot bir uygulama geliştirmiştir.

2.3.5. Hibrit Blokzinciri Ağları

Açık ve özel blokzinciri ağlarının güçlü yönlerinin alındığı ve zayıf yönlerinin sınırlandırılmaya çalışıldığı bir blokzinciri ağı çeşididir. Dolayısıyla bir hibrit blokzinciri ağında bütün defterin dünyadaki herkes tarafından erişilebilir olmasını açık blokzinciri ağı mantığı ile sağlarken arka plandan çalışan bir özel blokzinciri ağı ile de defterdeki değişikliklere erişim denetlenebilir (Sagirlar, Carminati, Ferrari, Sheehan & Ragnoli, 2018). XinFin projesi tedarik zinciri için geliştirilmiş bir hibrit blokzinciri projesidir. Bu proje hem açık bir ağ olan Ethereum blokzincirini hem de özel bir blokzinciri olan Quorum blokzincirini kullanmıştır. Bu projede özel blokzinciri ağı işlemlerin hashlerinin üretilmesi için kullanılırken bu işlemler daha sonra açık blokzinciri ağında doğrulanır.

2.4. Blokzinciri ve Bağlı Listelerin Karşılaştırılması

Bağlı listeleri oluşturan her düğümde temel olarak tutulmak istenen veri ve bir sonraki düğümün adres bilgisi tutulmaktadır. Bilgisayar bilimlerinde bağlı listeler bir veri yapısına karşılık gelir. Her bir düğümün bir sonrakiyle bağlı olması blokzincirindeki blokların birbirlerine bağlı oluşuyla benzerdir. Blokzincirindeki her blokta bir önceki bloğun hash bilgisi tutulmaktadır. Bu bilgiye bakılarak zincirin ilk bloğuna kadar ulaşılabilir. Blokzincirinde de bağlı listelerde olduğu gibi previousHash bilgisini tutmayan genesis blokları yani zinciri oluşturan ilk bloklar vardır. Blokzincirinde bir önceki bloğa hash fonksiyonuyla ulaşılırken bağlı listelerde pointer fonksiyonu ile ulaşılır. Bağlı listelerde veriler lineer bir yapıda tutulurken blokzincirinde transactionlar ve bu transactionlarla ilişkili veriler Merkle tree denilen ağaçlarda tutulur. Blokzincirinde doğrulama mekanizması varken bağlı listelerde böyle bir mekanizma yoktur (Lazarovich, 2015).

2.5. Blokzinciri ile Merkezi ve Dağıtık Veritabanı Sistemlerinin Karşılaştırılması

Geleneksel veritabanı teknolojileri istemci-sunucu mimarisini kullanmaktadır. Bir istemci merkezi bir sunucuda tutulan veriyi düzenleyebilir. Veritabanının kontrolü, veritabanına erişim sağlamadan önce bir müşterinin kimlik bilgilerini doğrulayan yetkili bir otoritede kalır. Bu yetkili otorite veri tabanının yönetiminden sorumlu olduğu için, otoritenin güvenliği tehlikedeyse, veriler değiştirilebilir, hatta silinebilir. Dağıtık veritabanlarında veritabanını oluşturan bütün makinelere güven zorunludur. Hiçbir makinenin verinin bütünlüğüne, doğruluğuna ve güvenliğine kötü niyetli bir yaklaşım sergilemeyeceği varsayılır ve bu yapı tamamen merkezidir.

Blokzinciri merkezi olmayan birden fazla düğümden oluşmaktadır. Blokzincirine veri eklenmesi için blokzinciri ağındaki düğümlerin %51'den fazla çoğunluğu bir uzlaşıya varmak zorundadır. Bu uzlaşı sayesinde verinin müdahale edilemezliği ve güvenliği garanti altına alınmış olur (Wüst & Gervais, 2018).

2.6. Uzlaşı (Consensus) Algoritmaları

Blokzincirine eklenecek bir blok ancak blokzinciri ağını oluşturan düğümlerin çoğunluğunun bir uzlaşıya varması sonucunda eklenebilir. Bu işleme uzlaşı adı verilir. Uzlaşı algoritmaları blokzinciri ağını oluşturan düğüm grubunun blokzincirine eklenecek yeni bloğun eklenmesi konusunda uzlaşıya varmasına yardım eden karar verme mekanizmalarıdır (Baliga, 2017).

Bazı uzlaşı algoritmaları ve çalışma mantıkları şu şekildedir:

2.6.1. Proof of Work

Proof of Work blokzinciri ağında tanıtılan ilk uzlaşı algoritmasıdır. Birçok blokzinciri teknolojileri bütün transactionlarını onaylamak ve bloklarını üretmek için bu algoritmayı kullanır. Bu uzlaşı algoritmasına göre ağdaki düğümler kendilerine verilen zor bir problemi çözmeye çalışırlar. Bulmaya çalıştıkları nonce denilen bir değer vardır. Bloğa yazılacak veri ile bulunmaya çalışılan bu nonce değerinin birlikte hashlenerek oluşturdukları değer düğümlerde mevcuttur. Düğümler bütün güçlerini eldeki veriyle hangi nonce değerini hashlersek bu son değere ulaşırız sorusuna harcarlar. Bu işleme madencilik adı verilir. Nonce değerini ilk bulan düğüm ağdaki bütün düğümlere doğru nonce değerini bulduğunu ilan eder. Diğer bütün düğümler bu nonce değerinin doğru olup olmadığının sağlamasını yaparlar. Yeterli çoğunluktaki düğüm bulunan bu nonce değerinin doğru olduğunu kabul eder ve

ağdaki bütün düğümlerle bu nonce değerini paylaşırsa blok zincire eklenir ve nonce değerini bulan düğüme teşvik edici ödülü olan sisteme özgü kriptopara (coin) verilir.

PoW algoritması zamanla hassaslaşmıştır. Buna sebep olarak sistemin giderek yavaşlaması, bir blok üretimi için geçen sürenin çok uzun olması ve bu yüzden isteklerin zamanında yerine getirilememesi gösterilebilir. Düğümlerin çözdükleri problemin kolaylaşması sistemin DDoS ataklarına açık hale gelmesi anlamına gelir.

Proof of Work algoritmasını ilk kullanan ve en popüler blokzinciri Bitcoin blokzinciridir. Bu algoritmada bir bloğun yaratılması için geçen yaklaşık süre 10 dakikadır. Bu süre istemci düğümlerin talebini karşılayacak bir hız olmadığı için bu algoritmanın bir dezavantajı olarak görülebilir. Dezavantaj olarak değerlendirilebilecek bir diğer konu iste bu zor problemi çözmeye çalışan düğümlerin yaptığı aşırı enerji tüketimidir (Bentov, Lee, Mizrahi & Rosenfeld, 2014). Yapılan biri araştırmaya göre PoW algoritmasını kullanan en popüler blokzinciri olan Bitcoin blokzincirinin enerji tüketim istatistikleri aşağıdaki tabloda ifade edilmiştir.

Çizelge 2.1. Bitcoin Enerji Tüketimi (O'Dwyer & Malone, 2014)

Açıklama	**Değer**
Bitcoin'in yıllık tahmini elektrik tüketimi (TWh)	64.37
Yıllık Küresel Madencilik Gelirleri	$3,218,288,371
Yıllık tahmini küresel madencilik faaliyetleri	$3,218,288,371
Mevcut Maliyet Yüzdesi	100.00%
Elektrik tüketimi açısından Bitcoin'e en yakın ülke	İsviçre
Toplam Network Hash Oranı PH/s (1,000,000 GH/s)	49,367
Transaction başına tüketilen elektrik (KWh)	640
Bitcoin tarafından elektriği sağlanabilecek Amerikan hane sayısı	5,959,793
Tek bir işlem için tüketilen elektrikle 1 gün süreyle çalışan ABD hane sayısı	21.64
Bitcoin'in elektrik tüketimi / dünyanın elektrik tüketimi	0.29%
Yıllık karbon ayak izi (kt of CO2)	31,539
İşlem başına karbon ayakizi (kg of CO2)	313.74

Ağdaki madencilerin yarsından en az bir fazlasının kendi işlerine yarayan yeni kurallar oluşturup sistemi ele geçirmelerine %51 atağı denir. Consensus algoritması ne olursa olsun blokzinciri ağında bulunan düğümlerin işlemsel güç olarak %51 lik kısmı ortak karar verip

kendilerine göre yeni bir zincir üzerinden devam etmeye çalışabilirler. Bu duruma fork yani çatallaşma denir. Bir örnekle açıklamak gerekirse blokzincirdeki işlem gücünün %51 lik kısmı, A kişisinin B kişisine transfer etmek istediği kripto paranın transferini ortaklaşa karar verip engellemek ve bu parayı kendi aralarında paylaşmak isteyebilirler. Bu durumda işlemsel gücün azınlık tarafında kalan düğümlerin bloklarını ekledikleri zincir ile işlemsel gücün çoğunluk olduğu grubun eklediği zincir farkı olacaktır. Çatallaşmanın hard ve soft olarak iki çeşidi vardır. Hard forkta yeni kuralları kullanan uçlar için artık eski yapı geçersiz hale gelirken eski kuralları kullanan uçlar için de yeni kurallar bir anlam ifade etmemektedir. Bitcoin Cash (BCH) ve Bitcoin Gold (BTG) Bitcoinden (BTC) yapılmış hard fork örnekleridir. Ethereum Classic (ETC) ise Ethereumdan (ETH) yapılmış bir hard fork örneğidir. Soft fork, hard forktakinin aksine eski versiyonlarla çalışılabilir. Yani eski kurallarla üretilen bloklar yeni olan kurallar tarafından da kabul görür ve tek bir zincir ile yola devam edilir. Belirli süre geçtikten sonra ağdaki düğümlerin yeterli çoğunluğu yeni ayrışıma geçerek fork başarılı bir şekilde tamamlamış olur.

2.6.2. Proof of Stake

Bu algoritmaya göre sistem içerisindeki tüm coinler başlangıçta üretilir ve yaptıkları yatırımlara göre düğümlere coinleri verilir. Artık blokzinciri ağındaki düğümler sahip oldukları coin kadar güce sahiplerdir. Sahip oldukları paya göre bloğu oluşturacak düğümü belirlemek için iki yöntem vardır. Birinci yönteme göre toplam coin sayısına göre en yüksek payda coine sahip düğüm paylaşılacak ilk bloğu paylaşma hakkına sahiptir. Eğer belirtilen zaman içerisinde bu beklenen bloğu paylaşmazsa sıra ikinci en yüksek paya sahip düğüme geçer ve benzer durumlarda işlem bu şekilde devam eder. İkinci yönteme göre ise başlangıçta bir düğüm belirlemesi yapılmaz fakat PoW algoritmasına benzer şekilde düğümlere çözmeleri için problemler verilirken payı yüksek olan düğümlere çözülmesi daha kolay problemler verilerek bloğu ilk oluşturacak düğümün en fazla paya sahip olan düğüm olma olasılığını artırır. Daha fazla paya sahip olan düğümlerin daha az paya sahip düğümlere göre sürekli ve daha fazla blok yaratıp daha fazla gelir elde etmesinin bir nebze önüne geçebilmek ve az paya sahip düğümlerin de bloklar yaratabilmesini sağlayabilmek için bir yaş kavramı geliştirilmiştir. Bu kavrama göre yaş değeri yüksek olan coinlerin blok yaratma ihtimali daha yüksektir ve bir düğümün blok yaratmak için kullanılan coinlerinin yaş değerleri blok yaratıldıktan sonra sıfırlanır. Sıfırlanan coinler zamanla yeniden yaş değeri kazanmaya başlarlar ve bu işlemler bu şekilde tekrar ederek devam eder. PoW algoritmasındaki blok

üretme işlemine mining (madencilik) denirken PoS algoritmasında ise buna forging veya minting denilmektedir. PoW algoritmasında blok üreten düğümlere kazıcı (miner, madenci) denirken PoS algoritmasına göre blok üreten düğümlere validator (doğrulayıcı) denir. PoS algoritması PoW gibi zorluk derecesi yüksek hesaplamalar gerektiren bir algoritma olmadığı için PoW algoritmasında olması çok muhtemel olan donanım tabanlı merkeziyetçilik ve yüksek enerji tüketimi meydana gelmeyecektir. PoW algoritmasına göre blok üretmek isteyen düğümler bedel olarak aşırı elektrik tüketirler. PoS algoritmasına göre blok üretmek ve/veya doğrulamak isteyen düğümler ise başlangıçta bir akıllı sözleşmeye depozito bedeli olarak belirli miktarda Ether yüklemelidir. Bir düğüm bu yüklemeyi gerçekleştirildiği halde diğer depozito yüklemesi yapan düğümler ile blok üretmek için rekabete girebilirler. Eğer depozito bedelini yükleyen bir düğüm bir sahtekârlık yapmaya kalkarsa yüklediği tüm depozito bedelini kaybeder (Kiayias, Russell, David & Oliynykov, 2017).

2.6.3. Delegated Proof of Stake

DPoS algoritmasına göre işlemlerin boyutları, üretilen blokların aralık, ücretlendirme gibi blokzinciri ağının gerektirdiği tüm işlemler blokzincirde bulunan temsilcilerin (delegate) seçeceği tanıklar (witness) tarafından yapılır. Tanıklar blokzincirine blok eklemekle yükümlüdür ve bu sayede ödüllendirilirler. Her tanığın sadece bir oy hakkı vardır. Tanıklardan oyu en yüksek olan hangi düğüm ise en güvenilir düğüm odur denir ve blok yaratma önceliği bu tanığa verilir. Tanıklar düğümlerini sürekli çalışır vaziyette tutmalı ve ağ üzerindeki transactionları bloklara toplamalıdırlar. Topladıkları bu blokları dijital imzalayıp yayınlamalıdırlar. Transactionları onaylamalıdırlar. Uzlaşı ile ilgili bir sorun yaşanması durumunda DPoS algoritması bu sorunların adil ve demokratik olarak çözülmesini sağlar. Delegelerin herhangi bir transactionın ayrıntısını değiştirmeye gücü yoktur. Bir bloğa yazılacak işlemlerin doğrulanabilmesi için gereken yaklaşık süre 10 saniyedir. Genel bir deyişle demokrasinin dijital versiyonu olarak işleyen bir uzlaşı algoritmasıdır. PoS algoritmasına göre yeni bloğu yaratacak düğüm başlangıçta düğümlere dağıtılan coin miktarına göre belirlenirken DPoS algoritmasına göre kullanıcılar belirli bir sayıda tanık seçmek için oy kullanırlar. En çok oyu olan tanıklar transactionları doğrulamak ve blokları yaratmak için ödüllendirilirler. DPoS algoritması işlemleri doğrulayan ve yeni blokları yaratan düğümlerin az olması nedeniyle daha hızlı ve daha ölçeklenebilirdir. DPoS uzlaşı algoritmasını kullanan blokzincirlerine örnek olarak ARK.io, EOS, Lisk, STEEM gösterilebilir (Zheng vd., 2017).

2.6.4. Leased Proof of Stake

Bu uzlaşı algoritmasına göre temel PoS algoritmasında olduğu gibi sahip olunan paya göre işlem onaylama ve blok üretme önceliğinin olmasının yanı sıra düğümler sahip oldukları token'leri blokzincir ağındaki diğer düğümlerin kullanımına sunabiliyor. Buna bir nevi token kiralamak da denebilir (Salimitari & Chatterjee, 2018). LPoS kullanan en bilinir blokzincir projesi WAVES projesidir. Bu projeye göre Waves tokeni olan düğümler full düğüm olan diğer düğümlere sahip oldukları tokenleri kiralayarak kiraladıkları token miktarına göre Waves madenci tokenleriyle ödüllendirilir. Bir düğümün full düğüm olabilmesi için 1000 Wavese sahip olması gerekmektedir. WavesfullNode ve lunesfullnode Waves bakiyesi yüksek olan ve tokenlerini kendilerine kiralayanlara teşvik edici ödüller veren full düğümlere örnek olarak verilebilir.

2.6.5. Proof of Elapsed Time

PoET yüksek kaynak kullanımını ve yüksek enerji tüketimini önleyen ve adil bir çekiliş sistemi ile işlemleri daha verimli gerçekleştirilmesini sağlayan bir blokzincir uzlaşı algoritmasıdır. Bu algoritma genelde izin gerektiren blokzincir ağlarında madencilik haklarını ve blok yaratacakları belirlemek için kullanılır. Ağdaki her düğümün rastgele seçilen bir zaman aralığını beklemesi zorunludur. Belirlenmiş zaman aralığını bekleme işlemini bitiren ilk düğüm yeni bloğu kazanır. Ağdaki her düğüm rastgele bir bekleme süresi üretir ve bu süre boyunca uykuda bekler. Bekleme süresi en kısa olan yani en erken uyanan düğüm yeni bloğu blokzincirine işler ve ağdaki bütün düğümlere zorunlu bilgiyi yayınlar. Aynı işlemler sonraki blokların üretilmesi için tekrar edilir. Bu algoritmanın adilliğinin garantisi olan iki önemli faktör vardır. Birincisi ağdaki düğümlerin yeni bloğu kazanmak için bilerek ve isteyerek kısa bir bekleme zamanı seçmemelerini, bu bekleme zamanını gerçekten rastgele üretmelerini sağlamaktır. İkincisi ise kazananın gerçekten bekleme süresini tamamlamasını sağlamaktır (Chen, Xu, Shah, Gao, Lu & Shi, 2017).

2.6.6. Practical Byzantine Fault Tolerance

Asenkron sistemlerde çalışmak üzere tasarlanan bu uzlaşı algoritmasına göre ağda bulunan ve işlemleri doğrulamak için bekleyen her doğrulayıcı (validator) düğümün bir açık-özel anahtar çifti bulunur. Ağdaki her doğrulayıcı düğümde diğer bütün doğrulayıcı düğümlerin açık anahtar bilgisi mevcuttur. Ağa dâhil her düğüm kendisine gönderilen bir işlem bilgisini

kontrol ettikten sonra şayet bu işlemi onaylıyorsa kendi özel anahtarıyla imzalayarak ağdaki diğer düğümlerle paylaşır. PBFT modelinde temel olarak ağda bulunan düğümler bir tanesi birincil düğüm (lider) ve diğerleri backup düğümleri olmak üzere sıralanır. Ağdaki bütün düğümler birbirleriyle iletişim halindedir ve temel amaç ağdaki bütün dürüst düğümler ile sistemin durumu konusunda çoğunluk olarak ortak bir karara varmak ve anlaşmaktır. Birbirleriyle iletişim halinde olan düğümler hem kendilerine gelen mesajın ağdaki belirli bir düğümden geldiğini ispatlamalı hem de bu mesajın kendilerine iletilmesi süresince değişime uğramadığını doğrulamalıdır. Bu uzlaşının temel dört aşaması vardır. Birinci aşamada istemci lider düğüme bir hizmet işlemi için çağrıda bulunur. İkinci aşamada lider düğüm kendisine gelen bu isteği backup düğümlerine iletir. Üçüncü aşamada backup düğümleri kendilerine lider düğüm tarafından iletilen isteği uygularlar ve istemciye her biri birer cevap yollar. İstemci yeterli çoğunlukta aynı cevabı alana kadar bekler. Yeterli çoğunluğa ulaşan sonuç istemcinin isteğinin sonucudur. Çoğunluğa sahip dürüst düğümler lider düğümün hatalı olduğunu ve kaldırılması gerektiğini belirlediklerinde lider düğümü kaldırırlar ve yerine sıradaki lider düğümü getirirler (Castro & Liskov, 2003).

2.6.7. Simplified Byzantine Fault Tolerance

Bu algoritmaya göre, belirlenmiş bir blok üretici düğüm ona önerilen işlemleri toplar, doğrular ve bunları periyodik olarak yeni bir blok önerisi içerisinde bir araya getirir. Uzlaşı, düğümler ve blok imzalayıcılar tarafından kabul görmüş kuralları uygulayan bu generator(üretici) düğüm tarafından sağlanır. Diğer blok imzalayıcı düğümler önerilen bloğu imzalayarak onaylarlar. Ağdaki bütün üyeler blok imzalayıcıların kimliklerini bilmektedir ve ancak imzalayıcıların çoğunluğunun bir bloğu imzalaması halinde bu bloğu kabul ederler. Bir blok imzalayıcı her bloğun içerisinde bulunan bütün işlemleri doğrular. Sadece yeterli çoğunluk tarafından doğrulanmış bloklar zincire eklenir (Golan Gueta vd., 2018).

2.6.8. Delegated Byzantine Fault Tolerance

Bu algoritma bir ülke yönetim biçimi benzer şekilde tasarlanmıştır. NEO projesiyle popüler hale gelen bu algoritmanın anlaşılması için bilinmesi gereken kavramlar vardır. Bunları sıralamak gerekirse:

- Citizens (vatandaşlar): NEO token sahipleri (sıradan düğümler).
- Delegates (temsilciler): Defter tutan düğümler.
- Speaker (konuşmacı): Temsilciler içerisinde rastgele seçilen bir düğüm.

Vatandaşlar temsilcilere sahip oldukları tokenden bağımsız bir şekilde oy verebilirler. Temsilcilerin görevi vatandaşların bu blokzinciri ağında gerçekleştirmek istedikleri çeşitli işlemleri dinlemek ve bu işlemlerin ağdaki takibini yapmaktır. Oy verilen bu temsilcilerden biri rastgele konuşucu olarak seçilir. İş, bir bloğu doğrulama aşamasına geldiğinde bu konuşucu düğümün kendi bloğunu önermesi gerekmektedir. Konuşucu düğüm daha sonra bloğunu diğer bütün temsilci düğümlere ürettikleri blokların doğru olup olmadığını görmeleri için yollar. Bir bloğun kabul görüp zincire eklenmesi için temsilcilerin üçte iki çoğunluğu konuşucunun gönderdiği bloğu doğrulamalıdır. En az üçte iki çoğunluk sağlanamazsa yeni bir konuşucu rastgele seçilir ve aynı işlemler tekrar eder (Zheng, Xie, Dai, Chen & Wang, 2017).

2.6.9. Directed Acyclic Graphs (DAG)

DAG aslında bir blokzinciri örneği değildir. DAG da blok kavramı yoktur. Düğümlerden ve yönlü kenarlardan oluşan bir yapıdır. Bu yapıda her düğüm bir verinin bir parçasını temsil eder. Blokzincirinde bloklar birbirine tek yönlü bağlı halde tutulur ve bir bloktan diğer bir bloğa gitmenin en fazla bir yolu vardır. Fakat DAG yapısına göre bir düğümden başka bir düğüme gitmenin birden fazla yolu olabilir. DAG'da bir işlemin geçerli olabilmesi için bu işlemin rastgele iki farklı işlemi doğrulaması gerekmektedir. DAG' ın acyclic yani çevrimsiz olarak tanımlanmasının nedeni iki düğüm arasında iki yönlü ilişkinin olamayışıdır. Yani bir X bloğundan Y bloğu yönünde bir edge (kenar) mevcutsa Y bloğundan X bloğuna aksi yönde bir edge olması mümkün değildir fakat bir blok birden fazla blokla tek yönlü ilişkili olabilir. Kök (root) düğüm atası olmayan DAG düğümüne, yaprak (leaf) düğüm ise çocuğu olmayan DAG düğümüne denir. DAG algoritmasında topolojik sıralama kullanılır. Yani her kenar her zaman bir önceki kenardan sonrakine yönlendirilir. Blokzinciri ve DAG birer dağıtık defter örneğidir fakat blokzinciri doğrusal bir dağıtık defter iken yönlü çevrimsiz çizgeler ise birbirine bağlı bloklardan oluşan bir zincire sahip olmayan bir yapıdır. Kriptopara dünyasında blokzinciri ile dağıtık defterler eş anlamlıymış gibi kullanılsa da DAG da blokzinciri olmayan bir dağıtık defterdir (Tariq, Aadil, Malik, Ejaz, Khan & Khan, 2018).

