Daron DEDEOĞLU

KODLAB.TV



OKU, İZLE, DİNLE, ÖĞREN!

A'DAN Z'YE

BLOCKCHAIN

İNTERAKTİF EĞİTİM VİDEOLARI KODLAB.TV 'DE!

KODLAB®
www.kodlab.com

KODLAB.tv

Kitabın yanında "Hediye" olarak verilen eğitim videolarını
www.kodlab.tv adresinden aşağıdaki adımları uygulayarak izleyebilirsin.



1 **WWW.KODLAB.TV ADRESİNE GİR**



2 **ÜYE OL!**



3 **KİTABINI SEÇEREK AKTİVASYON KODUNU GİR**

AKTİVASYON KODU

A'DAN Z'YE
BLOCKCHAIN

Daron Dedeoğlu

KODLAB®
Yayın Dağıtım Yazılım ve Eğitim
Hizmetleri San. ve Tic. Ltd. Şti.

A'DAN Z'YE BLOCKCHAIN

DARON DEDEOĞLU

ISBN 978-605-2118-42-9
Yayıncılık Sertifika No: 13206

1. Baskı: Temmuz 2019

Genel Yayın Yönetmeni
Hüseyin Üstünel

Grafik Tasarım
Mustafa Said Şahin

Yayına Hazırlık
Merve Nur Şengül

Editörler
İrem Soylu
Gizem Aksan

Sosyal Medya Sorumlusu
Mahmut İpek

Satış ve Pazarlama
Can Üstünel
Yavuz Şengönül

Baskı: İnkılâp Kitabevi Yayın San. Tic. A.Ş. Tel: (0212) 496 11 81
Çobançeşme Mah. Sanayi Cad. Altay Sk. No:8 Yenibosna/İSTANBUL
Sertifika No: 44066

Bu kitabın bütün yayın hakları Kodlab Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti.'ne aittir. Yayınevimizin yazılı izni olmaksızın kısmen veya tamamen alıntı yapılamaz, kopya çekilemez, çoğaltılamaz ve yayınlanamaz.

KODLAB Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti.
Merkezefendi Mah. Mevlana Cad. Tercüman Sit. B2 Blok K:6 D:26 Zeytinburnu / İSTANBUL

tel: 0(212) 514 55 66 **web:** www.kodlab.com
0(212) 514 66 61 **e-posta:** bilgi@kodlab.com

Daron Dedeoğlu

İktisat eğitimini Anadolu Üniversitesinde, işletme yüksek lisansını İstanbul Bilgi Üniversitesinde tamamlamıştır. Türkiye'nin ilk Türkçe içerikli eğlence portalı olan Kahkaha.com'un yayına başlamasıyla bilişim hayatına atılmıştır. Ardından uzun yıllar Byte, PC World, PCnet gibi Türkiye'nin önde gelen bilişim yayınlarında editörlük görevini üstlenmiş ve RIM (BlackBerry), Yandex gibi uluslararası teknoloji şirketlerinde geliştirici ilişkileri yöneticiliği görevi yapmıştır. Son yıllarda Blockchain teknolojisi ve kripto para tabanlı ödeme sistemleri üzerine çalışmalarını sürdürmektedir.

Yazım sürecinde bana destek olan eşim ve kızıma ithafen...

ÖN SÖZ

Dünya değişiyor. Web 1.0 ile başlayan ve hayatımızı kökünden değiştiren, internet tabanlı evrim çağlarını hızlıca tüketiyoruz. Artık YouTube'un, Facebook'un, Twitter'ın ya da Whatsapp'ın doğduğu çağda yaşamıyoruz. Sosyal ağların damgasını vurduğu Web 2.0 çağını çoktan geçtik. Arama motorlarının akıllandığı (semantic web), kullanıcının zevklerini ve alışkanlıklarını "öğrenerek" kişiselleştirilmiş sonuçlarla buluşmasını sağlayan Web 3.0 çağı da noktalandı. Artık sanal ve uzatılmış gerçeklik ile harmanlanmış yeni nesil internet çağı olan Web 4.0'ın içerisindeyiz; ancak kaçımız bunun farkında?

Nasıl ki Orta Çağ'da yaşamış olan insanlar, aslında Orta Çağ diye tanımladığımız o zaman aralığında bulunduklarının farkına varmamışlarsa biz de internetin çağlara ayrılmış kült dönemlerinin hayatımızdan akıp gidişinin farkına varamıyoruz. Üstelik şimdilerde hayatımıza sessiz sedasız girmiş olan başka bir teknoloji ile Web 4.0'ı ve interneti farklı bir perspektiften izleme fırsatı yakalıyoruz: Merkeziyetsiz yapılar! Merkezî olmayan, Web 2.0 döneminde doğup adeta interneti çatallayarak (fork) kendi alternatif evrenini yaratan "internet'in parası", "dijital para" ya da nam-ı diğer kripto para perspektifinden bahsediyorum.

Blockchain, kripto para, ICO, akıllı kontrat, madenci… Evet, Bitcoin hayatımıza girdiğinden beri bunlar gibi birçok anahtar sözcükle tanıştık. Haberlerde, gazetelerde, dergilerde, sosyal ağlarda kısacası kafamızı çevirdiğimiz her yerde bu ve benzeri Bitcoin evrenine ait anahtar sözcükleri görüyoruz. Kendi başına bir yapı olarak doğuşunun ardından, onu taban alan klonlar (altcoin'ler) ve başlı başına kendi içerisinde bağımsızlaşmış borsalar, yatırımcılar, geliştiriciler ve uzaktan izlemeyi tercih eden meraklılarıyla koskoca bir evren! Peki, bu evrene ait terimleşmiş sözcüklerin ne kadarının anlamını gerçekten biliyoruz? Ya da Bitcoin dünyasının hayatımıza kattığı ve gölgesinde kalmaya başladığı Blockchain teknolojisini ne kadar tanıyoruz?

Bu kitabı, son yıllarda adını sıklıkla duyduğumuz Bitcoin sayesinde tanıdığımız ve birçok otoriteye göre Bitcoin'den çok daha devrimsel bir yapı olan Blockchain (dilimize blok zinciri olarak geçmiştir) teknolojisini her yönüyle anlatmayı amaçlayarak kaleme aldım. Kitabı okurken "Blockchain teknolojisi nedir?", "Nasıl gelişmiştir?", "Kaç çeşit Blockchain vardır?", "Neden Blockchain'e ihtiyaç duyarız?" ve "Blockchain'i iş hayatımızda nasıl kullanabiliriz?" gibi sorulara yanıtlar bulacaksınız. Ayrıca projelerinizde ya da iş fikirlerinizde Blockchain teknolojisine yer vermek isterseniz, basit adımlarla nasıl kendi Blockchain ağınızı kuracağınızı örnek kod bloklarıyla anlattığım uygulama bölümlerini çok beğeneceğinizi düşünüyorum.

Kitapta Bitcoin, Ethereum ve IOTA gibi kendi Blockchain yapılarını oluşturmuş kripto paralara da yer verdim (bu üç Blockchain teoride aynı amaca hizmet etmekle birlikte geliştirilme yapıları ve yetenekleri birbirinden oldukça farklıdır). Ancak herhangi bir yatırım önerisi ya da geleceğe dönük "tüyo" bekliyorsanız bu kitabın bunu amaçlamadığını baştan belirtmek isterim.

Bu kitabı, genel okur kitlesini hedefleyerek yazdığımdan herhangi bir iktisadi ya da yazılım geliştirme bilgi birikimine sahip olmanız gerekmiyor. Ancak, komplike bir Blockchain ağı geliştirmek ya da Blockchain ile yüksek yapılı projeler tasarlamak istiyorsanız, başta GNU/Linux işletim sistemi olmak üzere, C / C++ dillerinden en az birine ve kriptografi konusuna aşina olmanızı şiddetle tavsiye ederim.

Şimdi sözü fazla uzatmadan gelin, beraberce "Blockchain nedir?" sorusuna yanıt bulalım.

Daron DEDEOĞLU

Kitabın Amacı ve Hedef Kitlesi

Çok yeni bir kavram olan Blockchain ya da dilimize geçiş yaptığı şekliyle blok zinciri, kripto paralarla anılan yeni nesil bir veritabanı türevi olarak karşımıza çıkıyor. Bu yapı hakkında yazılıp çizilenlere baktığımızda, dezenformasyonların yanı sıra genellikle herhangi bir kripto paranın odağında yapılan anlatımlarla karşılaşıyoruz. Bu nedenle konuya ilgi duyan geliştiriciler ya da girişimciler kripto para gölgesinde kalan bir yapı ile baş başa kaldıklarını düşünebiliyorlar. Daha da kötüsü, ulaşabildikleri kaynaklarda adı geçen kripto paralarla çalışmak zorunda oldukları gibi yanlış yönelimlere kapılabiliyorlar. Bu kitabı kaleme alış amaçlarımdan birisi de gündemi en çok meşgul eden tartışmaların başında gelen, "Kripto parasız Blockchain olabilir mi?" sorusuna yanıt vermeye çalışmaktır. Bu sorunun cevabını aldığınızda hem Blockchain teknolojisine hem kripto paralara hem de geliştirmeye çalıştığınız iş modelleri odağında yeni ekonomiye farklı bir perspektiften bakmayı öğreneceksiniz.

Bu kitap, kripto paralarla hayatımıza giren Blockchain teknolojisi hakkında net ve doğru bilgiler vererek gerek yatırım piyasasındaki gerekse de geliştirici dünyasındaki dezenformasyonları ortadan kaldırma amacıyla kaleme alındı. Kitabın ilerleyen bölümlerinde Blockchain'in ne olduğunu, türevlerini, temel prensiplerini ve kullanım alanlarını detaylı bir şekilde açıklamaya çalıştım. Aynı zamanda "Blockchain gerçekten kırılamaz mı?", "Kuantum bilgisayarlar Blockchain için risk mi?", "yüzde 51 saldırısı ne demek?", "Blockchain gerçekten merkeziyetsiz mi?" gibi çok tartışılan sorulara yanıt vermeye çalıştım.

Hedefini Blockchain dünyasına çevirmiş olan yazılım geliştiricilerinin, kod blokları arasında dolaşıp Blockchain yapısının nasıl çalıştığını görüp anlayabilecekleri workshop bölümlerini çok beğeneceklerini düşünüyorum. Bu aşamada hemen belirteyim; Blockchain 1.0 tarafında genel olarak kullanacağım dil yapısı, yılların eskitemediği C++ olacak. Çünkü bize bu teknolojinin kapılarını açan Bitcoin ve Ethereum bu dil kullanılarak geliştirilmişti. Elbette farklı diller kullanarak kendi özel Blockchain yapılarınızı oluşturabilmeniz de mümkün. Ancak bu konu oldukça derin ve detaylı olduğundan bu kitabın amacını aşıyor. Belki o konuyu da ileride kaleme alırım. Kitaptaki workshop bölümlerinde kaleme aldığım web tabanlı uygulamalarda PHP; akıllı sözleşmeleri geliştirirken de Ethereum'a ait olan Solidity dillerini kullanarak örnekler paylaşmaya çalıştım. Tabii ki amacım size yol göstermek ve kendi iş fikirleriniz üzerinde Blockchain yapısını hayata geçirebileceğiniz cesareti bulmanızı sağlamak olacak. Belki de bu kitabı okuduktan sonra, mevcut iş modelinizin Blockchain olmadan daha efektif kullanılabileceğine kanaat getireceksiniz. Yani bu kitapla sizleri bir hedefe kilitlemek istemiyorum; aksine, konuya hâkimiyetinizi artırıp kendi kararınızı verebilmenizi hedefliyorum. Bu nedenle sizlere objektif olarak Bitcoin ile tanıdığımız Blockchain teknolojisini, Ethereum ile hayatımıza giren Blockchain teknolojisini ve IOTA ile tanıdığımız Tangle teknolojisini anlatmaya çalıştım.

Ben kişisel tercihlerim doğrultusunda GNU/Linux işletim sistemi kullanmayı seviyorum. Kitabımda da örneklemelerimi GNU/Linux türevi olan ve kullanıcı dostu olduğunu düşündüğüm Ubuntu Linux dağıtımı üzerinde anlattım. Tabii dilerseniz Debian, MacOS X ve hatta bir iki eklenti kurarak Windows üzerinde de çalışabilirsiniz. Ancak kendi deneyimlerime dayanarak söyleyebilirim ki ciddi projeleri emülatörler ile geliştirmeye kalkmak ciddi riskleri beraberinde getirir. Ayrıca bu kitap dışında kalan online ya da basılı kaynaklara bakacak olursanız yine geliştirme ortamlarında hâkim olan sistemin GNU/Linux türevleri olduğunu göreceksiniz. Bu nedenle GNU/Linux konusunda acemi ve/veya hiç deneyimi olmayan bir kullanıcıysanız size acilen bu işletim sistemini öğrenmeye başlamanızı tavsiye ederim.

Bu kitap ayrıca, yatırımcıları ve yatırımcı olarak bu dünyaya giriş yapmayı amaçlayan kişileri de hedefliyor. Blockchain ve kripto paralarla birlikte hayatımıza giren ICO (initial coin offering) ve yeni yeni duymaya başladığımız IEO (initial exchange offering) gibi kavramlara yakından bakarak şirketlerin hayatlarında neleri değiştirebileceğine değineceğiz. Erken dönem yatırım yapmayı seven yatırımcıların ICO mantığına nasıl yaklaşması gerektiğini ve yatırım yapmayı hedeflediğiniz bir ICO'nun nelere sahip olması gerektiğini sizlerle paylaşacağım. Yalnız hemen belirteyim ki; bu kitapta özellikle kripto para odaklı yatırım önerileri bulunmayacak. Yani şu "coin" yükselecek ya da bu "token" gelecek vaat ediyor gibi paylaşımların veya önerilerin peşindeyseniz bu kitapta aradığınızı bulamayacaksınız, üzgünüm. Ama paranızı bu dünyaya akıtacaksanız en azından Blockchain ve kripto para hakkında doğru bilgilere sahip olmanız, piyasayı doğru okumanızı sağlayacaktır, diye düşünüyorum. Unutmadan ekleyeyim; kitapta ayrıca "coin" ve "token" arasındaki temel farklara da değindim. Keyifli okumalar.

İÇİNDEKİLER

1 ESKİDEN YENİYE KISA BİR YOLCULUK 1
Para Ve Muhasebenin Kısa Tarihi 2
Saatler Blockchain'e 1 Var! 13
Veri Ve Veritabanlarının Gelişimi 14
Kriptoloji 21
Hash 22
Merkle Ağacı (Merkle Tree) 24
İkinci Preimage (Ters Görüntü) Saldırısı 27
Kaplan Ağacı Özeti 27
Merkezsiz Yapılar 28
Blockchain Nedir? 30
Blok 31
Blok Süresi 32
Hard Fork (Ağı Çatallamak) 33
Merkezsizleştirme 34
Açıklık 35
İzin Gerektirmeyen Blockchain 36
İzin Gerektiren (Özel) Blockchain 37
Blockchain'i Doğuran Sebepler 38
Blockchain Kripto Parasız Olur Mu? 40
Blockchain Mimarileri 42
Bitcoin Ağı (Blockchain 1.0) 43
Zaman Damgası Sunucusu 44
İş Kanıtı (Proof Of Work) 45
Ağın İşletme Adımları 46
Teşvik 47
Disk Alanı 48
Ödeme Doğrulaması 49

Değerleri Birleştirme ve Bölme 50
Gizlilik 51
Ağın Güvenliği 52
Ethereum Ağı 56
Token 58
Ethereum Virtual Machine Ve Akıllı Sözleşmeler 61
Solidity Programlama Dili 63
İşlem Ücretleri ve Gas 64
IOTA Ağı: Tangle 65
Ölçeklenebilir Olmak 67
Kuantum Bilgisayarlı Hacker 68
Çevrimdışı Tangle Kümeleri 69
Son Söz 70

2 YENİ EKONOMİ VE BLOCKCHAIN 71
Blockchain Hangi İş Modellerine Çözüm Oluyor? 73
Kendi Blockchain Ağımız Olmalı Mı? 75

3 KENDİ KAYNAĞINI ÜRETMEK 77
Kripto Paralar 78
Kripto Paraların Artıları 81
Kripto Paraların Eksileri 83
Hayatımıza Giren İlk Kripto Para Bitcoin Mi? 84
Kripto Paraları "Değer"lendirmek 88
Kripto Paranın Anatomisi:
Algoritmalar, Ödül Dağıtım Yöntemleri ve Madencilik Tipleri 93
Mutabakat Yapısı ve Süreci 94
Emeğin İspatı: Proof Of Work 95
Sahipliğin İspatı: Proof Of Stake 97
Practical Byzantine Fault Tolerance - PBFT 99
Fork (Çatallanma) 101

Madencilik Yapıları ve Kaynak Tüketimi 103
Coin vs Token: Hangisi Gerçek Para? 108
Stable Coin: Karşılık Para 110
ICO ve IEO 111
Nasıl ICO Yapılır? 113
ICO'ların Olası Riskleri 115

4 BLOCKCHAIN VE GÜVENLİK 117
Yüzde 51 Saldırısı 118
Kuantum İşlemci Riski 121
Son Söz 122

5 WORKSHOP: KOD YAZMA ZAMANI! 123
Nasıl Bir Blockchain? 123
İhtiyaç Listesi 124
Linux Konsol Komutları 127
pwd 128
ls 128
cd 129
mkdir 129
rmdir 129
touch 129
man &-help 129
cp 129
mv 130
nano ve vi 130
sudo 130
apt-get 130
wget 130
ifconfig 130
Kodlab Blockchain 131

Blockchain 1.0 ve Web 174
Blockchain 2.0 ve Ethereum 182
Solidity Programlama Dili 182
Hızlandırılmış Solidity 186
Sözleşme Dediğimiz Uygulamalar 191
İlk Akıllı Sözleşmemiz 208