DAG'ın blokzincire göre avantajlı olan yönlerini şu şekilde listeyebiliriz:

- DAG blokzincire kıyasla daha ölçeklenebilirdir ve dolayısıyla hızlıdır.
- DAG blok içermez. Doğrulama işlemlerini düğümler yapar.
- DAG, yapısı sayesinde bütün işlemler bütün düğümler tarafından en az bir kere doğrulandığı için yüksek güvenilirliklidir.

- Enerji tüketimi yoktur. Yani donanımsal merkeziyetçiliğe sebep olan PoW algoritmasına göre merkeziyetçilikten daha uzaktır.
- DAG küçük ödeme sistemleri için sıfıra yakın komisyon alması nedeniyle çok idealdir.

DAG'ın sistem çökmelerine karşı koruma ile ilgili bir bilgi içermemesi ise bu sistemin bir dezavantajıdır. IOTA, Byteball ve Dagcoin projeleri bu yapıyı kullanan en bilindik projelerdir.

2.6.10. Proof-of-Activity

PoW ve PoS uzlaşı algoritmalarının en iyi iki özelliği olan daha güvenilirlikli olması ve yüksek enerji tüketimi gerektirmemesi özelliklerinin birleştirilmesiyle oluşturulmuş hibrit bir algoritmadır. PoA algoritmasına göre madencilik işlemleri tıpkı PoW algoritmasındaki gibi başlar fakat aralarındaki fark PoW algoritmasında tam işlemler içeren bloklar kazarken (mine ederken) PoS algoritmasında madenciler sadece başlık bilgisi ve madenciler için ödül adresi içeren blok templateleri kazarlar. Madenciler bu blok templatelerini kazdıktan sonra sistem PoS algoritmasına geçer. Blok içerisinde başlık bilgisi rastgele bir düğüme işaret eder ve bu düğüm daha sonra daha öncesinde kazılmış blokları doğrulamakla görevlidir. Doğrulayacak blok sayısı fazla olan doğrulayıcıların blokları onaylama şansı artmaktadır. Doğrulama işleminden sonra bir blok zincire eklenebilir. Ağ madencilere ve doğrulayıcılara adil bir şekilde ödeme yapmaktadır (Bentov, Lee, Mizrahi & Rosenfeld, 2014).

2.6.11. Proof-of-Importance

PoS algoritmasının geliştirilmesiyle ortaya çıkan bir uzlaşı algoritmasıdır. NEM projesiyle duyulan bu algoritmada hasat kavramı vardır. Hasat mekanizması bir düğümün blokzincire eklenmek için uygun olup olmadığını belirlemek için kullanılır. Bir düğümde ne kadar çok hasat yapılırsa o düğümün zincire eklenme ihtimali o kadar artar. Buna skorlama sistemi denmektedir. NEM projesine göre 30 günlük periyot içinde gerçekleştirilen işlemler temel alınır Hasadı teşvik edici olarak işlem ücretleri hasadı yapan düğüme verilir. NEM projesinde hasat yapabilen bir düğümün için en az 10.000 NEM'e sahip olması gerekmektedir. PoI algoritması diğer düğümlerle aralarında işlemler yapan düğümleri ödüllendirir. Sistem sahte işlem teşebbüslerini tespit edebilmektedir (Bach, Mihaljevic & Zagar, 2018).

2.6.12. Proof-of-Capacity

PoW algoritmasının geliştirilmiş halidir. Madencilerin bulmaya çalıştıkları nonce değeri vardır. Düğümler madencilik yapmaya başlamadan önce hard disk alanlarını madencilik işlemlerine tahsis etmek zorundadırlar. Bu özellik PoC algoritmasını PoW algoritmasına göre daha hızlı kılar. PoW algoritmasında nonce değeri sürekli üretilip tahmin edilemeye çalışılırken PoC algoritmasına göre nonce değeri düğümlerin madenciliğe başlamadan önce ayırdıkları hafızalarında tutulan muhtemel çözümler arasında aranır. Düğümler ne kadar fazla çözüme sahiplerse doğal olarak madencilik yarışını kazanma ihtimalleri de bir o kadar fazla olacaktır. Proof of Capacity algoritması temel olarak iki aşamadan oluşmaktadır. Bunlar plotting ve mining aşamalarıdır. Plotting aşamasında mümkün olan tüm nonce değerlerin listesi, bir madenci hesabı da dâhil olmak üzere, verilerin tekrar tekrar birleştirilmesiyle oluşturulur. Her nonce 0-8191 aralığında olan 8192 hash değerine sahiptir. Bitişik hash değerleri ikili gruplara ayrılır. İkinci aşamada madencilik işlemine başlayan madenci ilk aşamada oluşturulmuş ikili hash gruplarına verilen bir scoop numarası (N) üreterek madenciliğe başlar. Madenci 1 numaralı nonce değerinin ürettiği N numaralı scoop numarasına gider ve deadline değerini hesaplamak için bu bu scooptaki veriyi kullanır. Bu işlem madencinin hard diskinde tutulan her nonce değeri için tekrarlanır. Bütün deadline hesaplamalarından sonra madenci tarafından minimum deadline seçilir. Deadline bir madencinin son blok üretiminden yeni bir blok üretmesine izin verilen zamana kadar geçen süreye denir. Eğer bu süre zarfında başka bir düğüm blok oluşturmazsa madenci bloğunu oluşturup oluşturduğu blok için ödül talep edebilir. PoC algoritması Android tabanlı sistemler de dâhil olmak üzere çeşitli sistemlerin kullanılmasına izin verir. ASIC tabanlı madencilik işlemlerinden 30 kata kadar daha az enerji tüketir. Burstcoin PoC algoritmasını kullanan bir kriptoparadır (Zheng, Xie, Dai & Wang, 2016).

2.6.13. Proof-of-Burn

PoB madencilere madencilik yetkilerini veren coinlerinin 'yakılması' veya 'yok edilmesi' ilkesine dayanır. PoW algoritmasının enerji kaybı olmayan versiyonudur denilebilir. Madencilere yaktıkları veya yok ettikleri coinleri oranınca blok üretme hakkı verilir. Madenci sanal olan madencilik donanımını almak için coinlerini yakar ve bu ona yaktığı coin oranında blok kazma gücü verir. Madenciler Bitcoin gibi alternatif blokzinciri paraları da yakabilirler ama ağın doğal kriptopara birimini yaktıklarında ödüllendirilirler. Yakılan coinlerin madencilere verdiği güç yeni bloklar kazıldıkça azalmaktadır (Zheng vd., 2016).

2.6.14. Proof-of-Weight

PoS algoritmasının büyük bir ilerleme kaydetmiş versiyonudur. PoWeight algoritmasına göre sahip olunan coin miktarının yanı sıra 'ağırlıklı faktörler' denilen özelliklere de bakar. Bu sistemin temel avantajları özelleştirilebilr ve ölçeklenebilir olmasıdır. Özelleştirilebilir olması 'ağırlıklı faktörler'in çeşitlendirilebileceği anlamına gelmektedir.

2.6.15. Uzlaşı Algoritmalarının Karşılaştırılması

Çizelge 2.2. Uzlaşı Algoritmalarının Karşılaştırılması (Bach, Mihaljevic & Zagar, 2018)

Uzlaşı Algoritması	Blokzincir Platformu	Çıkış Yılı	Programlama Dilleri	Akıllı Sözleşmeler	Artıları	Eksileri
PoW	Bitcoin	2009	C++	Yok	Düşük 51% atak ihtimali	Yüksek enerji tüketimi Madencilerin Merkezileşmesi
PoS	Ethereum	2015	Solidity	Var	Düşük Enerji Tüketimi Daha merkeziyetçilikten uzak	Nothing-at-stake sorunu
DPoS	Lisk	2016	Javascript	Yok	Düşük enerji tüketimi Ölçeklenebilir Artırılmış Güvenlik	Kısmen merkezi İki kez harcama saldırısı
LPoS	Waves	2016	Scala	Var	Adil Kullanım Coinleri kiralayabilme	Desentralizeleşme Sorunu
PoET	Hyperledger Sawtooth	2018	Python, JS, Go, C++, Java ve Rust	Var	Katılımın ucuzluğu	Özellikli donanım gerekliliği Açık blokzincirler için iyi değil
PBFT	Hyperledger Fabric	2015	JS, Python, Java, REST ve Go	Var	Onaylamaya ihtiyaç yok Enerji tüketim azlığı	İletişim Boşluğu Cybil Saldırısı
SBFT	Chain	2014	Java, Node ve Ruby	Yok	Yüksek güvenlik İmza ile	Açık blokzincirler için

					doğrulama	kullanılmaz
DBFT	NEO	2016	Python, .NET, Java, C, C++, Go, Kotlin, JS	Evet	Ölçeklenebilir Hızlı	Zincirdeki çakışmalar
DAG	IOTA	2015	Javascript, Rust, Java, Go, C++	Yapım aşamasında	Düşük maliyetli ağ Ölçeklenebilirlik	Uygulama boşlukları Akıllı sözleşmeler için uygun değil
PoA	Decred	2016	Go	Var	51% saldırısının olma ihtimalini azaltır Eşit katılım	Yüksek enerji tüketimi Çifte işaretleme
PoI	NEM	2015	Java, C++	Var	Hasat/hakediş İşlem ortaklığı	Desentralizeleşme sorunu
PoC	Burstcoin	2014	Java	Var	Ucuz Verimli Dağıtık	Daha büyükleri desteklem e Desentralizeleşme sorunu
PoB	Slimcoin	2014	Python, C++, Shell, JS	Yok	Ağın korunması	Kısa vadeli yatırımcılar için değil Coin ziyanı
PoWeight	Filecoin	2017	SNARK/STARK	Var	Ölçeklenebilir Özelleştirilebilir	Teşviklendirme sorunu

2.7. ICO, IPO, DAO ve DAICO Kavramları

ICO (Initial Coin Offering) projelerin geliştirilmesi için gerekli olan mali kaynağın yatırımcılar tarafından fonlanıp karşılık olarak yatırımcılara yatırım miktarlarına göre projeye özgü dijital coin verilmesine denir. Blokzinciri startupları projelerini oluşturup whitepaperlarını (teknik dokümanları) hazırladıktan sonra belirli bir süre aralığında projelerini fonlamak için ICO ya çıkarlar ve kişi veya kurumlardan kendi projelerine özgü coinleri

karşılığında para toplarlar. Proje sahipleri vaat ettikleri şeyleri yapmak zorunda değillerdir ve bunun sonucunda yasal bir işleme tabi tutulmazlar.

IPO (Initial Public Offering) şirketlerin hisselerini şirketin genişlemesi ve gelişmesi için para toplamak amaçlı halka arz edip satılması anlamına gelmektedir. ICO whitepaperları genellikle IPO prospektüslerine göre daha az titizdir. ICO' lar bir regülasyona tabi olmadıkları için IPO' lara nazaran daha az güvenilirdir.

DAO (Decentralized Autonomous Organization-Merkezi olmayan otonom organizasyonlar) şirketlerin veya diğer organizasyonların hiyerarşik bir yönetime ihtiyaç duymadan akıllı sözleşmeler içinde belirtilmiş kurallar aracılığıyla ortak bir amaç peşinde fikir birliğine vardıkları bir organizasyondur.

DAICO, DAO ve ICO özelliklerinin birleştirilmesiyle oluşturulmuştur. DAICO kavramı ICO'nun fon elde etme özelliğine ve DAO'nun yatırımcıların token tutup projenin geleceğine karar vermek için oy verme özelliğine sahiptir. ICO nun aksine DAICO da proje ekibi sadece yatırımcıların onayladığı ölçüde fon kullanabilir, aldıkları tüm parayı istedikleri gibi harcayamazlar. ICO lara yapılacak bir 51% atağında tüm fon kaybedilecekken DAICO'da yatırımcıların proje ekibine kullanım izni verdikleri kadar miktarda bir kayıp olacaktır. Bu da DAO'nun kaybı azaltıcı bir avantajı olarak söylenebilir. Yatırımcıların oylama işlemlerinin proje gelişim sürecini yavaşlatması da DAICO için bir dezavantaj olabilir.

Çizelge 2.3. ICO-IPO Karşılaştırması (Mitchell, 2019)

Özellik	**ICO**	**IPO**
Yasal Gözetim	YOK	KAPSAMLI
Kayıt ve Güvenilirlik Takibi	ZAYIF	GÜÇLÜ
Teklif Süresi	KISA	UZUN
Teklife Erişim	HERKESE AÇIK	HERKESE AÇIK DEĞİL

2.8. Blokzincirinin Dezavantajları

Blokzincirinin sağladığı avantajların yanısıra dezavantaj olarak değerlendirilebilecek zorlukları aşağıdaki gibi listelenebilir:

- **Ölçeklenebilirlik:** Blokzinciri ağlarında bulunan çok sayıdaki düğümlerin taleplerine cevap vermekte zorlanmakta ve buna bağlı yavaş işlem hızları sunup işlem başına

düşen yüksek ücretler talep etmektedir. Gelişim ve yaygınlaşma sürecini tamamlayamayan blokzinciri teknolojisinin ilerideki adaptasyon ve verimlilik sorunlarına blokzinciri tamamen yaygınlaşmadan çözümler üretilmelidir. Bitcoin Lightning Network projesi Bitcoin blokzincirindeki ölçeklenebilirlik sorununu ortadan kaldırmak üzere geliştirilmiş bir projedir (Burchert, Decker & Wattenhofer, 2018). Standart Bitcoin blokzincirinde bütün işlem verileri bloklarda tutulurken bu projeye göre transfer yapmak isteyen tarafların aralarında bir ödeme kanalı açılır. Kanal açıldıktan sonra kanal kapatma isteğine kadar yapılan hiçbir işlem bilgisi blokzincire eklenmez. Ancak ödeme kanalı kapatılırken tarafların mevcut bakiye bilgilerini içeren bir işlem blokta tutulur. Bu sayede blokzinciri eklenecek işlem verisi azaltılmış olur. Zilliqa isimli proje de blokzincirinin ölçeklenebilirlik sorununa çözüm olarak üretilmiştir (Tasatanattakool & Techapanupreeda, 2018).

- **Verimsiz Teknolojik Tasarımlar:** Ethereum'un akıllı sözleşme platformu geliştiricilere kendi DAPP' larını çalıştırma imkanı sunmaktadır. Singapur Ulusal Üniversitesi'nde yapılan bir araştırmaya göre büyük sayıdaki akıllı sözleşmeler kodlamalarından kaynaklı zayıflıklar içermektedir. Bitcoin blokzincirinde ise önemli önemsiz bütün işlem verilerinin tutulması bu ağın ağır ve yavaş olmasına neden olmaktadır (Tasatanattakool & Techapanupreeda, 2018).

- **Yüksek Enerji Tüketen Uzlaşı Mekanizmaları:** Blokzinciri uygulamalarının büyük bir çoğunluğunda PoW uzlaşı algoritması kullanılmaktadır. PoW algoritması yüksek hesaplamalar gerektiren bir algoritma olduğu için düğümlerin bir sonuca varmak için yüksek hesaplamalara bağlı yüksek enerji tüketmeleri gerekir (Tasatanattakool & Techapanupreeda, 2018).

- **Gizlilik ve Güvenlik Sorunları:** İzin gerektirmeyen blokzinciri ağlarında işlem verileri ve geçmişleri dünyada internet bağlantısı olan herkes tarafından görülebilirdir. Bu durum blokzincirinin şeffaflığı için iyi bir şey olmasına rağmen hasta kayıtları vb. bilgilerin blokzinciri üzerinde tutulduğu bazı durumlarda mahremiyet sorunları ortaya çıkabilir (Tasatanattakool & Techapanupreeda, 2018).

- **Maliyet:** Blokzinciri teknolojisi maliyetleri azaltmak için etkili bir araçtır. Transfer bedeli ile ilgili ücretleri azaltır ve operasyonel süreçleri hızlandırabilir. Fakat yeni bir teknoloji olduğu için eski sistemlere entegre edilmesi zor olacaktır. Yüksek hesaplama gerektiren uzlaşı mekanzimaları kullanan blokzinciri çözümleri tükettikleri enerji yüzünden yüksek maliyetler ortaya çıkmaktadır (Tasatanattakool & Techapanupreeda, 2018).

- **Doğru Teknolojiyi Seçmek:** Blokzinciri hizmetleri popüler olduğu için hangi tür ihtiyaçlara cevap vereceği, artıları-eksileri, hangi işletme tarafından hangi blokzinciri çözümünün ne için kullanıldığı çok araştırılmadan talep edilebilir. Bilgi kirliliği de kişileri veya kurumları buna sebep olabilir (Tasatanattakool & Techapanupreeda, 2018).

- **Düzenlemeler:** Blokzinciri uygulamalarının, onların dışında olmayan mevcut düzenleyici yapılarda çalışmaları gerekebilir, ancak bu, tüm endüstrilerdeki düzenleyicilerin teknolojiyi ve sektörlerindeki işletmeler ve tüketiciler üzerindeki etkisini anlamak zorunda oldukları anlamına gelir. Eğer düzenleyiciler blokzinciri tam anlayamazsa düzenlemelerdeki yanlışlıklar yüzünden çeşitli sorunlar ortaya çıkabilir (Girasa, 2018).

2.9. Mevcut ve Gelecekteki Muhtemel Blokzinciri Uygulama Alanları

Blokzincirinin mevcut ve gelecekteki muhtemel uygulama alanlarına aşağıdaki alanlar örnek olarak verilebilir (Lindman, Tuunainen & Rossi, 2017):

Teknoloji

Blokzincirinin etkilediği teknoloji sektörleri aşağıdaki başlıklar altında gruplandırılabilir:

- **Bulut Depolama:** Bulut depolama merkezi bir yapıda çalıştığı için efektif olmasına rağmen güvenlik tehditlerine karşı savunmasızdır. Blokzinciri ile birlikte kullanıcılar, bulut depolama alanına verilerinin kaybolmasını engelleyeceği merkezi olmayan bir sisteme geçildiği için daha fazla güvenebilirler. Storj, Filecoin, Siacoin ve Maidsafe blokzinciri tabanlı bulut depolama projeleridir.
- **Nesnelerin İnterneti:** Güven oluşturma, düşük maliyet ve işlemlerin daha hızlı gerçekleşmesi nesnelerin internetinde blokzinciri kullanmanın faydalarıdır. IBM spotlighted blokzinciri ve Hyundai Digital Asset Currency (HDAC) birer blokzinciri tabanlı nesnelerin interneti projesidir.
- **Güç ve Enerji Yönetimi:** Blokzinciri, ilk etapta enerjiyi nasıl depoladığımızı, dağıttığımızı ve üreteceğimizi değiştirecek. Şu anda, enerji tedariği söz konusu olduğunda birçok aracı kurum olduğu için enerji iletimi için yüksek ücret talep etmekteler. Blokzinciri ile birlikte aracılardan kurtulup maliyetlerin düşürülmesi öngörülmektedir. Conjoule ve Drift bu alanla ilgili örnek projelerdir.
- **Siber Güvenlik:** Blokzinciri sayesinde kurumlar verilerini merkezi olmayan bir şekilde şifreleyerek tutabilmektedir. Bu sayede verilerini hacker saldırılarından, DDoS ataklarından koruyabilirler. Blokzinciri tabanlı siber güvenlik projelerine Civic, Cambridge Blokzinciri ve Gladius projeleri örnek gösterilebilir.

Medya

Sürekli büyüyen bir sektör olan medya sektörü kendilerine avantaj sağlayan ve bu süreçte tonlarca gelir elde eden 3. şahıs olarak nitelendirebileceğimiz merkezi otoriteler nedeniyle tüketici veya yaratıcı dostu değildir. Blokzinciri sayesinde yüksek maliyetin aradan kalkmasıyla birlikte medya içerikleri kullanıcılar için daha erişilebilir hale gelebilir. Blokzincirinin etkilediği medya sektörelerini aşağıdaki gibi gruplandırabilirz:

- **Reklam:** Blokzinciri reklam sektörünü iki açıdan değiştirebilir:
    i. Yeni blokzinciri tabanlı tarayıcılarla birlikte kullancıların reklamları kapayıp gizliliklerini koruması mümkün hale gelmektedir.
    ii. Brave isimli tarayıcı blokzinciri teknolojisini kullanarak kullanıcıların ve reklam şirketlerinin Facebook, Google gibi aracılar olmadan P2P etkileşimlerini sağlamak üzere geliştirilmiş bir projedir.

- **Pazar Değerlendirmesi / Öngörü:** Blokzinciri dönüşümü, öngörmeyi tamamen yeni bir seviyeye kadar hızlandırabilir. Blokzinciri büyük miktarda veri tutması, makine öğrenmesi veya yapay zekâ ile birlikte kullanılabilirliğini göstermektedir. Ethereum platformunu kullanan Augur projesi buna bir örnektir.

- **Oyun Sektörü:** Günümüzde oyun sektörü bu sektöre milyonlarca dolar yatırım yapan şirketler tarafından domine edilmektedir. Blokzinciri sayesinde küçük oyun geliştiricileri oyunlarını izin veya ücret temelli sorunlardan bağımsız dünyayla buluşturabilirler. Bountie bunu yapan bir projedir**.**

Kanun ve Suçlar

Blokzinciri, verilerin bütünlüğünü korur ve dolayısıyla tüm kanıtları saklamak için harika bir platformdur. Aynı zamanda endişe duymadan dağıtılabilir ve emniyet uygulamasına güvenlik katmanını ekler (Elsden, Manohar, Briggs, Harding, Speed & Vines, 2018. Chronicled ve Elliptic bununlailgili projelerdir. Blokzinciri ile yasa dışı silah hareketlerinin önüne geçilebilir. Neutrino bununla ilgili bir projedir.

Ulaştırma

Ulaştırma endüstrisi işlem ve idari maliyetleri düşürme, daha düşük anlaşmazlıklar, genel ulaşım hızını iyileştirme konularında blokzinciri destekli projelerle iyileştirilebilir. RedCab isimli proje taksicilik sektörünün ve müşterilen sorunlarını blokzincirine entegre çalışan bir yazılımla çözüm üretmektedir.