1

ESKİDEN YENİYE KISA BİR YOLCULUK

Bitcoin adlı kripto paranın hayatımıza girmesiyle birlikte adını duymaya başladığımız **Blockchain teknolojisi**, sessiz sedasız geçirdiği yılların ardından, özellikle takvimler 2017 yılını gösterdiğinde yavaş yavaş gölgelerden sıyrılıp endamını göstermeye başladı. Kripto paraların durmak bilmeyen yükselişi yatırımcılarını sarhoş ederken geliştiriciler ve fütüristik ekonomistler gözlerini kripto paralardan ziyade, **ardında yatan teknoloji**ye odaklamaya başladılar; yani Blockchain teknolojisine. Aslında ilk kripto para olan Bitcoin tasarlanmaya başlandığında, ardında Blockchain teknolojisi yoktu. Hatta, Bitcoin'in o dönemdeki en büyük sıkıntısı çift harcamaların nasıl önüne geçileceğinin tam olarak bilinmemesiydi. İşte tam da bu noktada Bitcoin'in harcama ve muhasebeleştirme aşamalarındaki sıkıntıları çözebilecek bir yapı arayışında olan **Satoshi Nakamoto** (*Bu isim gerçekte yaşayan bir insana ait olmakla birlikte Bitcoin ile bir ilişkisi olmadığı tespit edilmiştir. Şimdilerde Bitcoin'i geliştiren ya da geliştirenlerin Satoshi Nakamoto adını "nick name" olarak kullandıkları tezi hâkimdir. Aksi ispatlanana kadar ben de bu tezi kabul edeceğim.*), temelleri 1991 yılında **Stuart Haber** ve **W. Scott Stornetta** tarafından atılan "**Merkle Ağacı**" yöntemini Bitcoin'in temel yapısı içine entegre ederek günümüzün Blockchain tabanlı ilk kripto parasını çifte harcama riskinden kurtarmış oldu. Bu satırları okuduğunuzda aklınıza bazı sorular takılabilir; mesela, "*Bitcoin'in temelleri daha önce atılmış olabilir mi?*" gibi... **Bitcoin öncesi mikro ödeme yapıları** hakkında detaylara kitabın ilerleyen sayfalarında yer vereceğim.

Şimdilik net olan durum şu: Satoshi Nakamoto kendinden önce yapılmış olan bağımsız çalışmaları, kendi ideası merkezinde çok iyi harmanlayıp Bitcoin merkezinde kavramsallaştırmıştır. Bu nedenle Bitcoin ve Blockchain yapısını kendisine atfetmekte sıkıntı görmüyorum.

Bitcoin ve onun güven sıkıntısını çözen altyapısı Blockchain, geliştirici dünyası ve fütüristler için devrimsel bir çağı başlatmış olsa da eski nesil ekonomistler ve iktisatçıların gözünde hâlâ büyük bir soru işareti var: O da kripto paraların sahip oldukları değer karşılığının dayandığı **temeller**. Kripto paraları basından takip etmiş olan okurlar "*Bitcoin bir balon mu?*" benzeri başlıkları hatırlayacaklardır. Aslında her ne kadar bu tarz soruları **eski nesil** diye tabir ettiğim ekonomistlere atfetsem de kripto para dünyasına adım atmış olan hemen her kesimden kişinin aklının bir köşesinde bu soru, ayakları yere basan bir cevap bulmaya çalışıyor. Bu kitabı kaleme alırken cevaplamaya çalıştığım sorulardan belki de en önemlisi buydu diyebilirim. Ancak bu sorunun cevabını net bir şekilde verebilmek için öncelikle para, ticaret ve muhasebe kavramlarının nasıl geliştiğine kısaca değinmemiz gerekiyor.

PARA VE MUHASEBENİN KISA TARİHİ



Şekil 1.1: Bitcoin bize insan eliyle yaratılan, tamamen dijital ve karşılığı ayrılmamış bir yapının, yakın gelecekte itibari para olarak kullanılabileceğini gösterdi.

Bir Blockchain kitabında tarih, para ve muhasebeden bahsetmemiz size garip gelebilir; ama iktisadi enstrümanlar hakkında konuşurken ister istemez tarihe bakmamız gerekiyor. Hele ki insan eliyle geliştirilip değeri bilfiil onu kabullenen kitle tarafından atanmış olan kripto paranın, günümüz parasının evrimleşmiş hâli olduğunu savunuyorsak para dediğimiz kavramın doğuşunu ve gelişimini irdelememiz gerekiyor. "*Para hayatımıza ne zaman girdi? Hangi problemi çözdü ve o dönemdeki insanlar bu yeni aracı ne sürede benimsediler?*" gibi soruların cevaplarını, Bitcoin'in hayatımıza girmesinden sonra, aynı soruların kripto para özelindeki cevaplarıyla kıyaslamalıyız. Tahmin ettiğiniz üzere aynısını **genel geçer muhasebe sistemi**miz ve **Blockchain** özünde de yapacağız. Bu bölümün sonunda, "*Kripto para günümüz fiyat paraların evrimindeki son faz mı? Kripto paraların ardında toplanan para bir gün buhar olur mu? Kripto para dediğimiz rakamlardan ve hash'lardan oluşan olgu bir balon mu? Muhasebe sistemimiz ne kadar yeterli? Blockchain 21. yüzyılın yeni muhasebe sistemi olabilir mi?*" gibi soruların cevaplarını bulmuş olacağız. Fakat endişelenmeyin; tarihsel ve sıkıcı gibi duran bu fazı oldukça yüzeysel ve kısa tutacağım.

İnsanoğlu yerleşik hayata geçiş yaptığı andan itibaren sistematik olarak üretmeye ve ürettiklerini de üret(e)medikleriyle değiş-tokuş yapmaya başlamıştır. Bu şekilde **ticaret** dediğimiz olgu, insan eliyle ve insani ihtiyaçların karşılanması hedefiyle hayata geçirilmiş oldu. Ticaretin gün yüzüne çıktığı o erken dönemlerde (*yaklaşık 150 bin yıl öncesinden bahsediyorum*) elbette adına **para** dediğimiz ve amacı "**değiş-tokuş**" üzerine kurulmuş basit ticaret yapısında değer belirleme aracı olmak olan **itibari ve sanal yapı** henüz kurgulanmamıştı. Ama yine de insanlar birbirleri arasında değiş-tokuş yaparken adetler üzerinden değil tamamen subjektif olarak belirlenmiş olan **karşılık** esasına bağlı kalıyorlardı. Örneğin 1 adet yumurta karşılığında 1 adet tavuk alamıyordunuz; ama belki de 1 kg buğday karşılığında 40 adet yumurta alabilirdiniz. Tabii ki o dönemlerde bu tarz karşılık esaslı değerleme biçimleri üretilen **ürünlerin miktarı** ve **temel ihtiyaçların değişkenliği** ile yoğruluyordu. Yani 1 kg buğdayın kaç tavuk ettiği aslen o an buğdayı üreten kişinin tavuğa olan ihtiyacıyla orantılıydı.



Şekil 1.2: Ticaret, değiş-tokuş mantığından günümüzün gelişmiş ticaret ağlarının yapısına varana dek büyük bir evrim geçirdi.

Arkeolojik çalışmalar bize değiş-tokuş alanlarının her uygarlığın hemen her büyük şehrinde "**pazar yerleri**" olarak kurulmuş olduğunu gösteriyor. Ancak farklı toplumlar farklı ürünler ürettiği ve farklı ihtiyaçlara sahip olduğundan değiş-tokuş, zamanla anlamını yitirmeye başladı. Sonuçta insanlık ve coğrafi keşifler gelişimini sürdürdükçe işin içine kumaşlar, aletler, farklı iklimlerde yetişmiş farklı sebze, meyve ve baharatlar girince onların değerini yumurta, tavuk ya da buğday cinsinden ölçümlemek çok kolay olmasa gerek. İşte bu noktada **otoriter bir yapı** (*ki bu noktada Lidyalılardan bahsetmekle birlikte aslen devlet elini kastediyorum*) tarafından ortaya atılmış "**para**" kavramıyla tanışıyoruz. M.Ö. 7. yüzyıla denk gelen bu dönemde **karşılığı sabit** olan bir parayla tanışmış olmamız oldukça ilginç. Çünkü o döneme kadar insanoğlu, karşılığı sabit bir değere oturmuş ve hemen her şeyin değerini belirlemek için **gösterge fiyat** olma özelliği taşıyan bir araca ihtiyaç duymamıştı. Ve bu tarz yenilikçi bir yapıyı kabullenme hızımız da oldukça yüksek olmuş. M.Ö. 7. yüzyıldan sonra hızlı bir şekilde birçok ülkenin kendi parasını basmaya başladığını görüyoruz. Bu durum, günümüz **uluslararası ticareti**nin de temellerini atmış oldu. Ancak o dönemlerde kullanılan para, hemen her uygarlığın gözünde aynı değere sahip olan **altın** gibi bir elementten yapılıyordu. Zamanla bu kervana gümüş, bakır ve bronz gibi materyaller de eklendi.

Yalnız bu noktada aklınız karışmasın, hâlâ günümüzün itibari parasından bahsetmiyoruz. Aksine, karşılığı bir madene bağlanmış, "**darp**" tekniğiyle üretilmiş, kendi öz değeri de olan madeni bir paradan bahsediyoruz.



Şekil 1.3: İlk madeni paralar darp yöntemiyle basılıyor ve kolay işlenebilen altın, gümüş gibi madenlerden imal ediliyorlardı.