Kamu Hizmetleri

- **Yönetim:** Hükümetin blokzinciri teknolojisini kullanmak istemesinin bir numaralı amacı, yönetişim haklarını elde etmek, uygun bir şeffaf oylama sistemine sahip olmak, vatandaş haklarını sağlamak, sistemdeki sahtekarlıkları en aza indirmek, yasa ve düzene bağlı karar verme sürecini akıllı sözleşmeler koşabilen blokzinciri çözümleriyle iyileştirebilirler.
- **Seyahat Sektörü:** Blokzincirinde bireylere verilecek eşsiz dijital kimlikler ile hükümetin bunları takibi ve yönetmesi kolaylaşacaktır.
- **Sağlık Sektörü:** Hükümetlerin asli görevlerinden olan vatandaşlara sağlık hizmeti sunmak blokzinciri ile birlikte daha sürdürülebilir, güvenilir, ölçeklenebilir, takip edilebilir hale gelebilir. Gem, Doc.AI ve Nebula Genomics sağlık sektörüne yönelik blokzinciri çözümlerine örnek olarak gösterilebilir.
- **Eğitim:** Online eğitim hizmetlerinde hizmeti talep eden ile hizmeti sunan arasında bulunan üçüncü şahıs olan merkezi otorite blokzinciri ile ortadan kalkacaktır. Hizmeti talep edenler daha ucuza hizmet alabileceklerdir.

İnsan Hakları ve Bağışlar

- **Bağışlar:** Blokzinciri yapılan bağışları şeffaf kılar. Ayrıca, bağış paralarının nereye gittiğini bilmek için kolay takip imkânı sağlar. Alice ve Giventh bu konuyla ilgili geliştirilmiş projelerdir.

- **Bilgi Edinme Hakkı:** Bilgi hakkı temel insan hakkıdır. Günümüzde çoğu insan bazı konularla ilgili bilgi edinmek için gereksiz bürokrasileri yerine getirmek ve sonra cevap almak için günlerce beklemek zorunda kalabilmektedir. Blokzinciri, uygun veri yönetimi ve erişimiyle uygun bir işlemi insanlar için kolaylaştırabilir.

Fintech

Blokzincirinin günümüz fintech teknolojilerine göre güvenlik, şeffaflık ve hızlılık açısından daha verimli olduğu su götürmez bir gerçektir (Guo & Liang, 2016). Ubiqucoin, Lendoit ve Kyber Network fintech ile ilgili blokzinciri çözümleridir.

Sözleşmeler

Sözleşmeler iki taraf arasındaki iş anlaşması, ödeme vb. gibi konulardaki şartları belirlemek ve tarafların bunlara bağlı kalarak hareket etmesini sağlamak için vardır. Sözleşmeleri üç alt kategoride incelemek mümkündür;

- **Miras**: Akıllı sözleşmeler sayesinde miras işlemleri blokzinciri platformuna taşınabilir. Miraslar akıllı sözleşmelerle insanlar öldükten sonra istediklerine dijital olarak transfer edilebilir hale gelebilir. Digipulse isimli bir proje buna bir örnektir.
- **Yasal Sözleşmeler:** Akıllı sözleşmelerle birlikte fikri mülkiyet hakları başkaları tarafından çalınma riskine karşı korunabilir.
- **Mülk ve Arazi:** Mülk ve arazilerinin sahiplikleri, fiyatları akıllı sözleşmelerle belirlenip güvence altına alınabilir. Harbor, ShelterZoom ve Ubiquity bu konuda geliştirilmiş projelerdir.

BÖLÜM 3

AKILLI SÖZLEŞMELER

Akıllı sözleşmeler, anlaşma şartlarının yerine getirildiği zaman kendiliğinden yürütülecek olan ve merkezi olmayan yapısından ötürü kendi kendini uygulayan, aracısız ve müdahaleye karşı korumalı olan blokzinciri uzlaşı mimarisine dayanan dijital programlar olarak tanımlanabilir (Christidis & Devetsikiotis, 2016). Geleneksel sözleşme şartlarına ve koşullarına ek olarak akıllı sözleşmeler dış kaynaklardan veri toplanması ve sözleşmede belirtilen şartlara göre işlenmesi gibi işlemlerin yanı sıra bu prosedürün sonuçlarına dayalı somut çözümler benimseme becerisine de sahiptir. Ethereum'da bir akıllı sözleşmeyi yerşleştirmek için sözleşmeyi blokzincire tanıtan özel bir yaratma transactionu çalıştırılır. Bu süreçte sözleşmeye 160-bitlik bir eşsiz adres atanır ve sözleşmenin kodu blokzincire yüklenir. Başarılı bir şekilde oluşturulan akıllı sözleşme, sözleşme adresi, sözleşme bakiyesi, önceden tanımlanmış yürütülebilir kod ve bir durumdan oluşur. Daha sonra farklı düğümler bilinen sözleşme adreslerine sözleşme çağırıcı işlemler göndererek belirli sözleşmelerle etkileşime girebilir. Sözleşme çağrısı yapan işlem, yürütme ücretini (fee) içermelidir ve aynı zamanda, işlemi çağırandan sözleşmeye bir Ether transferi de içerebilir. Ek olarak, bir fonksiyonun çağrılması için giriş verilerini de tanımlayabilir. Bir işlem kabul edildikten sonra, tüm ağ katılımcıları yani düğümler blokzincirinin mevcut durumunu ve işlem verilerini girdi olarak dikkate alarak sözleşme kodunu yürütür. Ağ, daha sonra uzlaşı protokolüne katılarak sözleşmenin bir sonraki durumunu ve sözleşme çıktısını kabul eder. Böylece Ethereum, her işlemden sonra durumunun güncellendiği işlem tabanlı bir durum makinesi olarak görülebilir (Buterin, 2014).

Akıllı sözleşmelerin arkasında yatan konseptin yaratıcısı olan Nick Szabo'ya göre de akıllı sözleşmelerin en ilkel türü basit bir otomasyon tabanlı otomat makineleridir. Otomat makineleri parayı alır ve istenilen ürünü satar. Otomat bunu ancak gerekli şartlar sağlandığı taktirde gerçekleştirir. Eksik miktarda para ile alınmaya kalkılan ürünler otomatın içindeki ilkel sözleşme ile korunur. Akıllı sözleşmelerin amacı içerdiği koşullara göre çıktılar üretmek ve muhtemel hataları ve bu hataların sebep olabileceği sonuçları en aza indirgemektir. Blokzinciri teknolojisini kullanan bir akıllı sözleşmede sözleşme şartları belirli bir programlama dilinde ifade edilir. Daha sonra blokzinciri platformuna aktarılan bu akıllı

sözleşme ön şartları sağlandığı halde üçüncü bir tarafa ihtiyaç duymadan otomatik olarak kendini çalıştırır (Buterin, 2014).

3.1. Akıllı Sözleşme Koşabilen Blokzinciri Platformları

3.1.1. Ethereum

Ethereum, akıllı sözleşmeler için en iyi standartlara ve en çok kullanıma sahip, Turing-complete bir programlama diline sahip olan blokzinciri platformudur. Ethereumun aşağıdaki ilkeleri sağlaması amaçlanmıştır (Buterin, 2014):

- **Basitlik:** Ethereum protokolünün ortalama bir yazılımcının bütün özelliklerini takip edip uygulayabileceği şekilde olmalıdır. Karmaşıklık getiren herhangi bir optimizasyon çok önemli bir fayda sağlamadığı takdirde mevcut sisteme eklenmeyecektir.
- **Evrensellik:** Ethereum bir programcının herhangi bir akıllı sözleşme veya matematiksel olarak tanımlanmış bir işlem tipi oluşturması için kullanabileceği dahili bir Turing-complete script dili içermektedir. Ethereum ile birlikte kendi finansal versiyonunuzu üretebilirsiniz. Kendi tokeninizi Ethereum akıllı sözleşmeleri ile birlikte yapabilirsiniz.
- **Modülerlik:** Ethereum protokolü ayrılabilecek en ideal modül sayısına göre tasarlanmıştır. Ethash, Modified Patricia Tree gibi birbirinden bağımsız yenilikler için farklı kütüphaneler kullanılmaktadır.
- **Çeviklik:** Ethereum protokolünün ayrıntıları değiştirilemez bir şekilde ayarlanmamıştır. Yüksek seviyedeki yapılarda kararlı değişiklikler yapılabilir. Ethereum protokol mimarisine veya EVM'e yapılacak geliştirmeler ölçeklenebilirliği ve güvenliği önemli bir ölçüde geliştirebilir.
- **Ayrımcılık Yapmama ve Sansürsüzlük:** Ethereum protokolü bazı özel kullanım kategorilerini kısıtlamayacak veya engellemeyecektir. Protokoldeki tüm düzenleyici mekanizmalar, zararı doğrudan düzenleyecek ve belirli istenmeyen uygulamalara karşı çıkmaya teşebbüs etmeyecek şekilde tasarlanmıştır. Yani isteyen biri gerekli ücreti ödediği takdirde sonsuz bir döngü içeren bir sözleşmeyi Ethereum platformunda koşturabilir.

Ethereum platformu sürekli güncellenmektedir ve kurucusu Vitalik Buterin'in belirttiğine göre son halini alması için tamamlaması gereken 4 ana safha vardır. Bunları aşağıdaki gibi listelenebilir:

- **Frontier**: Ethereum ilk çıktığında kullanılan halidir. Sloganları "Güvenli Bir Merkezi Olmayan Yazılım Platformu" olsa da güvenli değildir.
- **Homestead:** Frontier safhasının güvenlik problemleri giderildikten ve gerekli testleri yapıldıktan sonraki stabil ve güvenli haline verilen isimdir. Günümüzde bu aşamaya kadar gelmiş olan bir Ethereum platformu kullanılmaktadır.
- **Metropolis:** Büyük ölçüde tamamlanan bu aşama ise teknik kullanıma aşina olmayan kullanıcıların işini kolaylaştırmak için oluşturulacak bir arayüzden sonra yayınlanacaktır. Fakat bir önceki aşama olan Metropolis aşamasıyla aralarında Byzantium ve Constantinople olmak üzere iki ara aşama olması beklenmektedir.
- **Serenity:** Yüksek enerji tüketimine neden olan PoW algoritmasını kullanan Ethereumun bu aşamada farklı bir uzlaşı algoritmasına, muhtemelen PoS, geçmesi planlanmaktadır.

3.1.1.1. Ethereum Hesapları

Ethereumda hesaplar 20 bytelık adreslerden oluşur. Bir Ethereum hesabı aşağıdaki 4 bilgiyi tutar:

- Her transactionun sadece bir kere çalışmasını sağlamak amaçlı kullanılan bir sayaç olan nonce değeri
- Hesabın mevcut ether bakiyesi
- Eğer varsa hesabın sözleşme kodu
- Hesabın depo alanı (varsayılan olarak boştur)

Ether bu platform için kullanılan ana kriptoparadır. İşlem ücretlerini ödemek için kullanılır. İki tür Ethereum hesabı vardır:

- Harici sahipli hesaplar: özel anahtarlarla kontrol edilir
- Sözleşme hesapları: sözleşme kodlarıyla kontrol edilir

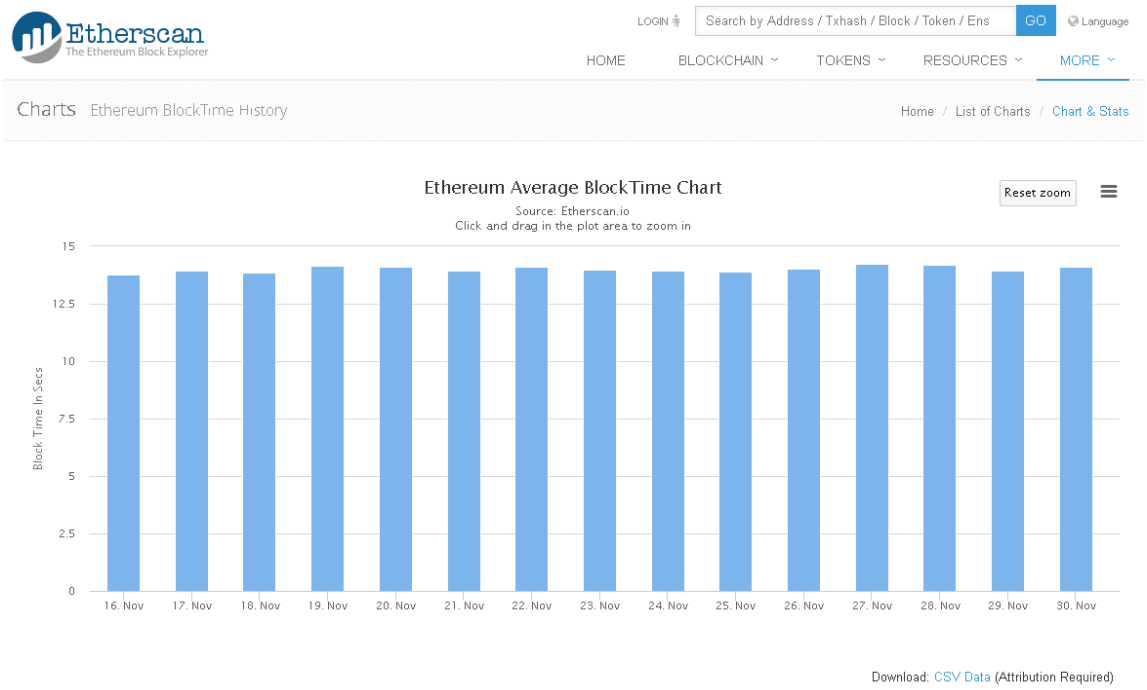
Harici sahipli hesaplar kod içermezler. Harici sahipli hesaplar bir transactionu yaratıp imzalayarak bir sözleşme hesabına gönderebilir. Sözleşme her mesaj aldığında kodu aktif hale gelir. Ethereumdaki akıllı sözleşmelerin yerine getirilmesi veya uyulması gereken kurallar olarak görülmemesi gerekmektedir. Bir mesaj veya işlem tarafından dürtüldüğünde özel bir kod parçasını çalıştıran kendi Ether bakiyeleri üzerinde doğrudan kontrol yetkisine sahip

otonom ajanlar olarak görülebilirler. Madencilik yapan Ethereum hesapları bloklarını şifrelemek için KECCAK-256 hash algoritmasını kullanmaktadır (Buterin, 2014).

Ethereumda blok büyüklüğü limiti değil gas limiti vardır. Farklı gas limitleri ile farklı sayıda işlem içerebilen blokların boyutları dolayısıyla değişkenlik göstermektedir. Aşağıda belirli bir tarih aralığında Ethereum ağına eklenen blokların boyutlarını gösteren bir grafik verilmiştir.

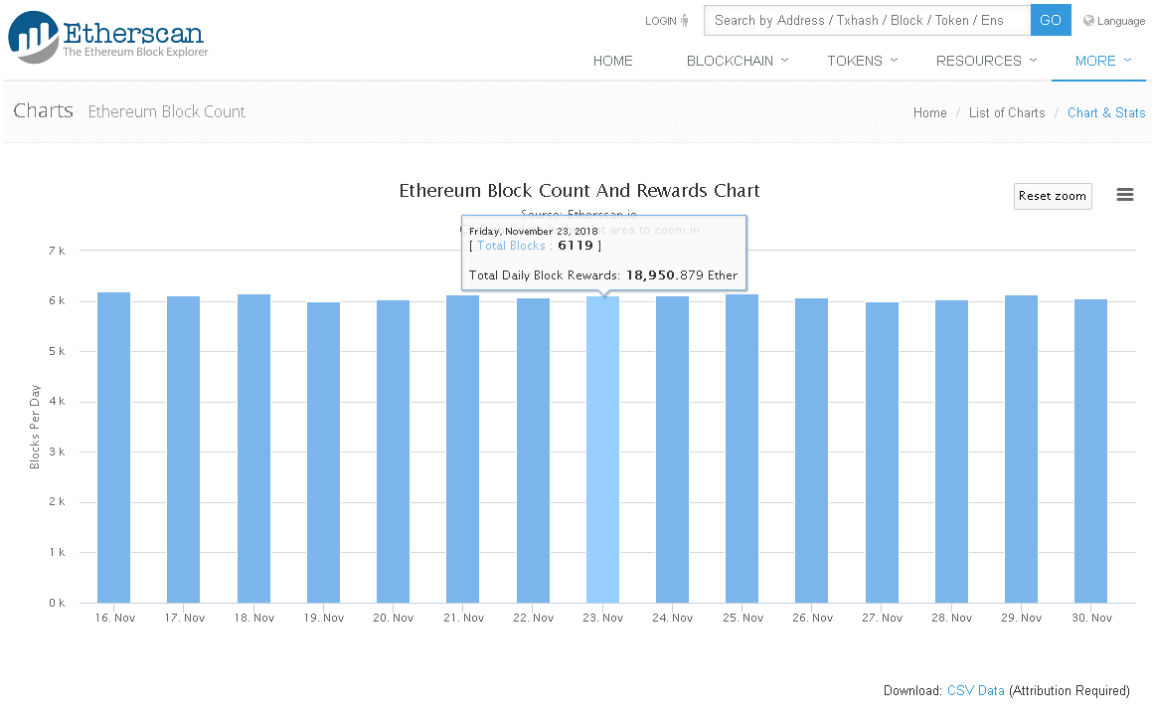
Şekil 3.1. Ethereum Blok Boyutları (https://etherscan.io/)

Farklı gas limitlerine ve boyutlara sahip Ethereum bloklarının oluşturulması için geçen zamanlar da farklılık göstermektedir. Aşağıda belirli zaman aralıklarında oluşturulan blokların oluşturulması için geçen ortalama süreleri içeren bir grafik verilmiştir.



Şekil 3.2. Ethereum Blok Oluşturulma Süreleri (https://etherscan.io/)

Aşağıda Ethereum Blokzincirine günlük eklenen blok sayılarını ve bu blokların oluşturulması için ödenen toplam ödülleri gösteren bir grafik verilmiştir.



Şekil 3.3. Ethereum Madencilerine Ödenen Ödüller (https://etherscan.io/)

3.1.1.2. Mesajlar ve İşlemler (Messages and Transactions)

Transaction terimi, Ethereum'da harici sahipli hesaplardan gönderilen ve içinde mesaj barındıran imzalanmış veri paketi için kullanılmıştır. İki tür işlem vardır. Birinde mesaj çağrılarıyla sonuç alınırken diğerinde kod ile ilişkili hesaplar yani sözleşmeler yaratılmaktadır. İki türün de sahip olduğu ortak alanlar aşağıda listelenmiştir:

- **Nonce**: gönderici tarafından gönderilen işlem sayısıdır. Tn ile temsil edilir.
- **gasPrice**: İşlemi gönderenin her hesaplama adımı başına ödeyeceği ücreti temsil eden bir değer. Tp ile temsil edilir.
- **gasLimit**: bir işlemi çalıştırmak için kullanılabilecek maksimum gas miktarıdır. Tg ile temsil edilir.
- **To**: mesaj çağrısının 160 bitlik hesap adresine sahip alıcısı
- **Value**: Göndericiden alıcıya gönderilecek Wei miktarı. Tv ile temsil edilir.
- **V, r, s** : Mesajı göndereni belirten bir imzayı oluşturan değerler. Tv, Tr ve Ts ile temsil edilmektedir.

Bir sözleşme yaratmak için gerçekleştirilmiş bir işlemde bu alanlara ek olarak init ve data alanları tutulur. init alanı hesabın başlatması için EVM kodunu belirten sınırsız boyutlu bayt dizisidir. İnit, hesaba her mesaj çağrısı geldiğinde çalışan bir EVM kod parçasını döndürür. İnit hesap yaratıldığında bir kere çalışır ve hemen atılır. Ti ile temsil edilir. Data alanı ise mesaj çağrısının giriş verisini belirten sınırsız boyutlu byte dizisidir. Td ile temsil edilir.

Kazara oluşan veya düşmanca gerçekleştirilen sonsuz döngüler veya kod içerisindeki diğer hesaplama kaynaklı israfları önlemek için her transactiona ne kadar hesaplamalı adım uygulayabileceğini belirten bir limit verilmiştir. Hesaplamanın temel birimi "gas" dır. Genellikler bir hesaplamalı adımın maliyeti bir gastır fakat bazı durumlarda hesaplamaların pahalılığı veya durumun bir parçası olarak depolanması gereken verinin miktarının artması nedeniyle adım başına düşen gas maliyeti 1 gastan fazla olabilir. Transaction verisindeki her bir byte için 5 gas ücret bulunmaktadır. Ücretlendirme sisteminin amacı bir saldırganın hesaplama, bant genişliği ve depolama gibi tükettiği her kaynak için orantılı ödeme yapmasını sağlamaktır. Bu kaynakların daha fazla tüketimine neden olan bir işlem tüketimine oranla daha fazla bir ücret ödemelidir (Buterin, 2014).

İşlemin Çalıştırılması

Ethereum protokolünün en karmaşık kısmı bir işlemin çalıştırılmasıdır. Çalıştırılan her işlemin esas geçerliliğinin test edildiği bir başlangıç testinden geçtiği varsayılmaktadır. Bu test aşağıdaki maddelerden oluşmaktadır (Buterin, 2014):

- İşlem iyi oluşturulmuş, takip eden byteları olmayan bir RLP çıktısı olmalıdır.
- İşlemin imzası geçerli olmalıdır.
- İşlemin nonce değeri işlemi gönderenin hesabının o anki nonce değerine eşit yani geçerli olmalıdır.
- Gas limiti işlem tarafından kullanılan gas miktarından az olmamalıdır.
- İşlemi gönderenin bakiyesi en az ön ödeme için gerekli maliyet kadar olmalıdır.

Y durum geçiş fonksiyonu, T işlem, σ durum ve σ' işlem sonrası durum olmak üzere;

$$\sigma' = Y(\sigma, T)$$ 'dir.

RLP fonksiyonu Ethereumdaki içiçe diziler gibi karmaşık yapıdaki verilerin, önüne veri tipi ve asıl veri ile offsetin uzunluğunu belirten bir prefix(önek) byte'ı ekleyerek seri hale getirilmesi ve bu halden geriye dönülmesi için geliştirilmiş bir encoding/decoding algoritmasına sahiptir. Bu fonksiyon decoding aşamasında ilk önce giriş verisinin ilk byte'ına bakarak veri tipi ve asıl veri ile offsetin uzunluğunu analiz eder. Verinin tipine ve offsetine göre veriyi decode eder. Aynı işlem serileştirilmiş diğer veriler için devam eder (Buterin, 2014).

İşlem yürütme süresince işlemi anlık takip eden bilgiler toplanır. Toplanılan bu bilgilere işlem altdurumları(substate) denir ve **A** ile temsil edilir.