Paranın itibari değerle tanışmasının temelleri ise M.Ö. 479 yılında Persler ve Yunanlılar arasında yapılan savaşın Yunanlılar tarafından kazanılmasının ardından basılan ve bu zafere atfedilen **hatıra parası**yla atılıyor. O zafer sonrası birçok farklı zafer, kutlama ve kişi adına hatıra paraları basılmaya devam etti. Böylece paranın bir başka uygulama alanı daha açılmış oldu.

Değerli metallerle para basılması elbette bir süre sonra sıkıntılı bir süreç almaya başladı. Özellikle nüfusu ve ticaret hacmi yüksek olan krallık ve imparatorluklar para basma konusunda yeni yollar ve yöntemler arayışına girdiler. Bu süreç M.S. 6. yüzyılın Çin'inde ilk **kâğıt para**nın basılmasıyla son buldu. Aslında bu kâğıt paralar ilk etapta "**para**"dan ziyade ticarette kullanılmak üzere düzenlenen "**senet**"lerdi. Ancak kısa sürede senet amacından ziyade para amacıyla kullanılmaya başlandı. M.S. 1279 yılına kadar da geçerliliğini korumayı başardı. Bu adım dünyanın geri kalanı için ticaret yapısını tamamen değiştirdi ve kendini Asya'da ispatlayan "**kâğıt para**" nihayet M.S. 1661 yılında Avrupa kıtasına sıçrayarak İsveç tarafından da basılmaya başladı. O dönemde okyanusun öbür ucundaki Amerika kıtasında yeni yeni oluşumunu tamamlamaya çalışan günümüzün Birleşik Devletleri'nde de kâğıt para mantığı, gümüş dolarların yerini almak için harekete geçti.

İlk hamle Massachusetts yönetiminden geldi ve Amerikan tarihinin ilk **banknot**u basılarak kullanıma sunulmuş oldu.

Gördüğünüz gibi aslında para dediğimiz olgunun evrimi, insanın evrimiyle doğru orantılı olarak gelişimini sürdürmüş ve sürdürmeye devam etmektedir. Günümüzde paranın nakit olarak kullanımı oldukça azalmış, gerek günlük gerekse de uluslararası ticarette para, bankalar arasında gidip gelen **rakam dizileri**nden ibaret hâle gelmiştir. Bu sürecin gelişimini tetikleyen en büyük etkense 20. yüzyıl insanını sanal parayla tanıştıran **kredi kartları** olmuştur diyebiliriz. Kredi kartları, elimizle tutmasak da hesaplarımızda duran ya da gelirimize endekslenmiş kredi limitlerimiz içerisinde kalan parayı kullanmamıza imkân veren araçlardır ve paranın evriminde mihenk taşı olabilecek kadar büyük bir rol oynamışlardır. Bugünün **e-ticaret**inin oluşmasında ve gelişmesinde de büyük payları vardır. *Peki, kredi kartları nasıl gün yüzüne çıktı dersiniz?*



Şekil 1.4: Frank McNamara'nın kişisel bir ihtiyacından doğan kredi kartının amacı, üzerimizde para olmasa da üye restoranlarda yemek yiyebilmemizdi.

Tarihte ilk kredi kartı aslında günümüzde de plaza insanının vazgeçilmezi olan yemek kartlarıyla aynı amaca hizmet ediyordu. Hatta ilginç bir de hikâyesi vardır: **Hamilton Credit Corporation** adlı, dönemin ileri gelen kredi kurumunun başkanı **Frank McNamara**, dostlarıyla yemeğe çıktığında yanında para olmadığını fark ediyor ve hem misafirlerine karşı hem de restorana karşı büyük bir mahcubiyet yaşıyor. Bu mahcubiyet sonrası hemen tüm zamanını, üzerinde para olmayan zamanlarda alışveriş problemini çözebilecek ve aynı zamanda **paranın yerini alabilecek** olan bir model geliştirmek için harcamaya başlıyor. *Sonucu tahmin ettiniz değil mi?*

Günümüz cüzdanlarının vazgeçilmezi olan kredi kartı ortaya çıkıyor. İlk kredi kartının anatomisi günümüzdeki gibi plastik değildi. Kalın bir kartondan oluşuyordu. Arkasında ise bu yeni "**ödeme aracı**" ile kabul anlaşması yapmış olan New York'un önde gelen 14 restoranının adı yazıyordu. Bu kartı kullanmak için de günümüzde olduğu gibi **yıllık aidat** ödenmesi gerekiyordu. Bu bedel, o zamanın ekonomik şartları düşünülerek 3 dolar olarak belirlenmişti.

Kredi kartının hayatımıza girdiği o andan itibaren hemen tüm banka ve kredi kuruluşları kendi kart modellerini geliştirmeye başladılar ve günümüzün kredi kartı şekillenmeye başladı. Ancak kredi kartları geliştikçe farklı ihtiyaçları da beraberinde getirdi. Bu ihtiyaçların en büyüğü nakit işlemlerinin yapılabilmesini sağlayacak **otomatlar**dı. Yani günümüz **ATM**'leri...



Şekil 1.5: Tarihin ilk ATM'si bir çikolata otomatından devşirilmişti ve Barclay's bankası tarafından kullanılmaya başlandı.

İlk ATM atağı, günümüz bankacılığının icat edildiği ve ilk bankaya ev sahipliği yapmış olan **İngiltere**'den geldi. İngiltere'nin önde gelen bankalarından **Barclay's**, kredi kartlarının kullanımını yaymak ve mudilerine rutin mesai saatleri dışında da işlem yapabilme imkânı sunmak adına ATM fikrini benimsedi. Fakat hiç örneği olmayan bir otomatı tasarlamak ve hayata geçirmek çok kolay bir iş değildi. Bu görev **John Sheperd-Barron**'a verildi. John Sheperd-Barron, ATM fikrini tasarlarken hemen herkesin severek kullandığı bir **çikolata otomatı**ndan ilham aldı. Bu sayede kredi kartı kullanıcıları zamanın ilk ATM'sini kullanırken korkmayacak, zaten çikolata ve şekerleme almak için alışık olduğu bir otomatın benzerini kullanacağından kendini rahat hissedecekti.

John, **1967** yılında **Kuzey Londra**'da ilk ATM'yi kurmayı başardı ve kredi kartı kullanılarak ATM üzerinden ilk işlem gerçekleştirilerek ticaret tarihinde bir mihenk taşı daha döşenmiş oldu. Frank McNamara'nın kendinden yola çıkarak bir ihtiyacı keşfedip buna yönelik bir ürün geliştirmesi; Barclay's Bank'ın bu ürün üzerine bir vizyon yaratması ve John Sheperd-Barron'un ATM tasarımının başarılı olması, ticaret dünyasını ve insan hayatını değiştirdi. Kısa sürede ATM kavramı tüm dünyayı sarmayı başardı. ATM'ler hâlâ hayatlarımızın vazgeçilmezlerinden biri olmaya devam ediyor.

Gördüğünüz gibi, yaklaşık 150 bin yıllık bir süreç içerisinde değiş-tokuş esasına dayalı olarak başlatılan ticaret, günümüzdeki hâline gelinceye kadar hep insanın ihtiyaçları doğrultusunda, insanlarla birlikte, sürekli yeni araçlar (*para, kredi kartı, ATM gibi*) geliştirerek evrimleşmeye devam etmiştir.

Elbette günümüz ticaret ve finans dünyasında insan eliyle yaratılmış enstrümanlar para ve kredi kartı ile sınırlı değil. Senet, bono, tahvil, hisse, çek ve tüm bunların işlem gördüğü borsa yapıları ile son yıllarda oldukça popüler olan, **kaldıraçlı finansal yatırım araçları** dur durak bilmeden geliştirilmeye devam etmiş ve etmektedir. Özetlemeye çalıştığım bu tarihsel süreçte, paranın ihtiyaçlarımız doğrultusunda yeni türevlere büründüğünü görüyoruz. Bu türevlerden biri de son yıllarda hayatımıza giren **Bitcoin** oldu. Bitcoin aracılığıyla hayatımıza giren **kripto para yapısı** sayesinde artık finans ve ticaret dünyamız çok farklı bir ufka yelken açabiliyor.

İktisat ve ekonomi perspektifinden bakıldığında, geliştirilen finansal bir uygulamanın ya da enstrümanın insan hayatında birtakım iyileştirmeler de yapabiliyor olması gerekir. Bu pencereden bakınca, "*Kripto para günümüz fiyat parasının evrimleşmiş hâli oldu*" demek pek yeterli kalmıyor. Kripto paranın günlük finansal hayatımızda birtakım "**sorunları**" çözüyor olması ve birtakım "**yenilikler**" ya da "**kolaylıklar**" sunuyor olması gerekiyor. Şimdi gelin, hep beraber kripto para bu perspektifteki ihtiyaçları karşılayabiliyor mu, bakalım.

Günümüz finansal işlemleri ne yazık ki mesai saatleri dışında işlem yapmamıza izin vermiyor. Üstelik hemen her işlem için "**middle man**" ya da Türkçe tabirle "**ortadaki adam**" modeline ihtiyacımız var. Bu modelde banka ya da işlem yaptığımız finans kurumu, yaptığımız işleme onay vererek işlemimizi gerçekleştirmemizi sağlıyor. Tahmin edebileceğiniz gibi bu yapının sınırları ve maliyetleri bulunuyor.

Günümüz finansal dünyasında bir başka sorun da **paranın maliyeti**. Yani paranın basımından dağıtımına kadar süren yolculuğun ciddi bir maliyeti bulunuyor. Biraz durup düşündüğümüzde aynı kâğıt, mürekkep ve baskı maliyetine katlanarak piyasadaki en küçük banknotla en büyük banknotu basabiliyoruz. Bu noktada anladığınız üzere, tedavüldeki en küçük para ile en büyük paranın piyasada dolaşıma girme maliyeti aynı ise devletler haklı olarak alım gücü daha yüksek olan banknotu basmayı tercih edecektir. Bir ekonomi içerisinde alım gücü küçük olan banknotun piyasadan çekilebilmesi için **enflasyon**a ihtiyaç vardır. Tabii ki biz vatandaşlar (*hangi ülkede ya da rejimde olduğunuz fark etmez*) enflasyonun pek de iyi bir şey olmadığını, alım gücümüz üzerinde yarattığı **negatif etki**den biliyoruz. Para özeline bakıldığında işin bir de **güvenlik** ve **takip sorunu** gündeme geliyor. Hemen her yıl dünyanın birçok ülkesinde sahte para operasyonları gerçekleştiriliyor. **Kalpazan** dediğimiz yapılar ve örgütler, sahte para basarak ülke ekonomilerine ciddi darbe vuruyorlar. Bunun önüne geçmek için devletler çeşitli güvenlik önlemleri almak zorunda kalıyor. Bu önlemler de ciddi maliyetlere yol açtığından ülkeler için para basımı çok ciddi bir yük hâline geliyor. Paranın bir noktadan başka bir noktaya fizikî olarak taşınması da ciddi bir başka sorundur. Büyük güvenlik önlemleri eşliğinde özel kargo yöntemleriyle para taşımak, finans kurumlarının ciddi maliyet kalemlerinin başında gelmektedir. Banka ve finans kurumları eliyle devletler, harcanan paraların kaydını tutmayı severler, bu bağlamda ülkeler kendi paralarını takip edebilmek için çeşitli özel yöntemler geliştirmişlerdir. Bu yöntemlerin en bilineni kuşkusuz **SWIFT** yöntemidir. ABD tarafından ABD dolarının (*USD*) uluslararası ve yurtiçi bağlamda hareketlerini takip etmek için geliştirilmiştir. Bu tarz yapılarda aynı bankanın aynı şehirdeki şubeleri arasında bile USD'nizi havale etmek istediğinizde SWIFT kullanmanız gerekir. Bu tarz işlemler, zaman ve para maliyetlidir.

Ancak bu gibi yöntemlerin (*SWIFT vb.*) geliştirilmediği bir gerçeklik hayal edecek olursak bu defa da ülkelerin birbiri üzerinde uyguladıkları çeşitleri ambargo ve yaptırımın kalıcılığı sağlanamaz, illegal oluşumların para trafikleri tespit edilemez ve kayıt dışı ekonominin önü alınamaz. Günümüz fiyat parasının ve sisteminin eksileri elbette bu kadarla sınırlı değil, ancak kripto paranın bu sıraladığım sorunların en azından birkaçını çözmesi bile, benim tezimi doğrulamam için yeterli olacaktır.

Kripto para dediğimiz yapı, tasarımı ve kurgusu açısından **aracıları** (*middle man*) ortadan kaldırarak paranın A kişisinin cüzdanından B kişisinin cüzdanına **hızlıca transfer edilmesi**ni sağlıyor. Middle man olmadığı zaman, işlemi onaylamanın, parayı transfer etmenin maliyeti ciddi anlamda düşürülmüş oluyor (*bazı kripto para modellerinde transfer maliyetleri -**transportation fee**- alınmadığından sıfır maliyetle işlem gerçekleşebiliyor*). Üstelik onay süreçleri kripto paranın bağlı olduğu **Blockchain** yapısında otomatik olarak tamamlandığından, mesai saatleri içerisinde işlem yapan insan bağımını ortadan kaldırmış oluyor. Bu sayede 7 gün 24 saat ve her türlü resmî ya da dinî bayram tatillerinde bile saat farkları gözetilmeksizin A kişisinin cüzdanından B kişisinin cüzdanına para transferi gerçekleştirilebiliyor. Kripto paranın anatomik yapısı itibariyle **100 milyon eşit parça**dan (*genellikle her kripto para için bu en küçük parçalara Satoshi adı veriliyor*) oluşuyor olması ve her bir parçanın harcanabiliyor olması tek tip para mantığını devreye soktuğundan, piyasada birden fazla banknota ihtiyaç olmuyor. Kripto paranın en büyük alametifarikalarından birisi de enflasyon sorununu kökten çözüyor olmasıdır. Kripto paraların hemen hepsinde (*Doge Coin gibi kuralın dışına çıkmış kripto paralar da vardır*) toplam para arzının **sabit** ve **sonlu bir sayı** olarak sınırlandırılmıştır. Bu sayede en nihayetinde bloklar içerisine gizlenmiş tüm ödül paraların **madencilik** esasıyla arz edilmesi tamamlanacak ve tüm arz piyasada dolaşıma girecektir. Toplam arzın **sabit** ve **değiştirilemez olması** esası, enflasyonun oluşumuna mâni olacaktır. Üstelik madencilik diye tabir ettiğimiz blok çözülüp ödül paraların arz edilmesinin maliyeti belli bir periyodik süreçte yarıya düşürüldüğünden, arz kontrollü bir şekilde azalacağı için birim kripto para değeri sürekli artacak ve enflasyonun (*ve tabii ki devalüasyonun*) oluşumuna izin vermeyecektir.

Blockchain mantığı (*ilerleyen bölümlerde detaylı olarak ele alacağız*) her işlemin kaydını tutmak, bu kaydın ağdaki herkes tarafından kontrol edilip onaylanmasını sağlamak ve ağın dışındakiler için bile yapılan işlemlerin açıklık ilkesi tabanında incelenebilmesine olanak sağlamak üzerine kurgulandığından; SWIFT benzeri yapılara, yüksek güvenlikli para basma ve paranın fizikî olarak bir noktadan başka bir noktaya taşınması maliyetlerine katlanılmasına gerek kalmıyor.

Tüm bunlar göz önüne alındığında kripto paranın, paranın evriminde, günümüz fiyat kurlarının tabi olduğu "**itibari para**"dan sonraki halka olduğu kesindir.

Peki, ya Blockchain yapısı? Blockchain yapısı için günümüz muhasebe sisteminin evrimleşmiş hâlidir diyebilir miyiz acaba? Hadi gelin, muhasebenin kısa tarihine beraberce bakalım.

Muhasebe 101 dersinden hatırladığım kadarıyla, finansal karakterli, para ile ifade edilebilen işlem ve olayların kaydedilmesi, sınıflandırılması, özetlenerek raporlar hâlinde sunulması ve yorumlanması ile ilgili bir sistem olarak tanımlanmaktaydı **muhasebe**. Tabii muhasebe sistemi de insan, ticaret ve para tarafında olduğu gibi ihtiyaçlarımızın evrimi sonucunda durmadan gelişerek günümüzdeki hâlini almıştır.

Muhasebeyle ilgili tarihsel sürece baktığımızda ilk uygulama örneklerinin M.Ö. 5000'li yıllara kadar uzandığını görüyoruz. **Babil, Mısır, Eski Yunan, Roma, Eski Avrupa** ve **Orta Çağ**'a ait belgelere baktığımızda, muhasebe tanımımıza uyan çok sayıda kayda rastlayabiliyoruz. Ancak bu kayıtları incelediğimizde, günümüz muhasebesinde kullandığımız sistematik yapılardan ziyade, envanterlerle ilgili listeler, ücret ödemeleri, borç-alacak hesapları, vergi takdirleri ile ilgili kayıtlar ve istihkaklar odağında tutulmuş belgelerden oluştuğunu görüyoruz. Ticaretteki gelişim süreci parayı evrimleştirdikçe, insanların muhasebe kayıt yapılarını ve uygulamalarını bu değişime uydurma gerekliliği doğmuş ve yeni yöntemler geliştirmek amacıyla birçok bilim insanı çalışmalar yapmıştır. Bu çalışmaların en tanınanı kuşkusuz **Luca Pacioli** adlı İtalyan rahip ve matematikçiye ait olanıdır. 1494 yılında Luca Pacioli "**Summa di Aritmetica, Geometria, Proportionni ét Proportionnalite**" adlı eseri yayınlamış ve bu kitabın bir bölümünü muhasebe konularına ayırmıştır. Pacioli bazı kaynaklar tarafından çift taraflı kayıt esaslı muhasebe sisteminin kurucusu kabul edilirken kimi kaynaklarda reddedilir. Detaylı araştırmalar göstermiştir ki Orta Doğu ve Asya ülkelerinde Pacioli'den daha önce çift taraflı kayıt sistemi kullanılan muhasebe belgelerine rastlanmıştır. Ancak Pacioli bu aşamada çok daha iyi bir tanıtıma sahip olduğundan ve kitabında sistematik olarak çift taraflı muhasebe kayıt sistemini anlattığından dolayı muhasebe alanına ismini yazdırmayı başarmıştır diyebilirim. Pacioli ortaya koyduğu muhasebe yapısını sistematiğe dökerken aynı zamanda muhasebecilerin tuttukları defterlerin yetkili makamlar tarafından denetlenip onaylanmasını; şüpheli alacaklar için amortisman ayrılması gerektiği gibi birçok muhasebe esasını da ortaya çıkarmıştır.