- **A**, 4 bileşenden oluşmaktadır: **A ≡ (As, Al, At, Ar)**
- **As**: işlemin tamamlanmasından sonra kullanılmayacak hesapların tutulduğu bir kümedir.
- **Al**: VM kodunun yürütülmesinde bulunan, sözleşme-çağrılarının Ethereum dünyasının dışındakiler (örneğin bir Dapp arayüzü) tarafından kolayca izlenebilmesine olanak sağlayan arşivlenebilir ve indekslenebilir 'kontrol noktaları'ndan oluşan bir log serisidir.
- **At**: dokunulan, iletişime geçilen, boş olanlarının işlemin sonunda silindiği hesapların bir kümesidir.
- **Ar**: Geri ödeme bakiyesidir.

Mesaj Çağrısının Çalıştırılması

Bir mesaj çağrısının çalıştırılması için gönderici (**s** - sender), işlem yaratıcısı(**o** - transaction originator), alıcı (**r** - recipient), kodu çalıştırılacak hesap(**c**, genelde alıcı ile aynıdır), mevcut gas(**g** - available gas), değer (**v** - value), gas ücreti(**p** - gas price), mesaj çağrısının giriş verisiyle birlikte keyfi uzunluktaki bir byte arrayi (**d**), mesaj-çağrısı/sözleşme-yaratma yığınının derinliği (**e**) ve durumda değişiklik yapabilmek için izin (**w**) verileri gerekmektedir. Bir işlemin yeni durumunu ve alt durumunu değerlendirmenin yanı sıra, mesaj çağrıları ekstra bir bileşen olarak ***o*** ile temsil edilen çıkış verisinin tutulduğu byte dizisine sahiptir. Bu *o* verisi işlemler çalıştırılırken yok sayılır.

Yz işlemin durum kodu olmak üzere $Y_z(\sigma, T) \equiv z$ dir.

Yukarıdaki veriler ışığında aşağıdaki gibi hangi giriş verilerinin bir fonksiyona sokulmasıyla hangi çıkış verilerinin hesaplandığı bir denklem yazmak doğru olacaktır:

$$(\sigma' , g' , A, z, o) \equiv \Theta(\sigma, s, o, r, c, g, p, v, d, e, w)$$ (Buterin, 2014)

Yürütme Modeli

Yürütme modeli, verilen bytecode komutları ve çevresel veriler kümesinin sistemin durumunu nasıl değiştirdiğini göstermektedir. Bu sanal durum makinesinin resmi bir modeli olan Ethereum Sanal Makinesi (EVM) ile belirtilir. Verilen bir gas değeri bu makinede gerçekleşecek hesaplama miktarını sınırlayacaktır (Buterin, 2014).

Temeller

EVM basit stack tabanlı bir mimariye sahiptir. Makinenin word boyutu ve dolayısıyla stack elemanlarının boyutu Keccak-256 hash algoritmasına ve eliptik eğri hesaplamalarına uyması için 256 bit olarak belirlenmiştir. Hafıza modeli word-adreslenmiş byte dizisidir. Word birden fazla byte içeren veri yapısıdır. EVM stack boyutu maksimum 1024'tür (Buterin, 2014).

3.1.1.3. Ethereumda Ücretler

Ethereumda işlemlerin alt elemanları olan operasyonların çalıştırılması için 3 ön zorunluluk temel alınarak ücretlendirme yapılır. İlk ve en yaygın olanı operasyonun hesaplanmasına özgü ücretlendirmedir. Ethereumda her opcode'un farklı gas maliyeti vardır. Ücretlendirme yapılırken de EVM kodunda bulunan opcodeların kullanım adetleriyle gas maliyetleri çarpılıp toplanarak bir sonuca varılır. Opcode'lar maliyetlerine göre 6 farklı gruba ayrılmıştır (https://github.com/djrtwo/evm-opcode-gas-costs). Bu gruplar aşağıdaki tabloda soldan sağa azalan hızlara göre listelenmiştir:

Çizelge 3.1. EVM Opcode Maliyet Grupları

	Temel	**En Hızlı**	**Hızlı**	**Orta Hızlı**	**Yavaş**	**Ekstra**
Gas Used	2	3	5	8	10	20
	ADDRESS	DUP	MUL	ADDMOD	JUMPI	BLOCKHASH
	ORIGIN	SWAP	DIV	MULMOD	EXPBASE	BALANCE
	CALLER	PUSH	MOD	JUMP		EXTCODESIZE
	CALLVALUE	ADD	SDIV			EXTCODECOPYBASE
	CALLDATASIZE	SUB	SMOD			
	CODESIZE	LT	SIGNEXTEND			
	GASPRICE	GT				
	COINBASE	SLT				
	TIMESTAMP	SGT				
	NUMBER	EQ				
	DIFFICULTY	AND				
	GASLIMIT	OR				
	POP	XOR				
	PC	NOT				
	MSIZE	BYTE				
	GAS	CALLDATALOAD				

		CALLDATACOPY				
		CODECOPY				
		MLOAD				
		MSTORE				
		MSTORE8				

Aşağıdaki tabloda opcodeların gas maliyetleri artan sırada verilmiş hexadecimal değerlere göre listelenmiştir (https://github.com/djrtwo/evm-opcode-gas-costs):

Çizelge 3.2. EVM Opcode Gas Maliyetleri

Değer	**OPCODE**	**Gas Used**	**Grubu**	**Açıklama**
0x00	STOP	0	sıfır adım	Yürütmeyi Durdurur
0x01	ADD	3	En Hızlı	Ekleme işlemi
0x02	MUL	5	Hızlı	çarpma işlemi
0x03	SUB	3	En Hızlı	çıkarma işlemi
0x04	DIV	5	Hızlı	bölme işlemi
0x05	SDIV	5	Hızlı	işaretli tamsayı bölme
0x06	MOD	5	Hızlı	modulo kalan işlemi
0x07	SMOD	5	Hızlı	işaretli modulo kalan işlemi
0x08	ADDMOD	8	Orta Hızlı	modulo ekleme işlemi
0x09	MULMOD	8	Orta Hızlı	modulo çarpma işlemi
0x0a	EXPBASE	10	Yavaş	üstel operasyon
0x0b	SIGNEXTEND	5	Hızlı	işaretli tamsayının two's complementini genişletir
0x10	LT	3	En Hızlı	less-than(-den/dan az)
0x11	GT	3	En Hızlı	greater-than(-den/dan fazla)
0x12	SLT	3	En Hızlı	signed less-than
0x13	SGT	3	En Hızlı	signed greater-than
0x14	EQ	3	En Hızlı	equality(eşitlik)

0x15	ISZERO	3	En Hızlı	basit NOT operatörü
0x16	AND	3	En Hızlı	bitwise AND operatörü
0x17	OR	3	En Hızlı	bitwise OR operatörü
0x18	XOR	3	En Hızlı	bitwise XOR operatörü
0x19	NOT	3	En Hızlı	bitwise NOT operatörü
0x1a	BYTE	3	En Hızlı	bir kelimeden tek byte çeker
0x20	SHA3BASE	30		Keccak-256 hashini hesaplar
0x30	ADDRESS	2	Temel	mevcut çalışan hesabın adresini verir
0x31	BALANCE	20	Ekstra	verilen hesabın bakiyesini döndürür
0x32	ORIGIN	2	Temel	işlemi gönderenin adresini döndürür
0x33	CALLER	2	Temel	çağıranın adresini verir
0x34	CALLVALUE	2	Temel	mevcut işlemin yürütülmesi için işlemle verilen depozito değerini verir
0x35	CALLDATALOAD	3	En Hızlı	mevcut ortamın giriş verilerini verir
0x36	CALLDATASIZE	2	Temel	mevcut ortamın giriş verilerinin boyutunu verir
0x37	CALLDATACOPYBASE	3	En Hızlı	mevcut ortamın giriş verisini hafızaya kopyalar
0x38	CODESIZE	2	Temel	mevcut ortamda koşan kodun boyutunu verir
0x39	CODECOPYBASE	3	En Hızlı	mevcut ortamda koşan kodun boyutunu verir
0x3a	GASPRICE	2	Temel	mevcut ortamdaki gasın birim fiyatını verir
0x3b	EXTCODESIZE	20	Ekstra	bir hesabın kodunun boyutunu verir
0x3c	EXTCODECOPYBASE	20	Ekstra	bir hesabın kodunu hafızaya kopyalar

0x3d	RETURNDATASIZE	3		mevcut ortamdaki bir önceki çağrıdan oluşan çıkış verisinin boyutunu verir
0x3e	RETURNDATACOPY			bir önceki çağrının çıkış verisini hafızaya kopyalar
0x40	BLOCKHASH	20	Ekstra	en son tamamlanan 256 bloktan birinin hash değerini verir
0x41	COINBASE	2	Temel	bloğu oluşturanın adresini verir
0x42	TIMESTAMP	2	Temel	bloğun zaman damgasını verir
0x43	NUMBER	2	Temel	bloğun numarasını verir
0x44	DIFFICULTY	2	Temel	bloğun zorluğunu verir
0x45	GASLIMIT	2	Temel	bloğun gas limitini verir
0x50	POP	2	Temel	stackten bir item çıkarır
0x51	MLOAD	3	En Hızlı	hafızadan bir word yükler
0x52	MSTORE	3	En Hızlı	hafızaya bir word kaydeder
0x53	MSTORE8	3	En Hızlı	hafızaya byte kaydeder
0x54	SLOAD	50		storage tan word yükler
0x55	SSTORE	20000		storage a word kaydeder
0x56	JUMP	8	Orta Hızlı	program sayacını değiştirir
0x57	JUMPI	10	Yavaş	program sayacını duruma bağlı değiştirir
0x58	PC	2	Temel	program sayacının değerini verir
0x59	MSIZE	2	Temel	aktif bir hafızanın byte cinsinden boyutunu verir
0x5a	GAS	2	Temel	Bu komutun maliyetine karşılık gelen indirgeme dahil olmak üzere mevcut gazın miktarını verir
0x5b	JUMPDEST	1		zıplamak için geçerli bir varış

				yeri işaretler
0x60 -- 0x7f	PUSH1-PUSH32		En Hızlı	PUSHN stack içine N kadar item koyar
0x80 -- 0x8f	DUP1 - DUP16		En Hızlı	DUPN stackin N. Elemanını duplike eder
0x90 -- 0x9f	SWAP1-SWAP16		En Hızlı	SWAPN stackin 1. elemanıyla N. Elemanını değiştir
0xa0 -- 0xa4	LOG0 - LOG4			LOGN N başlıklı bir log kaydı ekler
0xf0	CREATE	32000		ilişkili kod ile yeni bir hesap oluşturur
0xf1	CALL			bir hesaba mesaj çağrısı
0xf2	CALLCODE	0	sıfır adım	hesaba alternatif bir hesabın koduyla mesaj çağrısı gönderir
0xf3	RETURN	0	sıfır adım	yürütmeyi çıkış verisini döndürerek durdurur
0xf4	DELEGATECALL			Bu hesaba alternatif bir hesabın koduyla mesaj gönderir, ancak gönderenin ve değerin mevcut değerlerini aynı tutar
0xfa	STATICCALL	3		bir hesaba statik mesaj çağrısı
0xfd	REVERT			yürütmeyi durdurur, durum değişikliklerini geri alıp veriyi ve kalan gası döndürür
0xfe	INVALID	21000		belirlenmiş geçersiz komut
0xff	SELFDESTRUCT	68		Yürütmeyi durdurur ve daha sonra silinmesi için hesabı kaydeder

Yukarıdaki tablolarda görüldüğü gibi farklı opcodeların yürütülme maliyetleri değişkenlik göstermektedir. Akıllı sözleşmeleri akıllı yapan sadece günümüz sözleşmelerinin dijitalleşmiş halleri olması değil az maliyetle aynı işlemi yukarıdaki tablodaki ücretlendirmelere bakarak daha akıllıca programlanmasıdır.

3.1.1.4. Akıllı Sözleşme Kodunun Çalıştırılması

Ethereum akıllı sözleşmeleri düşük seviye programlama dilleriyle yazılabildiği gibi EVM koduna dönüştürülebilir yüksek seviye programlama dilleri ile de yazılabilir. EVM kodu her bit byteın bir operasyonu temsil ettiği bytelardan oluşan bir seridir. Genel olarak, kodun çalıştırılması mevcut program sayacının gösterdiği byteın ifade ettiği operasyonu sonsuz döngüde tekrarlanmasıdır. Daha sonra program sayacı kodun sonuna ulaşana kadar veya bir hata alana kadar veya RETURN veya STOP komutları görülene kadar bir arttırılır. İşlemler veriyi saklamak için üç farklı alan türüne erişebilir (Buterin, 2014):

- Yığın (**stack**), hangi değerler push veya pop edileceğini belirleyen son giren ilk çıkar mantığı ile çalışan bir veri tipi
- **Memory** (hafıza), sonsuza kadar genişleyebilir byte dizileri
- Sözleşmenin uzun vadeli depolama için kullanılan **storage** alanı vardır. Stack ve memory gibi hesaplamanın bitmesiyle kendini resetlemez, uzun süre kalıcıdır.

Kod blok başlık verisine erişebildiği gibi değere, göndericiye ve gelen mesaj verisine de erişebilir. Çıktı olarak da byte dizisi döndürebilir. Sistem durumu σ ve hesaplama için geriye kalan gas miktarı g ye ek olarak kodun veya işlemin yürütülme ortamında kullanılan ve yürütme sözleşmelerinin sağlaması gerektiği önemli bilgiler vardır. I kümesinde toplanılan bu bilgiler aşağıdaki gibidir:

- Ia: yürütülen kodun sahibi olan hesabın adresidir
- Io: yürütmeyi başlatan işlemi gönderenin adresidir
- Ip: yürütmede olan işlemin gas fiyatıdır.
- Id: Yürütmeye giriş verisi olan bir byte dizisidir. Eğer yürütmeyi yapan (yürütme aracısı) bir işlemse bu byte dizisinde işlem bilgileri olacaktır.
- Is: kodun yürütülmesine neden olan hesabın adresidir. Eğer yürütmeyi yapan (aracı) bir işlemse bu işlemi gönderen adrestir yani Io ile aynıdır.
- Iv: Yürütme ile aynı prosedürün bir parçası olarak hesaba geçirilen Wei cinsinden bir değer. Eğer yürütme aracısı bir işlemse bu işlem değeri (tx value) olacaktır.
- Ib: yürütülecek bir makine kodu olan bir byte arrayi

- IH: mevcut bloğun blok başlığı
- Ie: mevcut mesaj çağrısının veya sözleşme-yaratımının derinliğidir (örneğin: yürütülecek CALL veya CREATE komutlarının sayısı)
- Iw: durumda değişiklik yapmak için izin verilmesi
- Ξ : yürütme modeli fonksiyonu, σ' sonuç durumu, g' kalan gas, A altdurumlar ve o sonuç çıktısı olmak üzere

$$(\sigma' , g', A, o) \equiv \Xi(\sigma, g, I)$$ (Buterin, 2014)

3.1.1.5. Ethereum Blok Yapısı

Bir Ethereum bloğunda blok başlığı (block header-**H**), işlemleri (transactions-**T**) ve diğer blokların başlıklarından oluşan bir kümeyi içerir (**U**). U' da başıkları olan blokların ebeveynleri (parent) bulunulan bloğun ebeveyninin ebeveynine (parent's parent) eşit seviyede olmalıdır. Böyle bloklara ommers denir. Bir blok başlığı aşağıdaki alanlardan oluşmaktadır (Buterin, 2014):

- **parentHash**: ebeveyn bloğun başlığının 256 bitlik Keccak hash algoritmasıyla hashlenmiş halidir. Hp ile temsil edilir.
- **ommersHash**: bloğun ommer bloklarının listesinin Keccak 256 bit ile hashlenmiş halidir. Ho ile temsil edilir.
- **Beneficiary**: bloğun başarılı üretiminden toplanan ücretlerin transfer edileceği 160-bitlik adrestir. Hc ile temsil edilir.
- **stateRoot**: bütün işlemler gerçekleştirildikten ve sonlandırma uygulandıktan sonra, oluşan durum ağacının kök düğümünün Keccak 256-bit ile hashlenmiş halidir.
- **transactionsRoot**: bloğun işlem listesi kısmındaki işlemlerle doldurulmuş ağaç yapısının kök düğümünün Keccak 256-bit ile hashlenmiş halidir. Ht ile temsil edilir.
- **receiptsRoot**: bloğun işlem listesi kısmındaki işlemlerin alıcılarıyla doldurulmuş ağaç yapısının kök düğümünün Keccak 256-bit ile hashlenmiş halidir. He ile temsil edilir.
- **logsBloom**: işlemler listesindeki her işlemin alıcısında tutulan her log girişinin içerdiği, indekslenebilen, loglayanın adresi ve log başlığından oluşan Bloom filtresidir. Hb ile temsil edilir.
- **Difficulty**: bloğun zorluk derecesini gösteren skaler bir değer. Bir önceki bloğun zorluk derecesi ve zaman damgası ile hesaplanabilir. Hd ile gösterilir.
- **Number**: Ata blokların sayısını gösteren sklaer bir değerdir. Örneğin genesis bloğu için bu değer 0(sıfır)'dır.
- **gasLimit**: blok başına düşen gas harcama limitidir. Hl ile temsil edilir.

- **gasUsed**: blok içeriisindeki işlemler için kullanılan toplam gas miktarıdır. Hg ile temsil edilir.
- **Timestamp**: Unix'in time() fonksiyonunun bloğun başlangıcında döndürdüğü değerdir. Hs ile temsil edilir.
- **extraData**: en fazla 32 byte büyüklüğünde olan ve blokla ilişkil ekstra verileri tutmaya yarayan alandır. Hx ile temsil edilmektedir.
- **mixHash**: nonce değeri ile birleştirilen mevcut blokta yeterli miktarda iş yapıldığını kanıtlamaya yarayan 256-bitlik bir hash. Hm ile temsil edilir.
- **Nonce**: mixHash ile birleştirilen ve mevcut blokta yeterli miktarda iş yapıldığını kanıtlamaya yarayan 64-bitlik bir değer. Hn ile temsil edilir.

Blokzinciri tüm blok ağacından kökten yapraka giden bir yoldur. Hangi yolda olduğuna dair uzlaşıya varılması için üzerinde en fazla hesaplamayı yapan yol veya en ağır yol seçilir. En uzun yolu belirlemenin en iyi yolu yaprağın blok numarasına bakmaktır. Yaprağın blok numarası genesis bloğu hariç o yolu oluşturan kaç adet düğüm olduğunu gösterir. Yol ne kadar uzunsa, yaprağa varmak için yapılması gereken toplam madencilik çabası o kadar büyük olur. Blok başlığı zorluk (difficulty) bilgisini içerdiği için blok başlığı hesaplamanın yapıldığının doğrulanması için yeterlidir. Zincirdeki her blok o zincirin zorluğuna katkıda bulunur. B blok, Bt toplam zorluk, B' parent blok ve Bd de parent bloğun zorluğu olmak üzere B bloğunun toplam zorluğu aşağıdaki gibi hesaplanmaktadır (Buterin, 2014):

$$Bt = B't + Bd$$

Bir bloğu sonlandırma işlemi dört aşamadan oluşur:

- Ommerler onaylanır veya madencilik yapılıyorsa belirlenir. Ommer başlıklarının onaylanması, bu başlığın geçerli olması ve mevcut bloğun N<=6 olmak üzere N. Nesil ommeri olması anlamına gelmektedir.
- İşlemler onaylanır veya madencilik yapılıyorsa belirlenir. Verilen gasUsed miktarı blok içerisinde kullanılan toplam gas miktarından az olmamalı ve son işlemde dahil olmak üzere harcanan kümülatif gas değerine eşit olmalıdır.
- Ödüller dağıtılır. Bloğu üreten hesabın ve ommer hesaplarının bakiyelerinin artırılmasıdır. Bloğu üreten hesaba verilen ödüle ek olarak blok numarasına bakılarak bu bloktan önce ürettiği ommer bloklar için de ekstra aldığı ödülün 1/32 si eklenir.
- Durum ve blok nonce değeri doğrulanır veya madencilik yapılıyorsa geçerli bir durum ve blok nonce değeri hesaplanır.

3.1.1.6. Ethereum Durum Geçiş Fonksiyonu

Bir Ethereum hesabının state' i σ = state, a = adres olmak üzere aşağıdaki dört alandan oluşmaktadır (Buterin, 2014):

- **Nonce:** Bu hesabın sahip olduğu adresten gönderilen işlem sayısı veya kod ile ilişkili hesaplar için bu hesap tarafından yaratılan sözleşme sayısıdır. Resmi olarak bu ifade ile temsil edilir: σ[a]n. (n = nonce)
- **Balance:** Adresin sahip olduğu Wei miktarıdır. Resmi olarak bu ifade ile temsil edilir: σ[a]b.(b = balance)
- **storageRoot:** Merkle Patricia ağacının kök düğümünün 256-bitlik hashidir. Resmi olarak bu ifade ile temsil edilir: σ[a]s . (s = storage)
- **codeHash:** Bu hesabın EVM kodunun hashine eşittir. Hesaba bir mesaj çağrısı gelmesi durumunda bu kod çalışır. Yaratıldıktan sonra değiştirilemez. Resmi olarak bu ifade ile temsil edilir: σ[a]c. (c = code)

Kodun b ile temsil edildiğini varsayarsak KEC(b) = σ[a]c.

Ethereum durum geçiş fonksiyonu, APPLY(S, TX) -> S' aşağıdaki gibi tanımlanabilir:

- İşlem iyi oluşturulmuş mu(örneğin: doğru sayıda değer var mı), imza geçerli mi ve nonce gönderenin hesabındaki nonce değeri ile eşleşiyor mu diye kontrol eder. Şartlardan biri sağlanmadığı taktirde hata döndürülür.
- İşlem ücreti STARTGAS * GASPRICE formülüyle hesaplanır ve imzaya bakılarak gönderenin adresine karar verilir. Gönderenin hesap bakiyesinden ücret çıkarılır ve gönderenin nonce değeri artırılır. Eğer gönderenin hesabında yeterli bakiye bulunmuyorsa hata döndürülür.
- GAS = STARTGAS ataması yapılır ve işlemdeki her byte için gereken toplam gas miktarı hesaplanır.
- İşlem değeri gönderenin hesabından alıcının hesabına transfer edilir. Eğer alıcı hesabı yoksa yaratılır. Eğer alıcı hesap bir sözleşme hesabı ise sözleşme kodunu tamamlanana kadar veya gas bitene kadar çalıştır.
- Eğer değer transferi gönderen hesabın yeterli parası olmaması yüzünden başarısız olursa veya kodu tamamiyle çalıştırıp bitirmek için gerekli gas miktarı bulunmuyorsa, ücret ödemesi dışında gerçekleşen bütün durum değişikliklerini geriye alır ve tahsil edilen ücretleri madencinin hesabına ekler.
- Aksi durumlarda, kalan gasları göndericiye geri ödenir ve kullanılan gas ücretleri de madenciye gönderilir.