Şekil 1.6: Luca Pacioli, günümüz çift taraflı kayıt esasını muhasebe sistemine sokan kişi olarak tarihe geçmiştir.

1540 yıllına geldiğimizde ise "**Quaderno Doppio**" adlı eserin yazarı olan **Domenico Manzoni** hem kuram hem de uygulama açısından çift taraflı kayıt sisteminin ilkelerini tam anlamıyla açıklığa kavuşturarak günümüz muhasebe sistemini ortaya çıkarmıştır.

16. ve 17. yüzyıllara geldiğimizde ise çift taraflı kayıt sisteminin Avrupa tarafından **genel geçer muhasebe sistemi** olarak kabul edildiğini görüyoruz. Bu genişlemenin ardından çift taraflı kayıt esasına dayanan muhasebe sistemine olan ilgi artmış ve katkılar, gelişmelerle birlikte günümüzdeki hâlini almıştır.

Gördüğünüz gibi muhasebenin gelişimi de ticaretin insan hayatındaki yükselişini tamamlamak ve paranın evrimiyle ortaya çıkan yeni finansal düzenleri kayıt altına alabilmek için gelişip günümüzdeki hâlini almıştır. Ancak burada **iki kilit soru**ya odaklanmamız gerekiyor: *Tarih boyunca çeşitli muhasebe yöntemleri o dönemki insanların işini görürken neden çift taraflı kayıt esasına geçilmesi gerekti?*

Ve tutulan muhasebe kayıtlarının otoriter yapı eliyle kontrol edilmesi neden bir muhasebe esası hâlini aldı?

İki sorunun da cevabı oldukça basit: **Güven**. Evet, ticaretin yazılı olmayan kurallarını hatırlayacak olursak **dürüstlük, şeffaflık** ve **güven** ilkelerinin başı çektiğini biliyoruz.

Blockchain ya da dilimize geçtiği şekliyle **blok zinciri yapısı**nda ticaretin olmazsa olmazı bu üç ilke bulunuyor. Blockchain sayesinde bir kripto paranın çıkış yaptığı cüzdandan iki kez harcanamamasını sağlayan dağıtık mantıktaki çalışma prensibi, aynı zamanda bir **akıllı kontrat**ın (*Blok zincir üzerinde oluşturulmuş sözleşmelere kısaca akıllı kontrat ya da akıllı sözleşme diyoruz.*) oluşturulduktan sonra tarafların kendi çıkarları doğrultusunda, birbirinden habersiz değişiklik yapmasını da engelliyor. Üstelik yapılan her işlemin, ister para transferi, ister ödeme, isterse de akıllı kontrat olsun, tarafları dışında, üçüncü kişiler tarafından da kontrol edilip incelenmesine imkân sunuyor. Artık böylesi muazzam bir teknolojik yapılar bütününün neden muhasebenin ve paranın evrimindeki son halka olduğunu daha net anlamışsınızdır sanırım.

SAATLER BLOCKCHAIN'E 1 VAR!

Blockchain için kullanılan en yaygın tanım; kayıt içeren blokların, şifreleme ile birbirlerine bağlanmasıyla oluşan dağınık (*merkezî olmayan*) bir veritabanı olduğudur. Bu tanımda düzeltmemiz gereken en önemli nokta "**bir veritabanı olduğudur**" denilen son kısım. Blockchain için zaman zaman ben de veritabanı sıfatını kullansam da aslen Blockchain'e veritabanı demek çok doğru değildir. Çünkü veritabanlarının amacı veriyi kaydetmek, gerektiğinde silmek, işlemek için çağırabilmek ve değişiklik yapılmış veriyi eskisiyle değiştirmektir. Blockchain'in bize sunduğu güven yapısında verilerin sonradan değiştirilebilmesi ya da silinmesi ne yazık ki bulunmuyor. Bu nedenle Blockchain'i veritabanı olarak adlandıramıyoruz. *Peki, o zaman Blockchain tam olarak nedir?* Bu sorunun cevabını verebilmek için blokları oluştururken kullandığımız *veri, veritabanı, şifreleme, Merkle ağacı, merkezsiz yapı* gibi birtakım kavramlara değinmemiz gerekiyor.

VERİ VE VERİTABANLARININ GELİŞİMİ

Bilişim teknolojilerinin üzerine kurulduğu ve artık günlük yaşamda da dilimizde yerleşmiş olan, İngilizce kökenli "**data**" kelimesinin Türkçe karşılığına **veri** diyoruz. Bilişim sektöründe veri ya da data terimi ile **işlenmemiş, ham bilgi parçacığını** kastederiz. Veri, bir bütünün parçası olmadığı sürece bir anlam ifade etmez. Veriyi bir bütünün parçası hâline getirerek yani işleyip anlamlandırarak kullanmamız gerekir. Günümüzde **büyük veri** (*Big Data*) öbekleri ile çalışan şirketlerin tam da yapmaya çalıştıkları budur. Sadece çok daha büyük veri gruplarını hedeflerler ve işleme ile anlamlandırma konularında yapay zekâdan destek alırlar. Veriler toplandıktan sonra gruplanarak sıralanarak ve özetlenerek elle ya da bilgisayarla işlenip enformasyona dönüştürüldüklerinde anlam kazanırlar. Böylece "**anlamlı veri**"yi farklı amaçlarımız doğrultusunda işleyebiliriz.

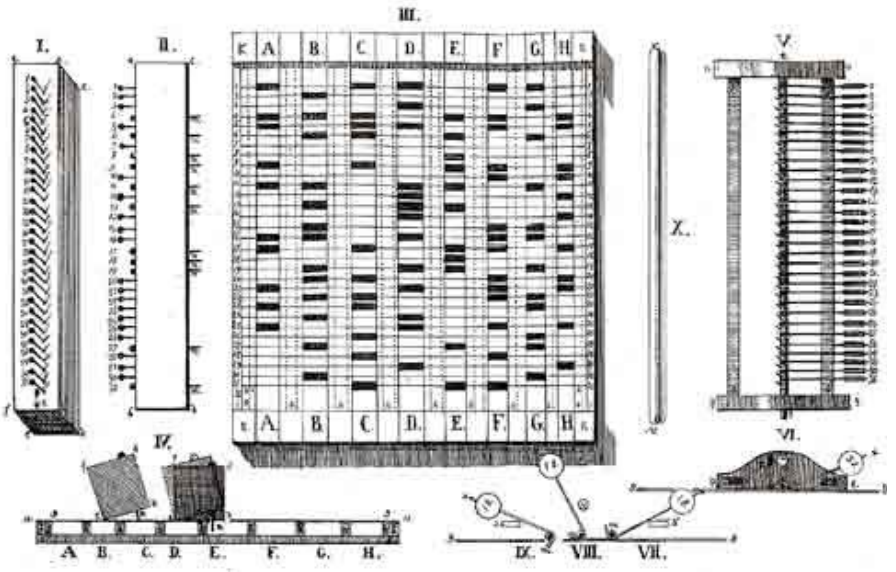
Veri dediğimiz yapı parçaları her zaman kayıtlı olmak zorunda değildir. Kaydı tutulamayan veri tipleri de vardır. Üstelik günlük iletişimimizin hatrı sayılır bir kısmını gerçekleştirdiğimiz konuşma esnasında sarf ettiğimiz **sözcükler**, müzik bestelerinde duyduğumuz melodileri oluşturan **notalar** veya bir deniz fenerinden ışık olarak iletilen **mesajlar**, kaydı tutulmayan verilere örnek olarak verilebilir. Bu tarz veriler, vericisi ile alıcısı arasında transfer edildikten sonra varlıklarını kaybederler.



Şekil 1.7: Tarihin ilk kayıt defteri diyebileceğimiz Ishango kemiği

Bizim şimdilik odaklandığımız veri tipi, **kaydı tutulan veriler**dir. Veriyi (*içeriği ne olursa olsun*) kayıt altına almak insanlık için her zaman önemli olmuştur. Hatta veri kaydıyla ilgili geçmişe baktığımızda insanlık tarihine ait bulunabilmiş en eski kaydın, M.Ö. 12 binli yıllara ait bir maymun kemiğinin üzerine atılan çentikler olduğunu görürüz. Günümüze kadar ulaşmayı başaran bu örnek, Afrika'nın aynı isimli bölgesinde bulunduğu için **Ishango kemiği** olarak isimlendirilmiştir. Ishango kemiği, bugün **Belçika Kraliyet Doğa Bilimleri Enstitüsü**nde koruma altında sergilenmeye devam ediyor.

Yazının keşfedilmesi ile birlikte veri kayıtlarının hızla arttığını ve insanlar arasında aktarılan bilgi miktarının çoğaldığını görüyoruz. Yazı bulunduktan sonra insanlar kaydını tuttukları verileri (*ki o dönemlerde parşömen kullanımı çok yaygındı*) toplu hâlde kütüphane benzeri merkezlerde saklamaya başladılar. Bu tarz yerleri günümüzün **veri merkezleri** olarak düşünebiliriz. Günümüzün modern veri merkezlerinin atası kabul edilebilecek ilk büyük kütüphanenin **M.Ö. 330** yıllarında **Büyük İskender** tarafından kurulduğunu biliyoruz. Bünyesinde 150 bini aşkın ciltten oluşan, toplamda 900 binden fazla el yazması eserin yer aldığı bu kadim kütüphane, maalesef 391 yılında yakılarak yok edilmiştir. Uzun yıllar boyunca el yazmaları şeklinde kaydedilen veriler, 593 yılında Çin'de tahta oyma yöntemi ile geliştirilen ilk matbaa aracılığı ile daha hızlı çoğaltılabilir hâle gelmiştir. Metal harfler kullanan ilk **modern matbaa** ise **1450** yılında Almanya'da **Johannes Gutenberg** tarafından geliştirilmiştir. Bildiğiniz gibi bu aşamadan sonra verinin "**kitap**" formunda çoğaltılması ve dağıtılması hızlanmış, böylece veri dünyasında yepyeni bir çağ başlamıştı. Ancak verinin bilgisayarlar yardımıyla işlenmesi için yaklaşık 400 yıl daha beklememiz gerekti.



Şekil 1.8: Korsakov'un, Jacquard'ın delikli kartlarını, veri depolamak amacıyla homeoskop adlı icadına dönüştürmesi günümüz veritabanlarının önünü açmış oldu.

Verinin insanların kontrolünde makine diline dönüşmesi ise 1801 yılında Fransız bir mucit olan **Joseph Jacquard**'ın karton plakalar üzerinde delikler açması ile gerçekleşti. Jacquard tarafından icat edilen delikli kart metodu 1832 yılında Rusya'da **Semyon Korsakov** tarafından, verinin depolanması ve istendiğinde hızlı bir biçimde bulunması için, "**homeoskop**" adı verilen bir cihaza dönüştürüldü. Delikli kartlar ile verinin işlenmesi, 1950'li yıllara kadar popülerliğini korumuş bir yöntemdir.

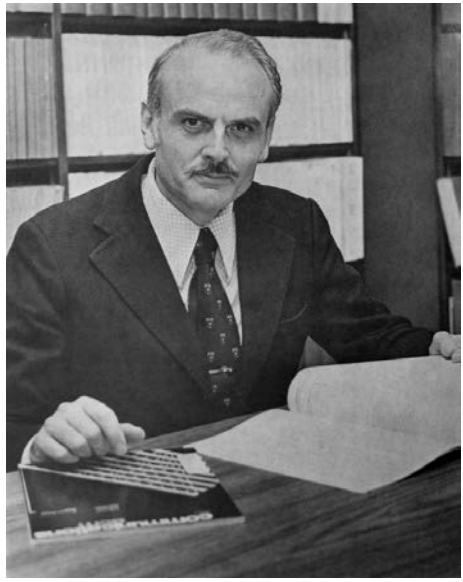
NASA, ilk insanlı uzay uçuşunun temellerini oluşturacak hesaplamaların bir kısmını delikli kartlar ile programlanan bilgisayarlar aracılığıyla gerçekleştirmiştir. Ancak, delikli kartların çok fazla yer kaplaması nedeniyle alternatif yöntemler aranmaya başlanmış ve ilk olarak 1950 yılında, **UNITYPER** isimli ABD merkezli bir şirket tarafından, bilgisayarlar için **manyetik depolama** sistemi geliştirilmiştir. Manyetik depolama sistemleri, uzun yıllar boyunca finans ve bankacılık sektöründe de kullanılan **UNIVAC** bilgisayar sistemlerinin en önemli bileşeni olmuştur. Manyetik şeritlerde veri depolama teknolojisinin 1951 yılında UNIVAC ile birlikte ticari olarak bilgisayar dünyasıyla buluşmasının ardından IBM, 1960 yılında ilk **8 inç** büyüklüğündeki **disket**i (*Floppy Disk*) geliştirmiştir. Bu dönemden sonra çok farklı standartlarda disketler üretilmiştir.

1990'lı yıllarda CD teknolojisinin hayatımıza girmesi ile birlikte manyetik diskler yerlerini **optik depolama üniteleri**ne bırakmış, 1995 yılından itibaren ticari CD kaydedici cihazlar ofis ve evlere girmeye başlamıştır. Bu geçiş döneminde disketlerde 1,44 MB gibi kapasiteler konuşulurken CD'lerle birlikte bu miktar 600 MB'ın üstüne çıkmış ve dijital depolamada devrim niteliğinde bir adım atılmıştır. 90'lı yılların sonuna doğru popüler hâle gelmeye başlayan bir diğer teknoloji ise **Evrensel Seri Yolu** (*USB*) üzerinden bilgisayarlarımıza bağlanan **flash bellekler** olmuştur. 90'ların sonundan 2000'li yılların ortalarına kadar önce disk başına 4,7 GB veri depolayan DVD ve sonrasında çok daha yüksek kapasitelere sahip **Blu-ray optik diskler** hayatımıza girdi. Ancak tüm bu gelişmeler, USB teknolojisinin yeni versiyonlarının çıkması ve kapasitesi sürekli büyüyen flash belleklerin üretilmesi ile birlikte 2010 ve sonrasında artık bilgisayarlardaki optik disk sürücüler bir standart olmaktan çıkmıştı. Günümüzde çoğu dizüstü modelde optik okuyucu bulunmuyor bile. İnternetin hayatımızın vazgeçilmezi hâlini aldığı 2005 yılı ve sonrasında hayatımıza giren bir diğer önemli veri depolama aracı ise **bulut** (*cloud*) kavramı ile internet üzerinden sunulan servisler olmuştur. 2006 yılında **Amazon Web Servisleri** (*AWS*) tüketicilere sunulmuş, ardından küçük büyük birçok şirketin online tabanlı yatırım maliyetleri hatrı sayılır bir şekilde azalmaya başlamıştır. Bu servisleri kullanarak hızla popüler hâle gelen bulut depolama çözümü **Dropbox** ve **benzeri servisler**i ise 2007 yılında hayatımıza girmiştir. 2010 yılından itibaren, hiçbir fiziksel bileşen içermeyen **SSD** (*Solid State Disk*) depolama teknolojisi, bilgisayarlarda önemli bir yer edinmeye başlamıştır. Hem ergonomi hem performans hem de pil ömrünü uzatması sayesinde, taşınabilir bilgisayarların vazgeçilmezi hâline gelmiştir. Günümüzde bulut servisleri, maliyetleri azaltmadaki rolleri başta olmak üzere, yönetimi ve üçüncü parti birçok ürünün hayata geçmesini sağladığından teknoloji evrimimizin bir halkasını başlatmıştır diyebiliriz. Özellikle kripto paraların gelişiminde düşük maliyetli bulut sunucu servislerinin ve sanallaştırma teknolojisinin yeri başkadır. Kitabımızın workshop bölümlerinde Blockchain geliştirmesi yaparken biz de bu tarz yapılardan faydalanacağız.Doğanın bir parçası olarak karşımıza çıkan çeşitli kaynaklardan (*mesela Güneş'ten yayılan ışık demetleri*), İnsan müdahalesi olmadan türeyen verilerden başlayarak insanlığın günümüzün en modern veri depolama sistemlerini geliştirmesine dek geçen tarihsel süreci hızlıca gözden geçirdikten sonra, elimizdeki veriye, bu veri ile üretilmiş bilgiye ve bunları saklayacak ortamlara artık yabancı değiliz.

Bu noktadan sonra, veriyi depolarken kullandığımız **metodoloji** ve **sistem**lere yakından bakacağız. Verinin evrimini anlattığım satırlarda bahsettiğim ve verinin tarihinde dönüm noktası saydığım, delikli kartların bir veritabanı olarak kullanılmasını ele alarak hikâyemize devam ediyoruz.

Delikli kartlar, veriyi depolamak kadar, organize şekilde saklamak için de kullanılan bir yöntemdi. Pek çok farklı delikli kart, üzerlerinde saklı olan verilerin amacına uygun şekilde kullanılabilmesi için, özel olarak kurgulanmış bir saklama düzeni içerisinde, oda dolusu dolaplarda saklanıyordu. Amaçları farklı olsa da günün sonunda veri kümeleri **organize** bir şekilde saklanmış oluyordu ve onlarla çalışan kişiler istedikleri veri grubuna, bu dolaplardaki saklama düzeni sayesinde **kolayca (!)** ulaşabiliyorlardı. 1950'li yıllardan itibaren **analog, manyetik** ve sonrasında **dijital veri saklama yöntemleri** geliştikçe verinin daha düzenli şekilde saklanmasına olan ihtiyaç artmaya başladı. Bu sebeple kökeni yüzlerce yıl öncesine dayanan kütüphanecilik ve arşivcilik teknikleri bilgisayar sistemleri ile buluşmaya ve şekil değiştirerek modernleşmeye başladı. "**Database**" yani Türkçe karşılığı ile "**veritabanı**" kavramı, bilgisayar sistemleri için ilk olarak 1960'lı yıllarda kullanılmaya başlandı. Bu alandaki ilk modern uygulama, aya gitmek üzere tasarlanan **Saturn V** (*Satürn 5*) roketine ait parçaların listelendiği ve kaydedildiği veritabanı sistemi olarak karşımıza çıkıyor. 1970'lere gelene kadar, aslında veritabanı olarak adlandırılan tüm sistemlerin, temelde bugün işletim sistemlerimizle ön tanımlı olarak gelen ve salt metin içerikler oluşturmak için kullandığımız "**not defteri**" uygulamalarına benzediğini görüyoruz. 1973 yılında **IBM San Jose Araştırma Laboratuvarı**nda çalışan **Edgar Codd**, devrimsel sayılabilecek bir kavramı; "**İlişkisel Veritabanı**" tanımını ortaya koymuştur. Bu noktadan sonra bilişim dünyasının ve yazılım geliştiricilerin vazgeçilmezi hâline gelen veritabanlarının temeli atılmış oldu.