Örneğin sözleşme senaryosunun sözde kodunun aşağıdaki gibi olduğunu varsayılsın:

```
if !self.storage[calldataload(0)]:
        self.storage[calldataload(0)] = calldataload(8)
```

Sözleşmenin storage alanı başlangıçta boş olsun ve 9 Ether, 2000 gas, 0.002 gasprice ve 0-7 byte aralığının 5 sayısını, 7-16 byte aralığının KEREM stringini temsil ettiği 16 byte lık bir veri içeren bir transaction gönderilmiş olsun. Bu durumda durum geçiş fonksiyonu aşağıdaki gibi işler:

- İşlem geçerli mi ve iyi oluşturulmuş mu diye kontrol edilir.
- İşlemi gönderenin bakiyesinde en az 2000 * 0.002 = 4 Ether var mı diye kontrol edilir. Eğer varsa gönderenin hesabından 4 Ether eksiltilir.
- Başlangıçta verilen gas 2000, işlemin toplam boyutu 200 byte olsun. Her byte için 5 Ether ücret ödeneceği için geriye 1000 gas kalır.
- Gönderenin hesabından 9 Ether daha çıkarılır ve sözleşmenin hesabına eklenir.
- Kod çalıştırılır, sözleşmenin index 5 teki storage alanının kullanılıp kullanılmadığı kontrol edilir. Kullanılmıyorsa index 5' e KEREM değeri yazılır. Bu işlemin 150 gas tükettiğini varsayarsak geriye kalan gas miktarı 1000-150 = 850 olur.
- İşlemi gönderenin hesabına 850 * 0.002 =1.7 Ether geri ödenir ve sonuç durum döndürülür.

Eğer işlemin gönderildiği alıcı tarafında biz akıllı sözleşme yoksa toplam işlem ücreti verilen GASPRICE değeri ile işlemin byte cinsinden uzunluğu ile çarpımına eşit olur ve işlem ile gönderilen veri anlamsız olacaktır (Buterin, 2014).

Yapılan işlemleri geri alma konusunda mesajlar işlemlere eşit şekilde çalışır: eğer bir mesajı çalıştırırken gas yetmezse mesajın çalıştırılması ve bunun tetiklediği diğer çalıştırmalar geri alınır fakat bu mesajın uygulanmasından önceki uygulamalar geri alınmaz. Eğer bir A sözleşmesi B sözleşmesini G miktar gas ile çağırırsa A nın çalışması en fazla G miktarda gasın harcanacağını garanti eder. Bu bir sözleşmeden bir başka sözleşmeyi çağırmanın güvenli olduğu anlamına gelmektedir (Buterin, 2014).

Kod içermeyen, Nonce ve balance değerleri 0 olan hesaplar boş hesaplardır.

$$EMPTY(\sigma, a) \equiv \sigma[a]c = KEC(()) \wedge \sigma[a]n = 0 \wedge \sigma[a]b = 0$$

Eğer bir hesabın state'i boşsa veya bulunmuyorsa bu hesap ölü hesaptır:

$$DEAD(\sigma, a) \equiv \sigma[a] = \emptyset \vee EMPTY(\sigma, a)$$

3.1.1.7. Ethereum Blokzinciri ve Madencilik

Ethereum blokzincirinin Bitcoin blokzincirine benzer ve farklı yönleri vardır. Ethereum ve Bitcoinin temel farkı blokzinciri mimarilerinde kaynaklanmaktadır. Bitcoin işlem listesinin bir kopyasını tutarken Ethereum blokları hem işlem listesini hem de en son durum bilgisini tutar. Bunun yanı sıra blok sayısı ve zorluk bilgileri de bloklarda tutulur. Aşağıda Ethereumun basit blok doğrulama algoritmasının adımları verilmiştir (Buterin, 2014):

- Referans edilen bir önceki blok var mı ve varsa geçerli mi diye kontrol et.
- Bloktaki zaman damgası bir önceki bloğun zaman damgasından büyük mü ve gelecekteki 15 dakikadan küçük mü diye kontrol et.
- Blok sayısını, zorluğu, işlemin kökünü, uncle kökünü ve gas limitini kontrol et.
- Bloktaki PoW geçerli mi diye kontrol et.
- S[0] bir önceki bloğun durumu olsun.
- TX bloğun n sayıdaki işlemin olduğu işlem listesi olsun. Her 0 - (n-1) aralığındaki i değeri için S[i+1] = APPLY(S{i], TX[i]) set edilsin. Eğer herhangi bir uygulama hata döndürürse veya blokta tüketilen toplam gas miktarı GASLIMIT değerini aşmışsa hata döndür.
- S_FINAL , S[n] olur ve blok ekleme ödülü madenciye ödenir.
- S_FINAL durumunun Merkle Ağacı kökü blok başlığında verilen final durum kök değerine eşit mi diye kontrol et.

Bu yaklaşım her bloktaki her durumun kaydedilmesi yüzünden verimsiz gibi görünebilir fakat durum bilgisinin ağaçta saklanması sayesinde her bloktan sonra ağacın hepsi değil küçük bir kısmı değişmek zorunda kalacağı için verimlidir. Bu nedenle ardışık iki bloktaki ağaçların büyük bir kısmı birbirine benzeyecektir ve dolayısıyla veri bir kere depolanacak ve pointer kullanılarak refere edilecektir. Merkle ağacının özelleşmiş bir hali olan Patricia ağacı bu durumun üstesinden gelmek için düğümlere sadece değiştirilebilme değil eklenip silinebilme özelliği verilmiş bir ağaçtır. Son durum bilgisi son blokta varolan bir bilgi olduğu için tüm blokzinciri geçmişinin saklanmasına gerek kalmayacaktır. Bunun sayesinde depolama alanlarından tasarruf edilir.

3.1.1.8. Endişelere Çözüm GHOST Protokolü

GHOST (Greedy Heaviest Observed Subtree-Gözlemlenen En Ağır Açgözlü Altağaç) protokolü ilk olarak Aralık 2013'te duyurulmuştur. Bitcoin'de en uzun zincir kuralı varken Ethereum'da ise bunun karşılığı en ağır altağaç kuralıdır (Buterin, 2014). Hızlı onaylama

süreleri olan blokzinciriler genelde güvenliğin azalıyor olmasından muzdarip olurlar çünkü blokların ağda yayılması süre almaktadır. Örneğin bir X madencisi bir blok kazar ve bir Y madencisi de X'in kazdığı blok kendisine ulaşmadan bir başka blok kazarsa bu durumda Y madencisinin kazdığı blok ziyan olmuş olur ve ağın güvenliğine bir katkıda bulunmaz. Buna ek olarak blokzincirinin merkeziyetçilikten uzak olması beklenirken bir merkezileşme durumu da söz konusudur. Örneğin; eğer bir A madenci havuzu 30% hash gücüne ve bir B ise 10% hash gücüne sahipse, yukarıdaki örnekte anlatıldığı üzere A havuzunun zamanın 70% lik kısmında ve B madencisinin de zamanın %90 luk kısmında bayat blok denilen üretilmiş ama kendisinde daha önce üretilen bir bloğun ona ulaşması geciktiği için zincire eklenmeyen için blok olan bloklar üretme riskini taşıyacaktır. Bu nedenle, eğer blok üretme aralığı bayat blok üretme oranının yüksek olması için yeterince kısaysa A havuzu yüksek hash gücü sayesinde büyük ölçüde daha verimli olacaktır. Bu iki neden birleştirildiğinde blok üretme hızı yüksek olan blokzincirlerde ağın hash gücünün çoğunluğunu taşıyan bir madenci havuzunun madencilik işlemlerini yönetmesi ve kontrol altına alması beklenir. GHOST ilk sorunu en uzun zinciri hesaplarken sadece bloğun ata bloklarını değil bayat torunlarını da hangi bloğun geçmişinde daha çok PoW olduğunu bulmak için hesaba katar. Bitcoindeki stale(bayat) bloğun Ethereumdaki karşılığı uncle bloktur. Ethereum, madencileri bir blok kazdıklarında onun içinde uncleların olduğu bir listeyi de tutması için teşvik eder. Bunun iki sebebi vardır:

- Yukarıda bahsedilen ilk sorun olan merkezileşme sorununu uncle blok üreten madencilere de hash gücü yüksek madenci havuzunun bir üyesi olmadıkları ve üretilen bloklardan ağdaki yayılma süresinden geç haberleri olduğu için ödül vererek azaltmak.
- Ana zincirdeki yani Ethereumdaki karşılığı olan en ağır alt ağaçtaki iş miktarını uncle bloklarda yapılan işlerle artırarak zincirin güvenliğini artırmak.

Ethereum aşağıdaki yedi adımdan oluşan basitleştirilmiş GHOST protokolünü kullanmaktadır:

- Bir blok bir parent ve 0 veya daha fazla uncle blok belirtmelidir.
- Bir B bloğunda bulunan bir uncle aşağıdaki özelliklere sahip olmalıdır:
    - Bu uncle, 2 <= k <= 7 olmak üzere B nin k. atasının bir çocuğu olmalıdır.
    - Bu uncle B'nin atası olamaz.
    - Uncle ın geçerli bir blok başlığı olmalıdır
    - Bir uncle, daha önceki bloklarda bulunan uncle lardan ve aynı blok içerisindeki diğer uncle lardan iki veya daha fazla kere aynı uncle ın eklenip teşvik kazanılmasının önüne geçilmesi için farklı olmalıdır (non-double-inclusion).

- B bloğundaki her U uncle için, B bloğunun madencisi coinbase ine eklenmek üzere ekstra 3.125% ödül alırken, U uncle bloğunun üreticisi ise geri standart coinbase ödülünün 93.75%' ini alır.

GHOST'un en fazla yedi nesil öncesine kadar olan uncle ların dahil edildiği bu sınırlı versiyonu iki neden için kullanılır. İlki, sınırı olmayan yani gidilebildiği kadar öncesine gidilerek bakılacak verilen blok içerisindeki uncle bloklarının hangisinin geçerli olduğunu bulmak hesaplamalarda çok karışıklığa neden olacaktır. İkincisi ise, limitsiz GHOST'un kullanılması teşviği ortadan kaldırdığı için madencinin saldırganların oluşturduğu zincirde değil de ana zincirde madencilik yapması için bir neden bırakmamasıdır (Buterin, 2014).

3.1.1.9. Hesaplama ve Turinge Tam Uyumluluk

Ethereum sanal makinesi (EVM) içinde koşan EVM kodu sonsuz döngüler de dahil olmak üzere her türlü hesaplamayı makul ölçülerde yapabildiği için EVM'ye Turinge tam uyumludur (Turing-Complete) denir. EVM kodu döngülere iki türlü izin verir. İlki olan JUMP komutu programın kod içindeki bir önceki noktaya zıplamasını sağlar ve JUMPI komutu, while a < 13: a = a * 2 gibi koşullar oluştuğu taktirde zıplama yapmak için kullanılır. İkincisi, sözleşmeler diğer sözleşmeleri potansiyel olarak özyineleme yoluyla döngü yapmaya izin verirler. Bu doğal olarak kötü niyetli kullanıcıların zorlamasıyla madencilerin ve full düğümlerin sonsuz döngülere sokularak kapatılmasına neden olabilir. Bu problem bilgisayar bilimlerinde durma problemi olarak bilinen verilen bir programın durup durmayacağı hakkında bir şey söylenememesi yani durma problemi yüzünden ortaya çıkmaktadır (Buterin, 2014).

Durum geçişlerinde anlatıldığı üzere bir işlemin süreceği maksimum adım sayısı belirtilir ve eğer hesaplama daha uzun sürerse hesaplama geri alınır ama ücretler yine de ödenir. Mesajlar da aynı şekilde çalışmaktadır. Bu çözümün arkasındaki mantığı aşağıdaki örnek daha anlaşılır kılmaktadır:

- Bir saldırgan sonsuz bir döngü içeren bir sözleşme koşar ve sonra madenciye bu döngüyü aktifleştirecek bir işlem gönderir. Madenci işlemi görüp sonsuz döngüyü çalıştırır ve gas tükenene kadar bekler. Bu döngü çalışırken gas bitse ve işlem yarıda kalsa bile işlem hala geçerlidir ve madenci hala her bir hesaplama adımı için saldırgandan ücretlerini talep eder.
- Bir saldırgan çok uzun sonsuz bir döngü yaratarak madenciyi bu uzun süre içerisinde hesaplama yapmaya zorlar. Çok uzun süren bu hesaplama bittiğinde ağda başka düğümler tarafından kazılmış başka bloklar ortaya çıkacaktır ve bu durum yüzünden

bu uzun sonsuz döngüyü koşan madenci yaptığı hesaplamalar için ücret talep edemeyecektir. Ama saldırganın madenciden yapmasını istediği hesaplamaların kaç adımdan oluştuğunu belirten bir STARTGAS değeri belirtmesi gerekecektir. Böylece madenci yapacağı hesaplamanın çok uzun süreceğini daha önceden bilmiş olacaktır.

- Bir saldırgan send(A, contract.storage[A]); contract.storage[A] = 0 şeklinde bir kod görür ve sadece ilk adımı gerçekleştirecek kadar gas içeren bir işlem gönderdiğini varsayalım. Bu durumda sözleşmenin sahibinin bu tür ataklara karşı endişelenmesine gerek yoktur çünkü işlem yarıda kesilirse bütün değişiklikler geri alınacaktır.
- Bir finansal sözleşmenin riski azaltmak için dokuz farklı veri kaynağının medyanını alarak çalıştığını varsayalım. Bir saldırgan, dokuz farklı veri kaynağından birini variable-address-call (değişken adres çağrısı) ile düzenleyip sonsuz bir döngü içeren bir hale getirebilir. Bu durumda yine gas yetersizliği sorunu ortaya çıkacaktır. Fakat, finansal sözleşmeye gelecek bir mesaja gas limiti koyularak bu sorun engellenebilir.

3.1.1.10. Ethereum Para Birimi

Ethereum ağı kendine has Ether isimli para birimine sahiptir. Bu para hem farklı tiplerdeki dijital varlıklarla takas için bir likidite katmanı hem de işlem ücretlerini ödemek için bir mekanizma sağlar. Aşağıdaki liste Ether para biriminin alt bölümlerini göstermektedir (Buterin, 2014).

Çizelge 3.3. Ethereum Para Birimleri

Birim	**Wei Değeri**	**Wei**
Wei	1 wei	1
Kwei(babbage)	10^3 wei	1.000
Mwei(lovelace)	10^6 wei	1.000.000
Gwei(shannon)	10^9 wei	1.000.000.000
Microether(szabo)	10^12 wei	1.000.000.000.000
Milliether(finney)	10^15 wei	1.000.000.000.000.000
Ether	10^18 wei	1.000.000.000.000.000.000

3.1.1.11. Ethereum Token Standartları

Ethereum tokenleri akıllı sözleşmelerle temsil edilen dijital varlıklardır. Ethereum platformu üzerinde geliştirilen her projenin kendine ait, belirlenmiş standartlara uygun bir tokeni

olabilir. Tokenler çeşitli amaçlar için kullanılabilirler ve genelde ICO yapılırken toplu satışa sunulur. Alınabilir, satılabilir veya ticareti yapılabilir.

ERC-20 Standardı (Ethereum Request for Comment)

Ethereum blokzincirinde geliştirilen projelerin kendi tokenlerini implemente edebilmesi için geliştirilmiş en yaygın kullanılan standarttır. Bu standartta Ethereum ağındaki tüm tokenlerin takip edeceği kurallar listelenmiştir (Fröwis, Fuchs & Böhme, 2018). Bu kurallar Ethereum token sözleşmesinin uygulamak zorunda olduğu fonksiyonları ve eventleri tanımlar. Bu standart sayesinde birbirinden farklı coinlerin birbirleriyle uyumluluğu sağlanır ve ağın işlevselliği artırılır. Ethereum blokzincirindeki standart token aşağıdaki özelliklere sahiptir:

- Token standardına uyumlu üretilen varlıklar takas edilebilirdir.
- Uyumlu platformlarda, projelerde ve takaslarda kullanılır.
- Uyumlu Dapp larla kullanımı garanti eder
- Diğer para birimleriyle ve akıllı sözleşmelerle etkileşim içindedir.

Protokol Fonksiyonları

Bir token yaratılırken ve akıllı sözleşme oluşturulurken yapılması gereken 6 farklı zorunlu kontrol ve işlem vardır:

- Başlangıçta kaç coin olduğu ve kaç coin verileceğinin kontrolü
- Başlangıç miktarının şirketin sahibinin adresine ve ICO'nun sahibine atanması
- Yatırımcılara bakiyeleriyle orantılı varlıklar yollama
- Coinlerin, işlemler, doğrulama ve coin transferlerinin gerçekleştirilmesi için kullanıcılar arasında dağıtılması
- Kalan bakiyenin kontrolü
- Adresteki coinin transfer için uyumlu olup olmadığının kontrolü

Ethereum platformunda kendi projesini yaratan her geliştirici yukarıdaki şartları sağlayan Ether tabanlı tokenler üretebilir.

Token Bakiyesi

Token sözleşmesinin aşağıdaki iki token tutucaya sahip olduğunu varsayalım:

0x1151161181181131114111611114112111111111 bakiyesi 300 birim olsun (A)

0x22222232222222222222232225222224222722 bakiyesi 400 birim olsun (B)

Sözleşmenin balances veri yapısı aşağı aşağıdaki bilgileri döndürecektir:

balances[A] = 300

balances[B] = 400

balanceOf(. . .) fonksiyonu yine aynı değerleri aşağıdaki şekilde kullanılarak döndürecektir:

tokenContract.balanceOf(A) 300,

tokenContract.balanceOf(B) 400 döndürecektir.

Balances ve balanceOf kavramları web3.js kütüphanesinde bulunan bakiye öğrenme metotlarıdır. Bunların kullanımları versiyonlara göre değişiklik gösterebilirken hızlıca gelişmekte ve değişmekte olan bu teknolojilere ait kavramların kullanımdan kalkmış olmaları da mümkün olabilmektedir.

Token Transferi

A hesabı B hesabına 100 token transfer etmek isterse aşağıdaki fonksiyonu çalıştırması gerekecektir:

tokenContract.transfer(B, 100)

Transfer fonksiyonu balances veri yapısının tuttuğu bakiye değerlerini değiştirir. Yani y eni durumda:

Balances[A] = 200

Balances[B] = 500 olur.

balancesOf fonksiyonu da aynı değerleri döndürecektir.

Approve ve TransferFrom ile Token Bakiyelerinin Değiştirilmesi

Approve bir veri yapısı ve TransferFrom bir fonksiyondur. Eğer A hesabı B hesabını B hesabının kendisine A dan token transferi yapmak isterse, A hesabı aşağıdaki fonksiyonu çalıştırmalıdır. İzin verilecek token miktarının 54 token olduğunu varsayarsak:

tokenContract.approve(B, 54)

Artık approve veri yapısı aşağıdaki bilgiyi tutacaktır:

tokenContract.allowed[A][B] = 54

Eğer ileride B hesabı A dan kendinde token transfer etmek isterse B hesabının aşağıdaki transferFrom() fonksiyonunu aşağıdaki şekilde çağırması gerekir:

B' nin A'dan transfer etmek istediği token sayısı 39 olsun.

tokenContract.transferFrom(A, B, 39)

Bu fonksiyonun çalışmasının bitmesinden sonra balances veri yapısı artık aşağıdaki değerleri göstecektir.

tokenContract.balances[A] = 161

tokenContract.balances[B] = 539

Yukarıdaki izin verilen token miktarının bir kısmının transferi gerçekleştiği için approve veri yapısının göstereceği değer değişecektir

tokenContract.allowed[A][B] = 15

B hesabı hala A hesabından 15 token harcayabilir.

tokenContract.balanceOf(A) = 161

tokenContract.balanceOf(B) = 539 döndürecektir

(https://theethereum.wiki/w/index.php/ERC20_Token_Standard).

ERC-721 Standardı

Ethereum blokzincirinde eşsiz tokenler yaratmak için kullanılan standarttır. Bu standartta arayüz minimize edilerek tanımlanmıştır. ERC-165 standardı ile birlikte çalışmaktadır. Bu standardı kullanan tokenler tek birimlerdir. Küçük birimlere ayrılamazlar (https://101blockchains.com/erc-standards/).

ERC-223 Standardı

ERC-20 standardına göre bir adresten tokenlerin geri çekilmesine izin vermeyen bir sözleşmeye token transferi yapıldığında tokenler yanar ve geriye gönderenin hesabına yüklenemezler. ERC-223 böyle bir sözleşmeye transfer yapılmasını engeller (https://101blockchains.com/erc-standards/).

ERC-621 Standardı

ERC-20 standardının, toplam token arzını artıran veya azaltan iki farklı fonksiyon eklenerek genişletilmiş halidir. ERC-20 standardı kullanılarak oluşturulan bir token sözleşmesine göre bu sözleşme çağırıldığında belirtilen toplam arz miktarı kadar token üretilir ve bir daha token üretilemez ve mevcut tokenden daha az bir tokene düşülemez. Bu toplam arzı değiştirilemez bir değer yapmaktadır. Bu standartla oluşturulan toplam arz miktarı kadar olan token sayısı arzı artıracak ve azaltacak fonksiyonların token sözleşmesine eklenmesiyle değiştirilebilir olur (https://101blockchains.com/erc-standards/).

ERC-777 Standardı

ERC-20 standardında bir transactiondan sonra bir akıllı sözleşme çağırılırken kriterlerin sağlanıp sağlanmadığının kontrolü için bir transaction daha gerekmektedir. Şartlar sağlandığı taktirde akıllı sözleşme çağırılır. Bu durumda işlem sayısı gereksiz artmaktadır. ERC-777

standardı iki transactionu tek seferde gerçekleştiren bir fonksiyon içermektedir (https://101blockchains.com/erc-standards/).

ERC-884 Standardı

Yatırımcılar için whitelist sloganına sahip bu standarda göre kurumlar hissedarlarını blokzinciri kullanarak kaydeder ve kurumlar sadece buraya eklenen yatırımcılardan coin kabul etmektedir (https://101blockchains.com/erc-standards/).

3.1.1.12. Ethereum İstemci Türleri

Komut Satırı Arayüzü (Command Line Interface- CLI) Olan İstemciler

- **Geth:** Ethereum protokolünün Go dili ile yazılmış açık kaynaklı bir versiyonu olan Go Ethereum için olan CLI tabanlı client uygulamasıdır.
- **Cpp-ethereum – eth:** C++ dili ile yazılmış, eski versiyonunun adı webthree-umbrella olan CLI tabanlı bir Ethereum client'ıdır.
- **Pyethapp:** Pyethereum projesi için geliştirilmiş Python client'ıdır (http://ethdocs.org/en/latest/ethereum-clients/).

Grafik Kullanıcı Arayüzü (GUI) Olan İstemciler

- **Parity Ethereum**

  Maksimum yükleme hızı ve en hızlı senkronizasyonu güvenli bir şekilde sağlamak için Rust programlama dili ile geliştirilmiş CLI (command-line interface) tabanlı bir clienttır. Bu clientın 8545 portunda varsayılan olarak JSON-RPC HTTP Sunucusu çalışmaktadır.

  JSON-RPC HTTP Sunucusu

  JSON yükte hafif olan bir veri değişim formatıdır. Ağaç yapısına sahiptir. Number, string, isim ve değer ikililerinden oluşan bir koleksiyonu, objeleri, boolean ve array veri tiplerini destekler. RPC (Remote Procedure Call) dağıtık, merkezi olmayan ağlarda kullanılan bir protokoldür. Bu protokol sayesinde bir bilgisayardaki program ağdaki bir başka bilgisayardaki bir fonksiyonu, metodu ve prosedürü çağırabilir. JSON-RPC ise ağdaki diğer bilgisayarlara yapılan çağrının ve bu çağrıya verilen cevabın yükte hafif olan JSON formatında iletildiği RPC protokolüdür. JSON-RPC HTTP Sunucusu ise JSON formatındaki RPC istek veya cevaplarını ağdan alan veya ağa ileten HTTP sunucusudur (https://wiki.parity.io/Parity-Ethereum).

- **Mist:** Mist tarayıcısında çalışan bir Ethereum cüzdan client uygulamasıdır.
- **Alethzero:** 2016 yılında gelitirilmesi bırakılan bir Ethereum client türüdür.
- **Nethermind:** Nethermind, Ethereum ana ağına(mainnet)ve testnet ağlarına bağlanan ve platformlar arası .NET Core teknolojisini kullanarak özel blokzincirleri ayarlamaya olanak tanıyan Ethereum client türüdür.
- **Pantheon:** Pantheon, Apache 2.0 lisansı altında geliştirilen ve Java ile yazılmış açık kaynaklı bir Ethereum clientıdır. Ethereum açık (public) ağında, özel (private) ağlarda ve Rinkeby, Ropsten ve Görli gibi test ağlarında (testnet) çalışır.
- **Trinitiy:** Python tabanlı Ethereum clientıdır.

3.1.1.13. Ethereum Ağları

Mainnet

Dappların geliştirilip ücretsiz olarak test edilebildiği test networklerinin aksine gerçek işlemlerin tutulduğu ana ağdır ve kullanıcıların gerçek işlemlerini gerçekleştirmeleri için geliştirilmiş bir son üründür. Bu ağda yapılan her işlem ücrete tabidir. Ethereum blokzincirini kullanan bazı projeler ethereum üzerinde kendi mainnetlerini yani kendi blockzincirlerini oluşturabilmektedir. Bütün mainnetlerin bütün blokzincirlerde olduğu gibi kendilerine ait genesis blokları olmak zorundadır. Genelde daha iyi bir performans elde edebilmek için böyle ayrılmalar gözükür. Örneğin blokzinciri tabanlı global bir işletim sistemi olarak çalışan ve önceden Ethereum mainneti üzerinde işlem gerçekleştiren TRON projesi Ethereumun saniyede 25 işlem çalıştırma kapasitesini 2000 işleme çıkardığı kendi mainnetini 2018 yılı içerisinde devreye sokmuştur (https://tron.network/static/doc/white_paper_v_2_0.pdf). Bitcoin Lightining Network de Bitcoin mainneti üzerinde oluşturulmuş off-chain tabanlı, sadece işlemlerin başlangıç ve bitiş bakiyeleriyle ilgilenip aradaki diğer işlemleri bloklara yazmayarak Bitcoin blokzincirinin ölçeklenebilirlik sorununa bir çözüm olarak üretilmiş bir mainnettir (http://lightning.network/docs/).

Testnetler

Testnetler geliştirilmiş akıllı sözleşmelerin veya DAPP'ların amacına uygun, hatasız ve yeterince akıllı yazılıp yazılmadığını test etmek için geliştirilmiş ağlardır. Bu ağlarda yapılan testlerden başarıyla geçen bir Dapp veya akıllı sözleşme artık mainnette gönül rahatlığıyla çalıştırılabilirdir. Eğer testnetlerin herhangi birinde test edilmeden mainnette koşturulan bir Dapp'ta bir hata, gas yetersizliği vb. sorunlar oluşursa bu kullanıcının zararına bir durum olur. Bu yüzden önce testnetler üzerinde tanımlı mali bir karşılığı olmayan kriptoparalarla bu

denemeler yapılır. Bu ağlardaki herhangi bir hesaba mainnet hesaplarından transfer yapıldığı zaman bunun geri dönüşü olamaz, transfer edilen kriptopara yanmış olur. Maddi bir karşılığı olmayan etherler karşılığında işlemleri doğrulayan ve bloklara yazan madencileri mainnetteki gibi teşvik edici bir mekanizma olması gerekmektedir. Aşağıda Ethereum platformu için geliştirilmiş Ethereum testnetleri ve bu ağlardaki düğümlerin yaptıkları test işlemleri karşılığında nasıl ödüllendirildiği listelenerek anlatılmıştır:

- **Morden:** 2015-2016 yılları arasında aktif olan Ethereumun ilk testnetidir. Artık kullanılmamaktadır.
- **Ropsten:** Kullandığı PoW uzlaşı algoritması yüzünden Mainnete en yakın testnet çeşididir. Parity ve geth clientlarının ikisiyle de çalışabilir. Network idsi 3 tür. Ortalama blok süresi 30 saniyedir. PoW algoritmasından dolayı bir madencilik söz konusudur ve teşvik edici Etherler faucet denilen web sayfalarından talep edilir. https://ropsten.etherscan.io/ adrresi üzerinden bu ağdaki işlemler takip edilebilir. Faucet, kullanıcıların hesaplarına madencilik, captcha gibi çeşitli görevleri tamamlaması karşılığında ether veya diğer para birimlerinden belirli miktarlarda gönderen bir ödüllendirme sistemidir. Faucetcrypto, Greenfaucet, Queenfaucet en yüksek ödemeleri yapan Ethereum faucetlerine örnektir.
- **Rinkeby:** Proof of Authority uzlaşı algoritmasını kullanan bir Ethereum testnetidir. Spam saldırılarına karşı korumalıdır ve Ether arzı güvenilir düğümlerce kontrol edilmektedir. Sadece geth clientları için kullanılabilir bir test ağıdır. Network idsi 4 tür. Ortalama blok süresi 15 saniyedir. Ana Ethereum ağından farklı bir uzlaşı algoritması kullandığı için asıl ortamın aynısını örnekleyemez. Teşvik edici ödüller yine faucetlerden talep edilebilir (Iyer & Dannen, 2018). https://rinkeby.etherscan.io/ adresi üzerinden bu testnetin işlemleri takip edilebilir.
- **Kovan:** Proof of Authority uzlaşı algoritmasını kullanan bir başka Ethereum testnetidir. Spam saldırılarına karşı güvenlidir. Ether arzı güvenilir düğümlerce kontrol edilmektedir. Sadece parity clientları ile kullanılabilir bir testnettir. Kullandığı uzlaşı algoritması yüzünden ethereum mainnetini örnekleyecek bir test ağı değildir (Hu, Lee, Chatzopoulos & Hui, 2018). Network idsi 42 dir. Ortalama blok süresi 4 saniyedir. Teşvik edici ödüller faucetlerden talep edilebilir. https://kovan.etherscan.io/ adresinden bu testnetteki işlemler takip edilebilir.
- **Sokol:** Parity istemcileri (client) ile çalışabilen PoA uzlaşı algoritmasını kullanan bir diğer testnettir. Kullandığı PoA algoritması yüzünden mainneti birebir örnekleyecek

bir testnet değildir. Network idsi 77 dir. Ortalama blok süresi 5 saniyedir. https://sokol-explorer.poa.network/ adresinden bu testnet üzerinde yapılan işlemler takip edilebilir. Teşvik edici ödüller faucetlerden talep edilebilir.

- **Görli:** PoA uzlaşı algoritmasını kullanan bir diğer testnettir. Henüz kullanım aşamasında değildir. Network idsi 6284 tür. Ortalama blok süresi 15 saniyedir. Farklı tür clientlar tarafından desteklenmektedir. https://blockscout.com/eth/goerli/ adresinden bu ağ ile ilgili işlemler takip edilebilir.
- **Tobalaba:** Energy Web blokzinciri, enerji sektörünün düzenleyici, operasyonel ve piyasa ihtiyaçları için tasarlanmış açık kaynaklı, ölçeklenebilir bir blokzinciri platformudur. Bu platform için geliştirilmiş testnetin adı Tobalabadır. PoA uzlaşı algoritmasını kullanmaktadır. Ortalama blok süresi 3 saniyedir. 2019 un ikinci çeyreğinde bu projenin mainnetine ait genesis bloğunun oluşturulması planlanmaktadır. Tobalaba testneti üzerinde gerçekleştirilen işlemler https://tobalaba.etherscan.com/ adresinden takip edilebilir.

Farklı işletim sistemlerinde çalışan farklı dillerde yazılmış farklı Ethereum istemci versiyonları mevcuttur. İstemci çeşitliliğinin bir sorun yaratmıyor olması Ethereum Yellow Paper' da belirlenen matematiksel arka planın sağlam, genel geçer bir mantık olduğunu göstermektedir. En çok kullanılan Ethereum clientları Go diliyle yazılmış olan ve Geth olarak bilinen go-ethereum ve Rust diliyle yazılmış olan Parity clientlarıdır. Bunlara ek olarak C++ diliyle yazılmış cpp-ethereum, Python diliyle yazılmış pyethapp, JavaScript ile yazılmış ethereumjs-lib, Java ile yazılmış Ethereum(J), Ruby diliyle yazılmış ruby-ethereum, Haskell diliyle yazılmış ethereumH, Java tabanlı Pantheon ve bunlara ek olarak Nethermind ve Trinity client uygulamaları mevcuttur. Mainette işlem yapmak isteyen kullanıcıların çoğunluğu Dappların keşfedilip kullanılması için bir araç olan Mist Tarayıcısı/ Ethereum Cüzdanını yükler. Mist, Chrome veya Firefox gibi, kullanıcılarına web sayfalarına erişim izni tanıyan bir web tarayıcısıdır. Buna ek olarak kullanıcıların Ethereum ağındaki Dapplara ulaşmasını sağlar. Eğer Mist çalıştırılmadan önce hali hazırda çalışan bir command-line clientı yoksa Mist ile birlikte paketlenmiş olarak gelen go-ethereum ve cpp-ethereum clientlarından biri- varsayılan olarak geth- blokzincirle senkronize edilmek üzere çalıştırılır. Mist ile özel ağ işlemleri yapmak istenildiğinde Mist başlatılmadan önce Parity vb. gibi özel ağ için kullanılabilen clientlarının çalıştırılması gerekmektedir (http://ethdocs.org/en/latest/ethereum-clients/).

3.1.2. Ethereum Dışındaki Diğer Platformlar

- **EOS:** Merkezi olmayan uygulamalar geliştirmek için yaratılmış ölçeklenebilir ve esnek bir altyapıdır (https://eos.io/).
- **Exonum:** Güvenli, izin gerektiren, özel türde blokzinciri uygulamaları geliştirmeye izin veren açık kaynaklı bir frameworktür (https://exonum.com/).
- **Lisk:** Javascript dilinde Dapp geliştirmek için geliştirilmiş bir blokzinciri platformudur (https://lisk.io/).
- **NEO:** Ölçeklenebilir Dapp networkü oluşturmak üzere tasarlanmış bir blokzinciri platformudur (https://neo.org/).
- **Neblio:** İşletmelere uygulamaları ve hizmetleri için geliştirilmiş blokzinciri tabanlı güvenli ve dağıtık bir platformdur (https://nebl.io/).
- **Qtum:** Qtum, açık kaynaklı, halka açık bir blokzinciri platformudur. Qtum, PoS tabanlıdır ve akıllı sözleşmelerden yararlanılarak belirli blok zinciri ayarlarının değiştirilmesine olanak tanıyan bir Merkezi Olmayan Yönetişim Protokolü'ne (DGP) sahiptir. Örneğin, Qtum'un blok boyutları hardfork gerekmeden arttırılabilir (https://qtum.org/en).
- **Rootstock:** Bitcoin Networkü tarafından güvenceye alınmış bir akıllı sözleşme platformudur (https://www.rsk.co/).
- **Snax:** Popüler sosyal medya platformlarında içerik üretenleri ödüllendiren bir blokzinciri çözümüdür (https://snax.one/whitepaper.pdf).
- **Tezos:** Paydaşların protokolü yönettiği, akıllı sözleşme ve merkezi olmayan uygulamalar için geliştirilmiş bir blokzinciri platformudur (https://tezos.com/).
- **Ubiq:** Stabiliteyi merkezinde bulunduran Ethereum Codebase üzerine inşa edilmiş bir platformdur (https://ubiqsmart.com/).
- **Waves:** Üstün nitelikteki Dapp projeleri için geliştirilmiş açık kaynaklı bir blokzinciri platformudur (https://wavesplatform.com/).

3.2. Ethereum Platformunda Koşan Akıllı Sözleşmelerde Kullanılabilen Programlama Dilleri

3.2.1. Solidity

Solidity akıllı sözleşmeleri uygulamak için kullanılan sözleşmeye-yönelik(contract-oriented) yüksek seviyeli bir programlama dilidir. C++, Javascript ve Python dillerinden esinlenerek

oluşturulmuştur ve Ethereum Sanal Makinesinde koşar. Kalıtımı, kütüphaneleri ve kullanıcı tanımlı karışık tipleri destekler (https://solidity.readthedocs.io/en/v0.5.3/).

Tarayıcı Tabanlı Solidity Geliştirme Ortamları

Remix

Remix kullanıcılara Solidity programlama diliyle ethereum akıllı sözleşmeleri oluşturma imkânı sunan tarayıcı tabanlı bir derleyici ve entegre geliştirme ortamıdır (IDE). Remix.ethereum.org adresinden bu ortama erişilebilir (https://github.com/ethereum/remix-ide).

EthFiddle

Ethereumun ölçeklenebilir olmasına yardım amaçlı kurulan temel altyapı platformu olan Loom Network'ün geliştirdiği EthFiddle da aynı Remix gibi tarayıcı tabanlı bir derleyici ve entegre geliştirme ortamıdır (https://ethfiddle.com/).

SuperblocksLab

Akılı sözleşme ve Dapp uygulamaları geliştirmek için geliştirilmiş tarayıcı tabanlı bir derleyici ve entegre geliştirme ortamıdır. Tarayıcı içi blokzinciri, otomatik kod tamamlama, dahili cüzdan, sorunsuz Mainnet yüklemesi ve Metamask entegrasyonu gibi hizmetler sağlamaktadır (https://superblocks.com/lab/).

3.2.2. Diğerler Programlama Dilleri

Vyper

Vyper güvenlik, programlama dili ve derleyici sadeliği ile okunabilirliğin artırılması amacıyla oluşturulmuş beta sürümünde Pythona benzer bir programlama dilidir (https://github.com/ethereum/vyper).

Bamboo

Bamboo durum değişikliğini açık yapan ve varsayılan olarak yeniden giriş sorunlarından kaçınan Ethereum platformunda koşacak akıllı sözleşmeleri geliştirmek için oluşturulmuş bir programlama dilidir (https://github.com/pirapira/bamboo).

eWASM

WebAssembly nin sınırlandırılmış bir altkümesi olan eWASM, Ethereum akıllı sözleşmeleri için geliştirilmiş assembly dilidir.

Serpent

Serpent çeşitli yüksek seviyeli özelliklerle genişletilmiş Ethereum sanal makinesi kodlarını derlemek için oluşturulmuş bir assembly dilidir. Düşük seviyeli opcode yönetimi için kullanılabilir. Güvenlik nedeniyle kullanımı tavsiye edilmemektedir (https://github.com/ethereum/serpent).

Mutan

Mutan Ethereum projeleri için geliştirilmiş C benzeri bir programlama dilidir. 2015 yılının mart ayından itibaren kullanımdan kalkmıştır.

Idris

Chalmers Teknoloji Üniversitesi Bilgisayar Bilimleri Bölümü'nde yapılan bir yüksek lisans tezinde Serpent aracılığıyla EVM'e derlenen projeler yazmak için kullanılan Haskell benzeri fonksiyonel bir programlama dilidir **(**Pettersson & Edström, 2016).

Babbage

Ethereumun fonksiyonalitesi görselleştirmek için sözleşme tasarımcıları tarafından kullanılan görsel bir programlama dilidir.
(https://github.com/s-tikhomirov/smart-contract-languages)

Pyramid

Henüz yapım aşamasında olan Pyramid, Scheme programlama dilinin bir lehçesidir ve Ethereum Akıllı Sözleşmeleri geliştirmek için kullanılan bir programlama dilidir **(**https://github.com/MichaelBurge/pyramid-scheme).

SolidityX

Solidity programlama dilinin eksik yönlerine yönelik en son güvenlik çözümleri ile derlenmiş bir programlama dilidir. Solidity ile tam uyumludur. Swarm ve IPFS entegrasyonu, blokzincirleri arası iletişim için uygulama eklendileri mevcuttur **(**https://solidityx.org/).

Flint

Flint, Ethereum akıllı sözleşmeleri yazmak için geliştirilmiş sözleşme yönelimli(contract-oriented) bir programlama dilidir ve hala geliştirilmektedir (https://github.com/flintlang/flint).

Lolisa

Lolisa Solidity için geliştirilmiş resmi olarak ilk onaylanmış semantik ve sözdizimidir. Lolisa geliştirilmiş tip güvenliği için Solidity'den daha güçlü bir yapıyı benimsiyor. Lolisa, Solidity'de var olan mapping, modifier, contract ve adres tiplerine ek olarak çoklu değer döndürme, pointer ve struct gibi genel amaçlı programlama dillerinde bulunan sözdizimi bileşenlerini de içerir (Yang & Lei, 2018).

Formality

Front-end uygulamaları, back-end servisleri ve akıllı sözleşmeler için geliştirilmiş genel amaçlı bir ispat asistanı ve programlama dilidir. Hızlı, güvenli ve basittir (https://github.com/MaiaVictor/Formality).

Logikon

Akıllı sözleşmeler için geliştirilmeye başlanan deneysel bir programlama dilidir. Aktif olarak kullanılmamaktadır.

3.3. Akıllı Sözleşme Koşabilen Ethereum dışındaki diğer Platformlarda Kullanılan Programlama Dilleri

Michelson

Tezos blokzincirinde koşabilecek akıllı sözleşmeler yazmak için geliştirilen stack tabanlı bir programlama dilidir (Parizi & Dehghantanha, 2018). Michelson kullanıcıların sözleşmelerinin özelliklerini kullanarak resmi onaylamayı kolaylaştırmak için tasarlanmıştır (https://www.michelson-lang.com/).

Liquidity

Liquidity akıllı sözleşmeler için geliştirilmiş, Michelson programlama dilinin kısıtlamalarına tam olarak uyan yüksek seviyeli bir programlama dilidir. Michelson programlama dilinin tüm özelliklerine sahiptir (https://github.com/OCamlPro/liquidity/blob/master/docs/liquidity.md).

Plutus

Cardano blokzincirinde koşacak akıllı sözleşmeleri yazmak için kullanılan, Haskell benzeri bir sözdizimine sahip fonksiyonel bir programlama dilidir (https://cardanodocs.com/technical/plutus/introduction/).

Rholang

Rholang dağıtık sistemlerde kullanılmak üzere tasarlanmış "süreç odaklı (process-oriented)" bir programlama dilidir. Bütün hesaplamalar kanallardan geçen mesajlar ile gerçekleştirilir. Rholang ile bir kanaldan mesaj okurken ve bu mesajı işleme sokarken başka bir mesaj gönderilmesine izin vermeyen asenkron bir yapıdadır. Rholang için geliştirilmiş bir IDE yoktur. Rnode veya Rchain topluluğu üyelerinin geliştirdikleri https://rchain.cloud/ web arayüzü veya IntelliJ platformu için geliştirilmiş bir plugin aracılığıyla yazılabilir (https://developer.rchain.coop/tutorial/).

Obsidian

Obsidian akıllı sözleşmeler için geliştirilmiş bir programlama dilidir. Bu amaçla geliştirilmiş mevcut dillere göre iki ana yenilik içermektedir. Birincisi, durum-yönelimli (state-oriented) programlama sayesinde durumlar arasında açıkça geçiş yapılmasına izin vermesidir. İkincisi ise bu dilde kullanılan lineer tiplerin akıllı sözleşmeler tarafından yönetilen önemli kaynakların doğru şekilde yönetilmesini sağlaması ve kazara gerçekleşen para kayıplarını önlemesidir. Obsidian henüz geliştirilme aşamasında olan ve genel kullanım için uygun olmayan bir dildir (https://mcoblenz.github.io/Obsidian/).

DAML

Dijital Varlık Modelleme Dili (Digital Asset Modeling Language) finansal kurumlar için akıllı sözleşmelere alternatif olarak sunulan bir programlama dilidir. Açık bir blokzinciri platformunda değil de özel bir blokzinciri platformunda kullanım için optimize edilmiştir. DAML, Akıllı Sözleşmelerin, finansal hizmetler için uygun olmayan özellikleri olmaksızın, faydalarından birçoğunu elde etmek için tasarlanmıştır (Bernauer, Blummer, Kfir, Litsios & Meier, 2018)

QSCL

Akıllı sözleşme ve değer transfer protokolü olan QTUM platformu için geliştirilmiş açık kaynaklı ve merkezi olmayan bir akıllı sözleşme dilidir (Dai, Mahi, Earls & Norta, 2017).

Simvolio

Apla platformunda koşacak akıllı sözleşmelerin yazıldığı Turing-Complete programlama dilidir. Apla platformunda sözleşmeler düzenlenebilmesi için Molis isimli yazılım istemcisinde bulunan editör kullanılmaktadır (https://blog.apla.io/apla-for-developers-8cfb458e4cc4).

Marlowe

Cardano blokzincirinde finansal akıllı sözleşmeler oluşturabilmek için geliştirilmekte olan alana özgü bir programlama dilidir (DSL-Domain Specific Language) (Seijas & Thompson, 2018).

SCILLA

Akıllı Sözleşme Ara Seviye Dilinin (Smart Contract Intermaediate-Level Language) kısaltmasıdır. Zilliqa projesi için geliştirilmiştir. Scilla, akıllı sözleşmeler üzerinde bilinen bazı güvenlik açıklarını doğrudan dil seviyesinde ortadan kaldırarak uygulamaları saldırılara karşı daha az savunmasız hale getirecek bir yapıya sahiptir (Sergey, Kumar & Hobor, 2018).

IELE

IELE blokzincirde akıllı sözleşmeler koşmak için geliştirilmiş bir sanal makinedir (Kasampalis, Guth, Moore, Serbanuta B., Serbanuta V., Filaretti & Johnson, 2018).

Pact

Pact, özel blokzincirlere bir örnek olan Kadena blokzincirinde çalıştırılmak üzere akıllı sözleşmeler yaratmak üzere geliştirilmiş bir programlama dilidir. Geliştiricilere kritik konulardaki işleri için hızlı, güvenli ve performanslı işlem mantığı uygulama fırsatı sunmaktadır (Popejoy, 2016).

RIDE

Waves platformunda koşacak akıllı sözleşmeler için geliştirilmiş tembel, statik yazılmış ve ifade tabanlı bir programlama dilidir. RIDE ile yazılan bir akıllı sözleşme Maven isimli bir derleyicide derlenip testler yapılabilir (Chepurnoy, Kharin & Meshkov, 2018).

LDL

Public Ledgerlar oluşturmak için geliştirilmiş bir programlama dilidir. LDL, bir modelleme dili oluştururken üst seviye bileşenlerin kullanımının sadeliğe tanınan öncelik yüzünden en az seviyede tutulmasını amaçlar (Kobeissi & Kulatova, 2018).

BÖLÜM 4

Merkezi Olmayan Uygulamalar (DAPP)

Merkezi olmayan uygulamanın (Decentralized Application) kısaltması olan DAPP merkezi bir sunucusu olmayan, veritabanını kimsenin değiştirip silemeyeceği bir uygulama anlamına gelmektedir. DAPP' in belli başlı özellikleri aşağıdaki gibi listelenebilir:

- Kaynak kodu bütün kullanıcılara açıktır.
- Blokzincire uyumludur ve merkezi değildir.
- Uygulama kendine özgü kriptoparaya sahiptir.
- Kullanıcıları için uzlaşı algoritması sağlar.

Ethereum whitepaper'ına göre 3 tür DAPP vardır (Buterin, 2014);

- **Para Merkezli DAPP:** Bu tür merkezi olmayan uygulamalar taraflar arasındaki sözleşmeyi onaylamak için dijital paralara ihtiyaç duyar. Örneğin Ethereum platformunda koşan akıllı sözleşmeler ancak bu sözleşmeleri koşmak için gerekli olan para ve diğer sözleşme şartları oluştuğunda aktif hale gelir.
- **Para ve Çeşitli Bilgiler Gerektiren Uygulamalar:** Bu tür uygulamalar tokenler ve hesaplamalarda kullanılmak üzere ekstra bilgiler gerektirir. Örneğin tedarik zincirindeki bir ürünün üreticiden tüketiciye gelişindeki temel şey ürünün fiyatıyken taraflar arasındaki mesafe de maliyet hesaplaması için dikkate alınması gereken ekstra bir bilgidir.
- **Hükümet ve Oylama Yönetimi Uygulamaları:** Dapp ile birlikte ortaya çıkan DAO kavramı akıllı sözleşmelerin yönetimsel alandaki sorunlara bir çözüm olarak üretilmiştir. Yönetim kuralları akıllı sözleşmelerle dijital zorunluluklar haline gelir ve Ethereum gibi akıllı sözleşme koşabilen platformlarda koşar. Ethereum blokzincirine koyulan sözleşmeleri değiştirmek zordur. Bu kimsenin kuralları değiştiremeyeceğini düşünürsek güvenlik açısından iyidir fakat yürürlükte olan bir DAO'nun sözleşmesinde bir hata farkedilirse kod değiştirilemeyebilir.

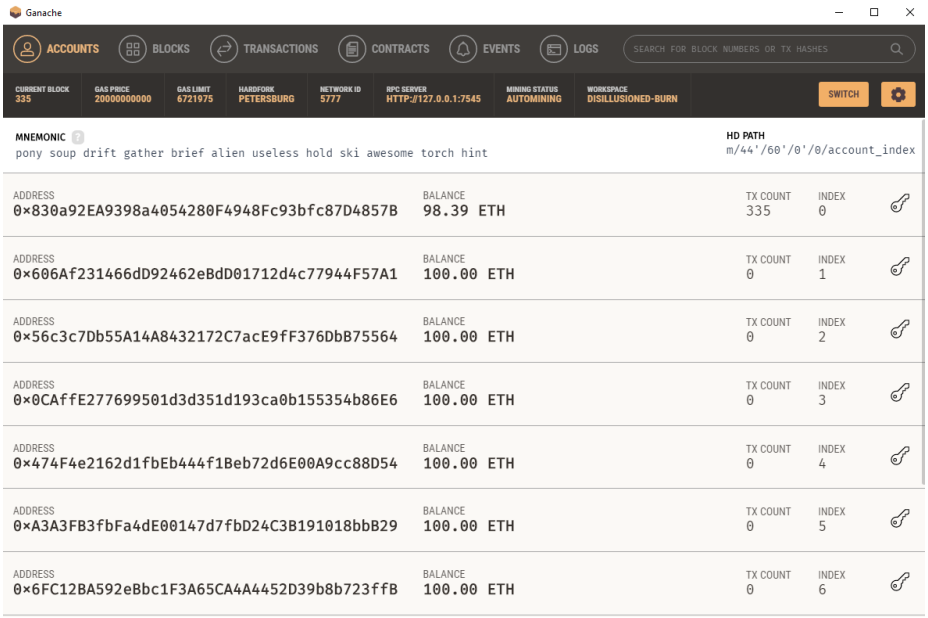
Tez kapsamında geliştirilen Dapp uygulamasının Backend servisi olarak NodeJS, akıllı sözleşme dili olarak Solidity, web uygulaması geliştirilmesi ve tasarımı için de HTML5, Jquery, Javascript ve Bootstrap kullanılmıştır. Ayrıca geliştirilen uygulama için kullanılan diğer teknoloji ve yöntemler aşağıda açıklanmıştır.

4.1. Truffle

Akıllı sözleşme geliştirmek için yaratılmış bir framework'tür. Windows makinelerde Node.js komut satırından **npm install –g truffle** komutuyla indirilebilir. Box denilen hazır akıllı sözleşme uygulaması template'lerine sahiptir. Bunlara https://truffleframework.com/boxes adresinden erişilebilir. Node.js komut satırı arayüzünde **truffle unbox box-ismi** şeklinde proje klasörüne hazır box'lar indirilebilir. Bu uygulamada pet-shop box'ı kullanılmıştır. Truffle ile geliştirilen bu box'lar isteğe göre düzenlendikten sonra truffle develop komutuyla geçilen truffle geliştirme komut satırı ekranında sırasıyla derlenip (compile), deploy (migrate) edilmelidir. Bu işlemler truffle develop ortamına geçilmeden de truffle compile ve truffle migrate şeklinde gerçekleştirilebilir. Derlenen ve deploy edilen bir akıllı sözleşme uygulamasında yapılan değişikliklerin işleme konulması için truffle migrate –reset komutu çalıştırılmaldır. Bu box'ı çalıştırmak için proje klasörü içindeyken npm run dev komutu kullanılmalıdır. Bu komut çalıştırıldıktan sonra NodeJS tabanlı lite-server çalışacak ve varsayılan olarak http://localhost:3000 adresinde uygulama çalışmaya başlayacaktır.

4.2. Ganache

Blockzinciri teknolojisinin temel dayanağı olan antimerkeziyetçiliğin kişisel bilgisayarlarda temsil edilmesi için yerelde özel bir Ethereum blokzinciri oluşturmayı sağlayan bir yazılımdır. Ganache' ın kurulumu ile birlikte oluşacak yerel özel ethereum blokzincirinde varsayılan olarak 10 hesap oluşturulur. Her hesapta, gerçek hayatta karşılığı olmayan, sadece bağlanılacak bir testnette veya özel RPC networkünde kullanılabilecek 100ETH tanımlıdır. Hesaba tanımlı bu paralar, bu hesap tarafından test edilecek veya ağa yükelenecek akıllı sözleşmelerin blokzincirinde koşmasından kaynaklı maliyeti karşılamak için kullanılacaktır. Ganache hem masaüstü uygulaması hem de CLI olarak mevcuttur (https://truffleframework.com/docs). Masaüstü uygulamasının ekran görüntüsü aşağıdaki gibidir:



Şekil 4.1. Ganache Masaüstü Uygulaması ve Tanımlı Test Hesapları

Ganache genelde http://localhost:9545, http://localhost:8545 veya http://localhost:7545 RPC sunucularında ve 5777 id'li ağda çalışır. Bu tezdeki uygulamada Ganache http://localhost:7545 adresinde ve 5777 id'sine sahip ağda çalışmaktadır. Kazılan toplam blok sayısı, gas price, gas limit, ağ idsi, bulunulan RPC sunucu adresi ve her hesaba tanımlı güncel bakiye arayüzde kullanıcıya sunulmaktadır. Ayrıca arayüz, blokları ve blokları oluşturan transactionları ayrı ayrı incelemeye fırsat sunmaktadır. Görüntüde görülebilen, adreslerin en sağında bulunan anahtar simgesine tıklanıldığında bu ağa ait private key bilgisine erişilebilir. Bu private key, bu adresin, bilgisayara kurulan bir full ethereum nodu olmadan çalışması için kullanılan Metamask'ın Chrome eklentisindeki light düğüme ether aktarmak için kullanılır.

4.3. Metamask

Metamask, full düğüm koşamayan yani donanımsal olarak yeterli olmayan bilgisayarlar için gelişitirilmiş, light düğüm koşucu bir tarayıcı eklentisidir. Bu tezde yapılan uygulama için Metamask Chrome eklentisi kullanılmıştır. Metamask'ta oluşturulan veya metamaska import edilen bir düğüm Ethereum mainnetine, Kovan, Rinkeby, Ropsten gibi testnetlerine bağlanabilir (https://metamask.io/).

4.4. Infura

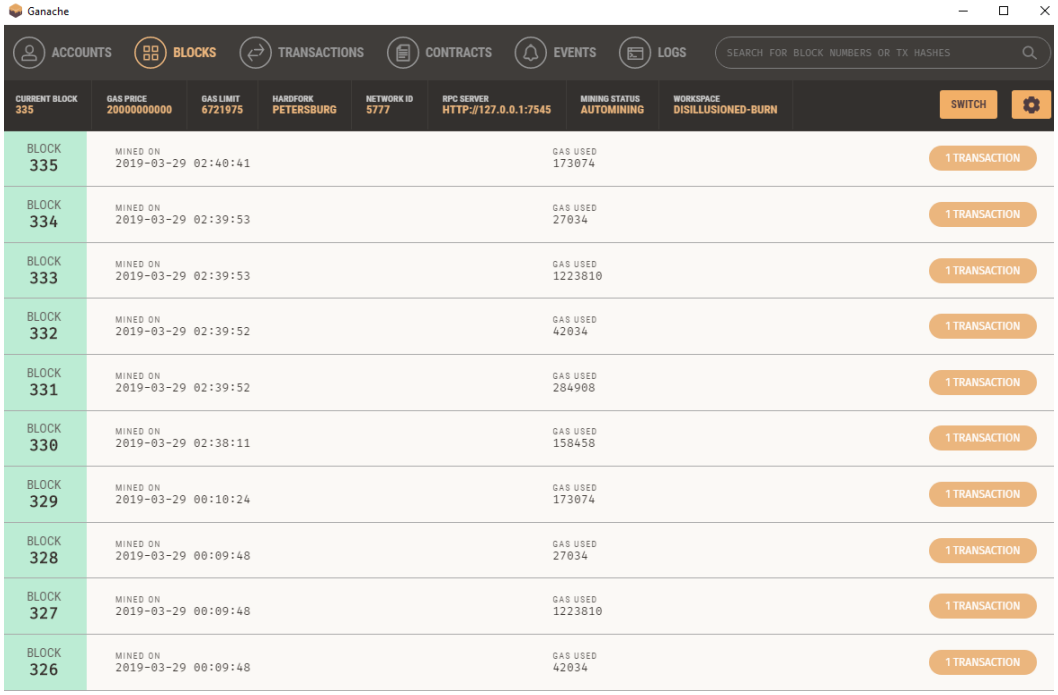
Testnetleri için blokzinciri ağları sağlayan bir altyapıdır. Metamaska tanımlanan private key sayesinde Ganache tarafından sunulan bir test hesabı Infura altyapısında çalışan diğer düğümlerle aynı ağa katılıp işlemlerini yapmaya başlar (https://infura.io/).

BÖLÜM 5

BLOKZİNCİRİ TABANLI DİJİTAL SERTİFİKASYON UYGULAMASININ GELİŞTİRİLMESİ

5.1. Lokal Blokzincirinin Oluşturulması

Ethereum ağında ful bir düğüm koşmak için düğümün bulunacağı bilgisayara yüksek miktarda işlem geçmişi içeren bir düğüm, bir cüzdan indirilmesi gerekmektedir. Bu durum sadece uygulama geliştirmek isteyen kişiler için kullanışsızdır. Buna çözüm olarak sadece kendi düğümü işlem yaptıkça hafızasında bloklar tutacak bir yöntem geliştirilmiştir. Bu uygulamanın içerdiği akıllı sözleşmenin yaratıldığını ve gerekli durumlarda çağırıldığını kolayca gözlemleyebilmek için Ganache teknolojisinin masaüstü uygulaması versiyonu kullanılmıştır. Arayüzünden, oluşturulan blokzincirinin bloklarıyla ilgili bilgiler takibe açıktır.



Şekil 5.1. Ganache Üzerinden Blokzincirinin Görüntülenmesi

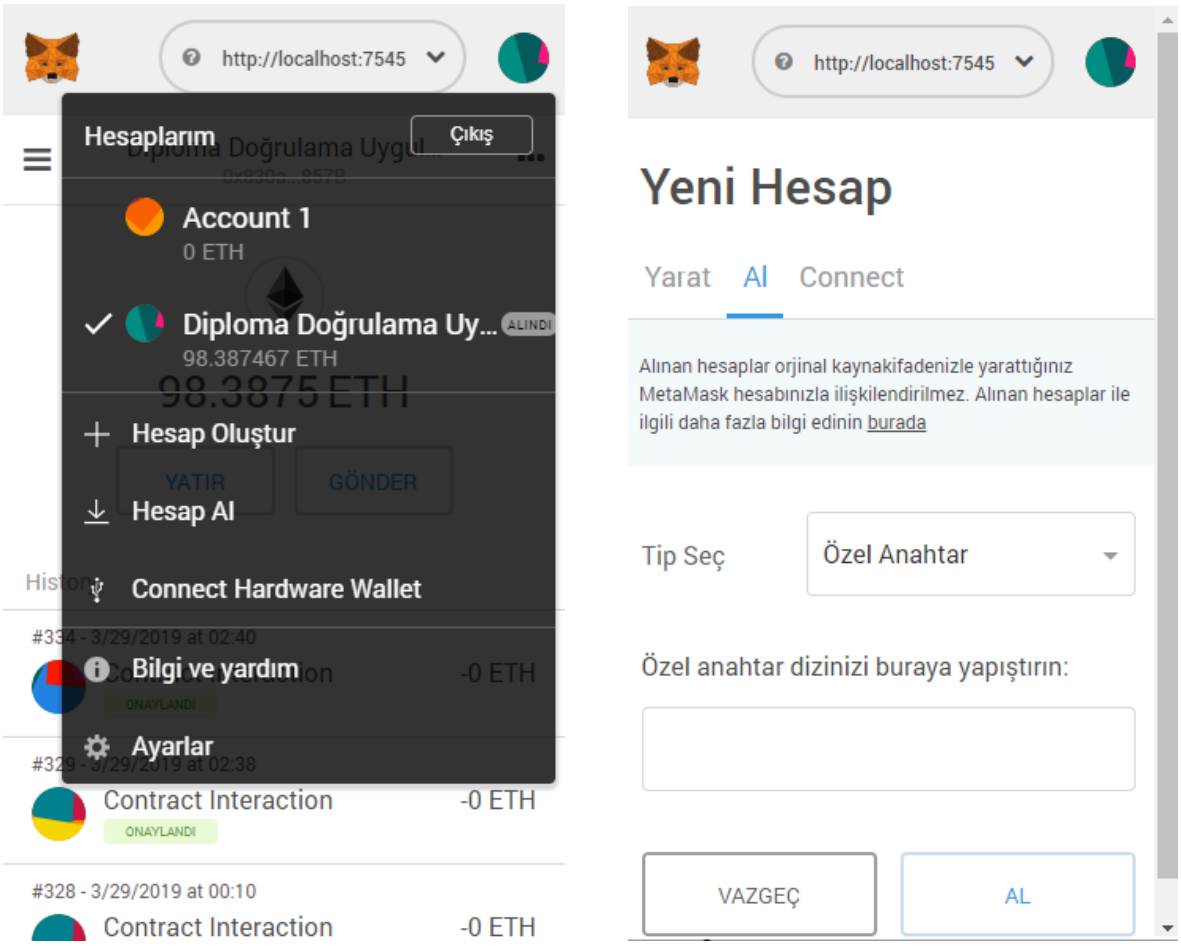
5.2. Test Hesabın Metamask Light Düğümü Olarak Devreye Alınması

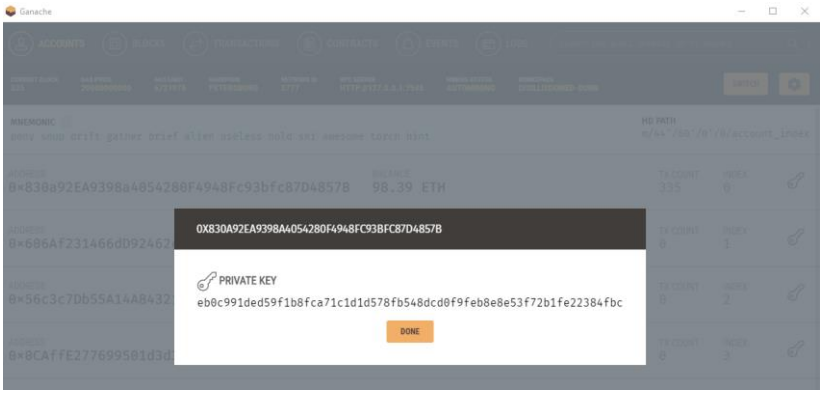
Ganache kurulduktan sonra test kullanımları için kullanıcıya sunulan 100 ether bakiyeli hesaplardan herhangi biri Infura destekli testnetlerde test işlemlerinin gerçekleştirilmesi için

kullanılabilir. Bu hesaplardan birinin Metamask eklentisiyle kullanılması için sırasıyla aşağıdaki adımlar izlenmelidir:

- Metamask eklentisine tıklanmalıdır.
- Açılan pencerede görünen sağ üstteki avatar tıklanmalıdır.
- Açılan pencerede görünen Hesap Al bağlantısına tıklanmalıdır.
- Ganache tarafından verilen 10 hesaptan birinin özel anahtarı (private key) adres bilgisinin en sağında bulunan anahtar simgesine tıklayarak elde edilip ilgili alana eklenmelidir.

Yapılan işlemlerin, lokalde bulunan ve Ganache tarafından sağlanan blokzincirine Metamaskta koşan light düğüm tarafından yazılabilmesi için Özel RPC ağı oluşturulmalı ve bunun adresine Ganache'ın çalıştığı adres verilmelidir. Bu uygulamada blokzinciri http://localhost:7545 adresinde çalışmaktadır.

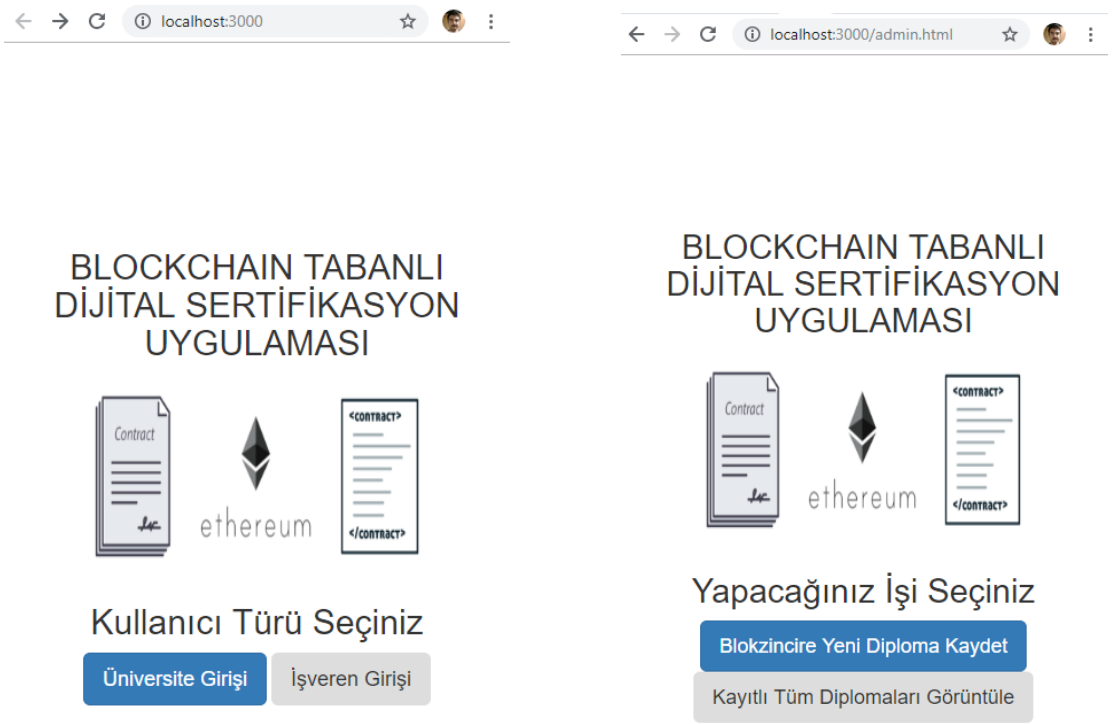




Şekil 5.2. Ganache-Metamask Entegrasyonu

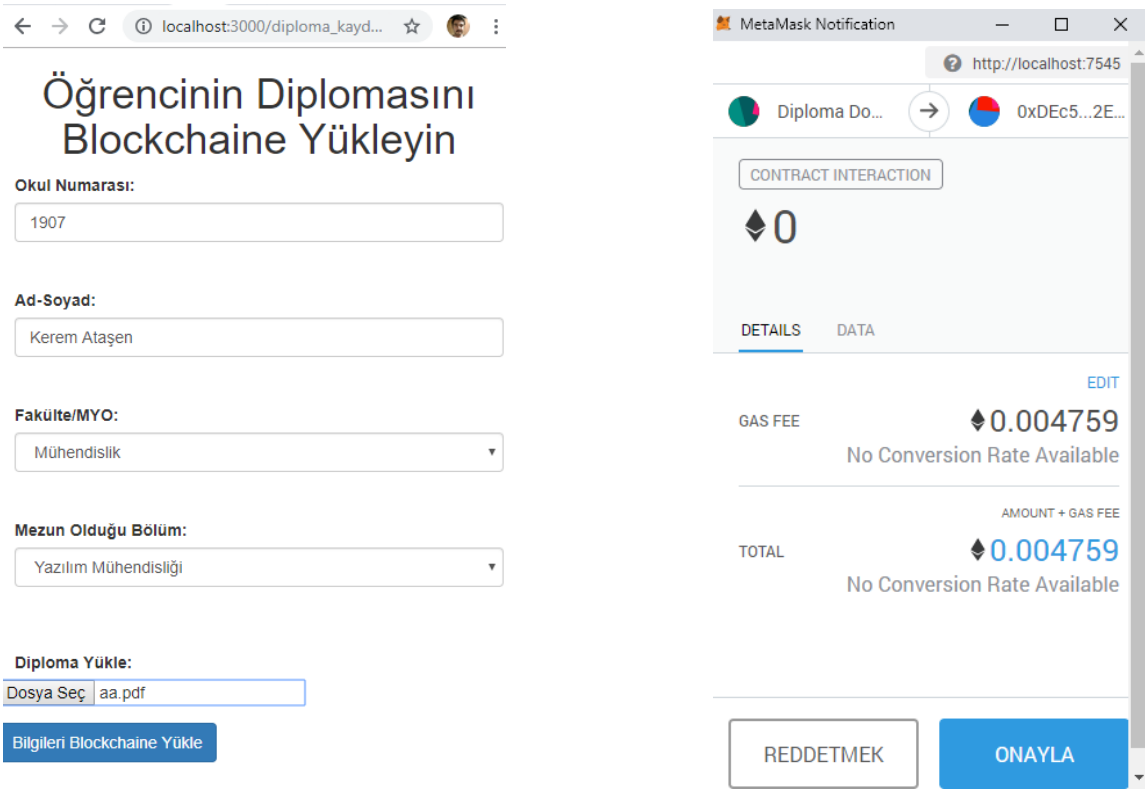
5.3. DAPP Geliştirilmesi

Dapp için truffle boxlarından olan pet-shop boxı tercih edilmiştir. Uygulamanın tutulması istenilen klasör altında **truffle unbox pet-shop** komutuyla indirilir (https://truffleframework.com/docs). İndirilen bu uygulamanın sayfalarında yapılacak uygulamaya göre özelleştirmeler gerçekleştirilir. Bu uygulamada biri sisteme diploma yükleyen üniversite ve diğeri diploma doğrulama gereksinimi duyan işveren için olmak üzere iki farklı kullanıcı girişi sunulmuştur. Üniversite girişinde açılan ekranda yeni diploma kaydı ve kayıtlı olan tüm diplomaları görüntülemek için iki seçenek sunulmuştur.



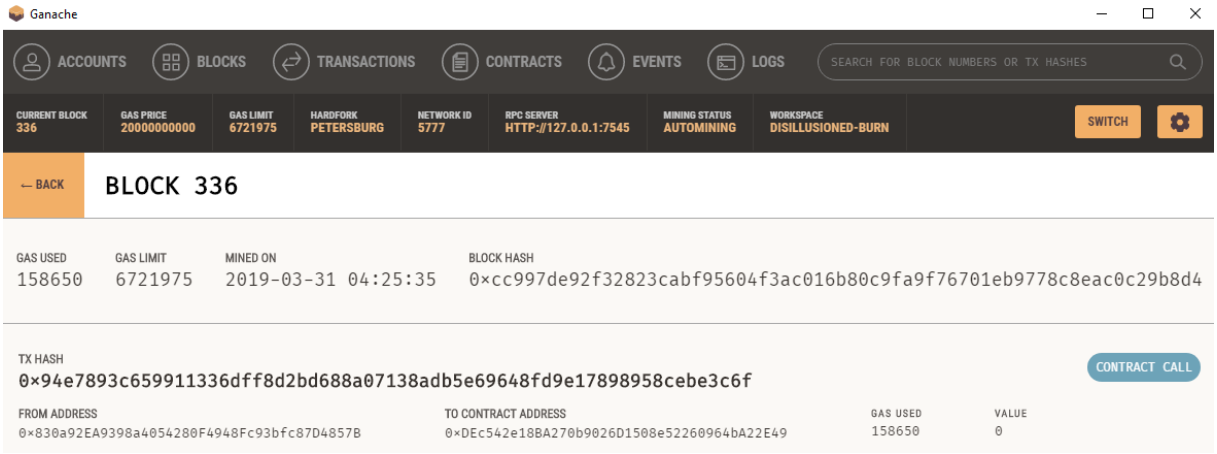
Şekil 5.3. Üniversite Girişi İçin DAPP Arayüzü

Blokzincire yeni diploma kaydet butonuna tıklandıktan sonra diploması kaydedilecek öğrencinin ad-soyad bilgisi, okul numarası, mezun olduğu fakülte ve bölüm isimlerinin girildiği ve öğrencinin diplomasının yüklendiği sayfa açılacaktır. İlgili bilgiler girildikten ve Bilgileri Blockchaine Yükle butonuna tıklandıktan sonra Metamask bu işlemin maliyetini gösteren ve onaylanıp onaylanmadığını soran bir ekran açar.



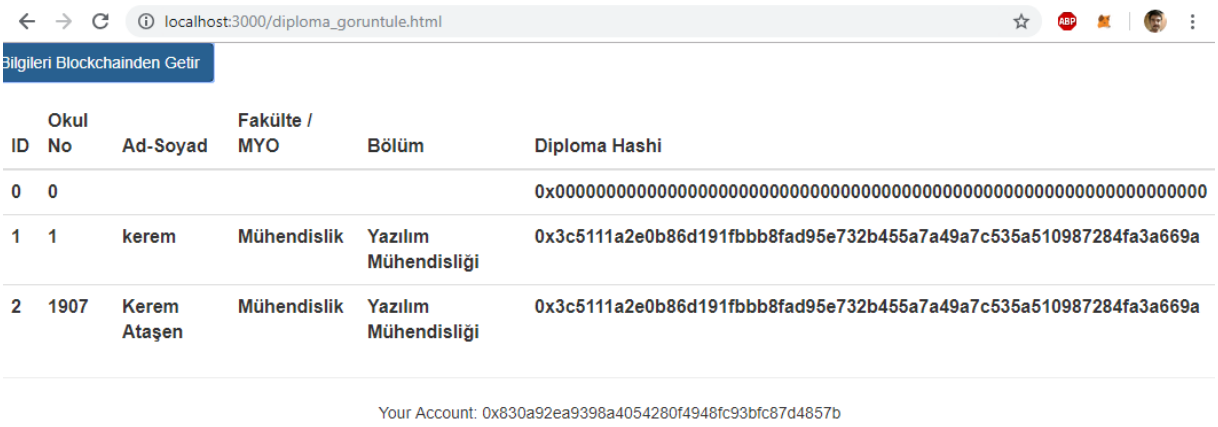
Şekil 5.4. Diploma Yükleme ve İşlem Onaylama Ekranları

Onaylanan işlem Ganache arayüzünden görülebilir.



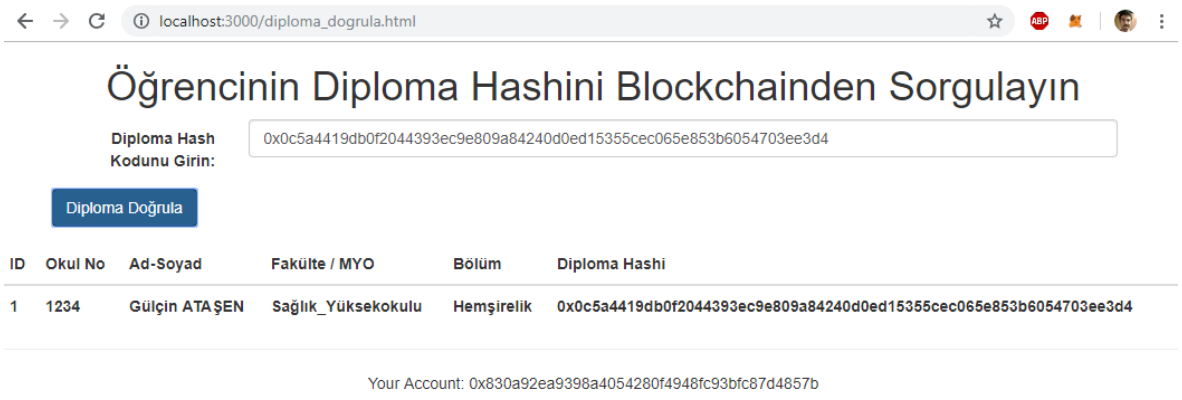
Şekil 5.5. Ganache Üzerinden Seçilen Blok Bilgisi Görüntüleme

Kayıtlı Tüm Diplomaları Görüntüle butonuna tıklandığında öğrencilerin bilgileri ve diplomlarının hash değeri gösterilmektedir.



Şekil 5.6. Diploması Kayıtlı Tüm Öğrencilerin Görüntülendiği Ekran

İşveren girişinde ise sadece iş bavurusunda bulunan öğrencinin işverene sunduğu diploma hash bilgisinin girileceği bir alan bulunmaktadır. Bu alana girilen hash bilgisinden sonra Diploma Doğrula butonuna tıklandığında blokzincirde bu diploma hash bilgisine sahip bir öğrenci varsa bu öğrencinin bilgileri listelenir. Eğer eşleşme bulunamazsa böyle bir diploma hash değerine sahip bir öğrenci olmadığını belirten bir uyarı verilir.



Şekil 5.7. Diploma Doğrulama Ekranı

BÖLÜM 6

SONUÇLAR

Klasik sistemlerdeki veri merkeziyetçiliği veriye olan güvenin sorgulanmasına yol açmaktadır. Merkezi bir sunucuda veya dağıtık bir veritabanında tutulan veriler her zaman müdahaleye açıktır. Güven teminatı veren üçüncü tarafların ortaya çıkmasıyla tarafların giderleri artmaktadır. Bu durumda bile veriye duyulan güven tartışılır durumdadır. Bu eksikliklere çözüm olarak önümüze çıkan blokzinciri teknolojisinin gelişmesiyle birlikte merkezi olmayan uygulamaların ve akıllı sözleşmelerin veriye olan güven açısında giderek önem kazandığı görülmektedir.

Bu tez çalımasında gelişmekte olan ve kullanım alanları yaygınlaşan blokzinciri teknolojisi ile ilgili kapsamlı bir literatür araştırması yapılmıştır. Blokzinciri teknolojisiyle birlikte popülerleşmeye başlayan akıllı sözleşmeler, akıllı sözleşme koşabilen blokzinciri platformaları ve akıllı sözleşme yazılan programlama dilleri hakkında bilgiler verilmiştir.

Güvenilirliği klasik uygulama çözümlerine göre tartışılamaz derecede ön planda olan Merkezi Olmayan Uygulamalar (DAPP) hakkında literatür araştırması yapılmıştır. Bütün bu literatür araştırmalarının akabininde bir lokal Ethereum özel blokzincirinde koşan, Solidity programlama dili ile geliştirilmiş bir akıllı sözleşme içeren DAPP geliştirilmiştir. Geliştirilen DAPP ile üniversiteler mezun ettiği öğrencilerin diplomalarını blokzincirinde değiştirilemez, geriye dönük takip edilebilir ve şeffaf bir şekilde saklayabileceklerdir. İşverenler iş başvurusunda bulunan mezun öğrencilerin kendilerine sundukları diplomaların geçerliliklerini bu blokzincirinden geliştirilen DAPP aracılığıyla sorgulayabileceklerdir. Diploma verileri merkezi bir veritabanında tutulmayıp blokzinciri içerisinde kaydedildiğinden dolayı herhangi bir müdahaleye izin vermez. Bu uygulama sayesinde haksız kazanç sağlanmasının, diploma dolandırıcılığının önüne geçilecektir.

KAYNAKLAR

Bach, L. M., Mihaljevic, B., & Zagar, M. (2018, May). Comparative analysis of blockchain consensus algorithms. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 1545-1550). IEEE.

Bach, L. M., Mihaljevic, B., & Zagar, M. (2018, May). Comparative analysis of blockchain consensus algorithms. In 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 1545-1550). IEEE.

Baliga, A. (2017). Understanding blockchain consensus models. In Persistent.

Bashir, I. (2017). *Mastering blockchain*. Packt Publishing Ltd.

Becker, G. (2008). Merkle signature schemes, merkle trees and their cryptanalysis. *Ruhr-University Bochum, Tech. Rep*.

Bentov, I., Lee, C., Mizrahi, A., & Rosenfeld, M. (2014). Proof of Activity: Extending

Bitcoin's Proof of Work via Proof of Stake. *IACR Cryptology ePrint Archive*, *2014*, 452.

Bentov, I., Lee, C., Mizrahi, A., & Rosenfeld, M. (2014). Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake. IACR Cryptology ePrint Archive, 2014, 452.

Bernauer, A., Blummer, T., Kfir, S., Litsios, J., & Meier, S. (2018). U.S. Patent Application No. 15/210,668.

Burchert, C., Decker, C., & Wattenhofer, R. (2018). Scalable funding of Bitcoin micropayment channel networks. Royal Society open science, 5(8), 180089.

Buterin, V. (2014). A next-generation smart contract and decentralized application platform. *white paper*.

Castro, M., & Liskov, B. (1999, February). Practical Byzantine fault tolerance. In OSDI (Vol. 99, pp. 173-186).

Castro, M., & Liskov, B. (2003). U.S. Patent No. 6,671,821. Washington, DC: U.S. Patent and Trademark Office.

Chen, L., Xu, L., Shah, N., Gao, Z., Lu, Y., & Shi, W. (2017, November). On security analysis of proof-of-elapsed-time (poet). In International Symposium on Stabilization, Safety, and Security of Distributed Systems (pp. 282-297). Springer, Cham.

Chepurnoy, A., Kharin, V., & Meshkov, D. (2018). Self-reproducing Coins as Universal Turing Machine. In Data Privacy Management, Cryptocurrencies and Blockchain Technology (pp. 57-64). Springer, Cham.

Christidis, K., & Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. Ieee Access, 4, 2292-2303.

Dai, P., Mahi, N., Earls, J., & Norta, A. (2017). Smart-contract value-transfer protocols on a distributed mobile application platform. URL: https://qtum.org/uploads/files/cf6d69348ca50dd985b60425ccf282f3. pdf.

Dorri, A., Kanhere, S. S., Jurdak, R., & Gauravaram, P. (2017, March). Blockchain for IoT security and privacy: The case study of a smart home. In 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops) (pp. 618-623). IEEE.

Elsden, C., Manohar, A., Briggs, J., Harding, M., Speed, C., & Vines, J. (2018, April). Making sense of blockchain applications: A typology for HCI. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (p. 458). ACM.

Fröwis, M., Fuchs, A., & Böhme, R. (2018). Detecting Token Systems on Ethereum. arXiv preprint arXiv:1811.11645.

Girasa, R. (2018). Regulation of Cryptocurrencies and Blockchain Technologies: National and International Perspectives. Springer.

Golan Gueta, G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M. K., ... &

Tomescu, A. (2018). SBFT: a scalable decentralized trust infrastructure for blockchains. *arXiv preprint arXiv:1804.01626*.

Guo, Y., & Liang, C. (2016). Blockchain application and outlook in the banking industry. Financial Innovation, 2(1), 24.

http://ethdocs.org/en/latest/ethereum-clients/

http://lightning.network/docs/ , Erişim Tarihi: 02.01.2019

https://blockscout.com/eth/goerli/ , Erişim Tarihi: 02.01.2019

https://cardanodocs.com/technical/plutus/introduction/ , Erişim Tarihi: 11.01.2019

https://developer.rchain.coop/tutorial/ , Erişim Tarihi: 11.01.2019

https://docs.wavesplatform.com/en/technical-details/ride-language/maven-compiler.html , Erişim Tarihi: 15.01.2019

https://eos.io/ , Erişim Tarihi: 03.01.2019

https://ethfiddle.com/ , Erişim Tarihi: 07.01.2019

https://exonum.com/ , Erişim Tarihi: 03.01.2019

https://github.com/djrtwo/evm-opcode-gas-costs , Erişim Tarihi: 01.01.2019
https://github.com/ethereum/remix-ide , Erişim Tarihi: 07.01.2019
https://github.com/ethereum/serpent , Erişim Tarihi: 08.01.2019
https://github.com/ethereum/vyper , Erişim Tarihi: 07.01.2019
https://github.com/flintlang/flint , Erişim Tarihi: 09.01.2019
https://github.com/MaiaVictor/Formality, Erişim Tarihi: 11.01.2019
https://github.com/MichaelBurge/pyramid-scheme , Erişim Tarihi: 09.01.2019
https://github.com/OCamlPro/liquidity/blob/master/docs/liquidity.md , Erişim Tarihi: 09.01.2019
https://github.com/pirapira/bamboo , Erişim Tarihi: 07.01.2019
https://github.com/s-tikhomirov/smart-contract-languages , Erişim Tarihi: 09.01.2019
https://infura.io/ , Erişim Tarihi: 18.01.2019
https://kovan.etherscan.io/ , Erişim Tarihi: 02.01.2019
https://lisk.io/ , Erişim Tarihi: 03.01.2019
https://mcoblenz.github.io/Obsidian/ , Erişim Tarihi: 12.01.2019
https://metamask.io/ , Erişim Tarihi: 15.01.2019
https://nebl.io/ , Erişim Tarihi: 03.01.2019
https://neo.org/ , Erişim Tarihi: 03.01.2019
https://qtum.org/en , Erişim Tarihi: 02.01.2019
https://snax.one/whitepaper.pdf , Erişim Tarihi: 02.01.2019
https://sokol-explorer.poa.network/ , Erişim Tarihi: 02.01.2019
https://solidity.readthedocs.io/en/v0.5.3/ , Erişim Tarihi: 07.01.2019
https://superblocks.com/lab/ , Erişim Tarihi: 07.01.2019
https://tezos.com/ , Erişim Tarihi: 02.01.2019
https://theethereum.wiki/w/index.php/ERC20_Token_Standard
https://tobalaba.etherscan.com/ , Erişim Tarihi: 02.01.2019
https://tron.network/static/doc/white_paper_v_2_0.pdf, Erişim Tarihi: 02.01.2019
https://truffleframework.com/docs , Erişim Tarihi: 15.01.2019
https://ubiqsmart.com/ , Erişim Tarihi: 02.01.2019
https://wavesplatform.com/files/docs/white_paper_waves_smart_contracts.pdf?cache=b , Erişim Tarihi: 15.01.2019
https://wiki.parity.io/Parity-Ethereum , Erişim Tarihi: 02.01.2019
https://www.rsk.co/ , Erişim Tarihi: 03.01.2019

Hu, Y. C., Lee, T. T., Chatzopoulos, D., & Hui, P. (2018, June). Hierarchical interactions between ethereum smart contracts across testnets. In Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems (pp. 7-12). ACM.

Iyer, K., & Dannen, C. (2018). The ethereum development environment. In Building Games with Ethereum Smart Contracts (pp. 19-36). Apress, Berkeley, CA.

Kasampalis, T., Guth, D., Moore, B., Serbanuta, T., Serbanuta, V., Filaretti, D., ... & Johnson, R. (2018). IELE: An Intermediate-Level Blockchain Language Designed and Implemented Using Formal Semantics.

Khan, M. A., & Salah, K. (2018). IoT security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, *82*, 395-411.

Kiayias, A., Russell, A., David, B., & Oliynykov, R. (2017, August). Ouroboros: A provably secure proof-of-stake blockchain protocol. In Annual International Cryptology Conference (pp. 357-388). Springer, Cham.

Kobeissi, N., & Kulatova, N. (2018, April). Ledger Design Language: Designing and Deploying Formally Verified Public Ledgers. In Workshop on Security Protocol Implementations: Development and Analysis.

Kroll, J. A., Davey, I. C., & Felten, E. W. (2013, June). The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In Proceedings of WEIS (Vol. 2013, p. 11).

Lazarovich, A. (2015). Invisible Ink: blockchain for data privacy (Doctoral dissertation, Massachusetts Institute of Technology).

Lindman, J., Tuunainen, V. K., & Rossi, M. (2017). Opportunities and risks of Blockchain Technologies– Araştırma Raporu.

Mitchell, K. (2019). Blockchain IPOs: capital raising in a crypto-world. Without Prejudice, 19(1), 7-8.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

O'Dwyer, K. J., & Malone, D. (2014). Bitcoin mining and its energy footprint.

Parizi, R. M., & Dehghantanha, A. (2018, June). Smart Contract Programming Languages on Blockchains: An Empirical Evaluation of Usability and Security. In International Conference on Blockchain (pp. 75-91). Springer, Cham.

Pettersson, J., & Edström, R. (2016). Safer smart contracts through type-driven development. Master's thesis. Chalmers University of Technology, Sweden.

Popejoy, S. (2016). The Pact smart contract language. June-2017.[Online]. Available: http://kadena.io/docs/Kadena-PactWhitepaper.pdf.

Radanović, I., & Likić, R. (2018). Opportunities for Use of Blockchain Technology in Medicine. Applied health economics and health policy, 16(5), 583-590.

Sagirlar, G., Carminati, B., Ferrari, E., Sheehan, J. D., & Ragnoli, E. (2018). Hybrid-iot: Hybrid blockchain architecture for internet of things-pow sub-blockchains. arXiv preprint arXiv:1804.03903.

Salimitari, M., & Chatterjee, M. (2018). An Overview of Blockchain and Consensus Protocols for IoT Networks. arXiv preprint arXiv:1809.05613.

Seijas, P. L., & Thompson, S. (2018, November). Marlowe: Financial Contracts on Blockchain. In International Symposium on Leveraging Applications of Formal Methods (pp. 356-375). Springer, Cham.

Sergey, I., Kumar, A., & Hobor, A. (2018). Scilla: a smart contract intermediate-level language. arXiv preprint arXiv:1801.00687.

Tariq, R., Aadil, F., Malik, M. F., Ejaz, S., Khan, M. U., & Khan, M. F. (2018, September). Directed Acyclic Graph Based Task Scheduling Algorithm for Heterogeneous Systems. In Proceedings of SAI Intelligent Systems Conference (pp. 936-947). Springer, Cham.

Tasatanattakool, P., & Techapanupreeda, C. (2018, January). Blockchain: Challenges and applications. In 2018 International Conference on Information Networking (ICOIN) (pp. 473-475). IEEE.

Usta, A., & Doğantekin, S. (2017). Blockchain 101. *Kapital Medya Hizmetleri A. fi*.

Wüst, K., & Gervais, A. (2018, June). Do you need a Blockchain?. In 2018 Crypto Valley Conference on Blockchain Technology (CVCBT) (pp. 45-54). IEEE.

Yang, Z., & Lei, H. (2018). Lolisa: Formal syntax and semantics for a subset of the solidity programming language. arXiv preprint arXiv:1803.09885.

Zheng, Z., Xie, S., Dai, H. N., & Wang, H. (2016). Blockchain challenges and opportunities: A survey. *Work Pap.–2016*.

Zheng, Z., Xie, S., Dai, H. N., & Wang, H. (2016). Blockchain challenges and opportunities: A survey. Work Pap.–2016.

Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017, June). An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE international congress on big data (BigData congress)* (pp. 557-564). IEEE.

Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017, June). An overview of blockchain technology: Architecture, consensus, and future trends. In 2017 IEEE international congress on big data (BigData congress) (pp. 557-564). IEEE.

ÖZGEÇMİŞ

KEREM ATAŞEN

Doğum Tarihi: 06.11.1992

EĞİTİM

1999-2006: Akarca Türk-Fransız Kardeşlik İlköğretim Okulu (Söğütlü/SAKARYA)

2006-2010: Bolu Fen Lisesi (Bolu)

2010-2015: Ankara Üniversitesi Mühendislik Fakültesi, Bilgisayar Mühendisliği (Ankara)

2017-……: Trakya Üniversitesi Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı (Edirne)