Codd, kurum içinde gerçekleştirdiği sunumda bu tanımı yaparken şöyle bir ifade kullanmıştır: "*Gelecekte büyük veri kümeleri ile çalışacak kullanıcıların, makinelerin bu veriyi nasıl sakladığı konusunda bilgi sahibi olmasına gerek olmamalıdır.*"



Şekil 1.9: Edgar Codd ilişkisel veritabanı fikriyle bilişim dünyasını kökünden değiştirecek bir adım atmış oldu.

İlişkisel veritabanları, veriyi tablolarda saklar ve bu tablolar arasındaki bağlantıları oluşturur. Kullanıcılar verilerin nasıl saklandığı ile değil, kendilerine **nasıl sunulduğu** ile ilgilenirler. Veritabanında kayıtlı verilerin sorgulanması için yine Edgar Codd tarafından geliştirilen **SQL** dili kullanılır. (*Evet, günümüzün Oracle DB, MS SQL, MySQL, MariaDB ve PostgreSQL gibi veritabanı çözümleri aslen SQL türevleridir ve kullanım şekilleri küçük farklılıklara rağmen, hemen hemen aynıdır diyebiliriz.*) SQL, 1980'li yıllardan itibaren bir veritabanı sorgulama standardına dönüşmüştür. 1980'li yıllarda kişisel bilgisayarların yaygınlaşmaya başlaması ile hayatımıza giren bir diğer düzenli veri depolama çözümü ise 90'lı yıllarda **Hesap Tablosu** (*aklınıza hemen MS Excel gelebilir*) olarak da isimlendirilen **tablolama uygulamaları** olmuştur. İlk olarak **IBM Lotus 1-2-3** olarak karşımıza çıkan tablolama çözümleri, ilerleyen yıllarda Microsoft'un geliştirdiği **MS Office** ürün ailesi içinde yer alan **Excel** ile birlikte, temel eğitimden en kompleks şirketlerin finans ve muhasebe servislerine kadar her alanda kullanılan bir çözüme dönüşmüştür. Hatta Microsoft bu iş için ofis paketi içerisine **MS Access** adlı bir ürün daha yerleştirdi. Tabii tablolama uygulamalarının sınırlı veri depolama ve analiz çözümleri olduğu gözden kaçmamalıdır.

Fakat yine de birçok geliştirici, bu ürünleri kullanarak birçok farklı iş uygulamasını hayata geçirip sektörde şirketlerin hayatını kolaylaştırmayı başardılar. Artık gelişen teknoloji sayesinde günümüzün devasa veritabanları trilyonlarca satır veri içerebiliyor ve boyutları **Petabyte** (*1 milyon GB'a eşittir*) düzeyinde olabiliyor.

2000'li yıllardan itibaren kullanılmaya başlayan bir diğer veritabanı çözümü ise **NoSQL** olarak tabir edilen çözümler olmuştur. Bu veritabanları, ilişkisel tablolar yerine **sabit yapıda tekil şemalar** içerirler ve bu sayede çok büyük veri kümeleri içinde çok hızlı arama yapmaya imkân tanırlar. Örneğin, *Google, Bing* ve *Yandex* gibi arama motorlarında bu tarz bir veritabanı kullanılır ve bu sayede milyarlarca web sayfası içinde aradığınız bir veya daha fazla kavram için milisaniyeler içinde sonuç almanız mümkün olur. Önceki satırlarda ele aldığımız depolama amaçlı kullanılan bulut servisleri, aynı zamanda veritabanı hizmetleri için de sunulmaktadır. Örneğin **Amazon**, bu konuda kendi ürünleri olan *RDS, DynamoDB, Redshi* gibi farklı çözümler sunmaktadır.

NoSQL bize aynı zamanda Blockchain özelinde de gördüğümüz dağıtık yapılar için de bir kapı açmış oldu. Bugün internete bağlanan cihaz sayısı insan nüfusunu geçti diyebiliriz. **Nesnelerin İnterneti** (*IoT - Internet of Things. Bu yapı, ilerleyen bölümlerde Blockchain türlerini irdelerken yer vereceğim Tangle için temel oluşturur*) adı verilen bu yapı katlanarak büyürken oluşturduğu ekonomi **trilyonlarca dolar** olarak kabul ediliyor. *Bütün bu sistem tarafından üretilen verinin dağıtık şekilde kaydedilmesi ve saklanması mümkün olabilir mi?* Aslında bu sorunun cevabı uzun yıllar önce verilmişti. 2000'li yılların başında birbirinden bağımsız iki proje olarak karşımıza çıkan **eDonkey** ve **BitTorrent**, tam olarak internet üzerinde hiç tanımadığımız ama iletişime geçebileceğimiz diğer kişiler, daha doğrusu makineler üzerinden verinin paylaşılması için geliştirilmiş, uçtan uca "**Peer to Peer - P2P**" olarak isimlendirilen bir tür veri depolama çözümü olarak karşımıza çıktılar. Bu sitemlerde veri, tek bir merkezde değil, sayısı milyonları bulabilecek makineler üzerinde dağıtılmış veya paylaşılmış şekilde bulunur. Bu makinelerden bazıları verinin tamamını, bazıları ise kısmen belli bir parçasını içerebilir. Ulaşmak istediğinizde sistem size bu veriyi, sunulabilecek en hızlı düzenlemeyle farklı makinelerden çekecek şekilde yönlendirir. Siz veriyi kendi bilgisayarınıza çekerken aldığınız veri karşılığında aynı zamanda diğer kullanıcılara bir veri kaynağı olarak hizmet edersiniz.

Sistemin kullanıcıları, sisteme verdikleri **destek süreci** ile doğru orantılı olarak sistemden faydalanabilirler. Bu platformlar, bulutun ötesinde bir çözüm sunmakla birlikte, içeriğin şifrelenmemesi, verinin nerelerde saklanacağına dair seçenek sunmaması gibi nedenlerden dolayı, kurumsal veya mahremiyet içeren kişisel veriler için **güvenli bir depolama çözümü değildir.** Üstelik bilgisayarlarda P2P için bir port sürekli açık tutulurken tercih edilen P2P, uygulamanın paylaşım klasörünü tüm ağın erişimine açtığından kimileri için ciddi bir **güvenlik açığı** olarak tanımlanır. Eksilerine rağmen P2P'nin internete kattığı bu fütüristik yaklaşım sayesinde **Dağıtık Kayıt Defterleri** (*Distributed Ledger Systems*) adı verilen çözümlerle tanışmış olduk. Kitabımızın konusunu oluşturan Blockchain teknolojisi de P2P ile tanıştığımız bu çözümün farklı bir uygulaması olarak karşımıza çıkmıştır diyebilirim.

KRİPTOLOJİ

Veritabanları ve dağıtık kayıt defterinin doğuşu sayesinde temelleri atılan Blockchain teknolojisinin, P2P ile gelen güvenlik sıkıntılarından ve diğer açıklardan korunması gerekiyordu. Öncelikle de verinin üçüncü gözlerden saklanması ve büyük boyutlu veritabanı içerisinde hızlıca aranıp bulunabilmesi için sağlam bir çözüme ihtiyaç vardı. Bu noktada geliştiricilerin aklına **şifreleme yöntemlerini ağa dâhil etmek** geldi. Böylece **kriptoloji** dediğimiz alanla **dağıtık kayıt defteri mantığı** harmanlanmaya başlandı. İşte şimdi sıra, Blockchain teknolojisinin temelinde yatan ve belki de çok büyük bir öneme sahip olan **kriptoloji** kavramına geldi.

Kriptoloji her ne kadar kulağa modern ve yakın zamanda ortaya çıkmış bir kelime gibi gelse de aslında eski bir Yunanca kelime olan **kryptos** kelimesinden türetilmiştir. Kryptos gizlilik anlamına gelirken kriptoloji **gizlilik bilimi** anlamına gelmektedir. Önceki bölümlerde verinin tarihinin ne kadar eskiye dayandığından bahsetmiştik. Bu noktadan da anlayabileceğiniz üzere binlerce yıldır, insanlar üretilen veriyi kayıt altına alırken aynı zamanda istenmeyen gözlerden de saklamak istemiştir ve kriptoloji yöntemlerine başvurmuşlardır. Bu yöntemlerin başında da kriptolojinin bir alt bilim dalı olan **kriptografi** tekniği gelir. Kriptografi yine eski bir Yunanca kelime olan ve yazı anlamına gelen **graphein** sözcüğü kullanılarak türetilmiştir. Kriptografi (*cryptography*) **verilerin *(yazıların)* şifrelenmesi**ni ifade eder. Şifreleme, herhangi bir veri kümesini bir kural yapısı kullanarak rastgele görünen bir veri kümesine dönüştürür.

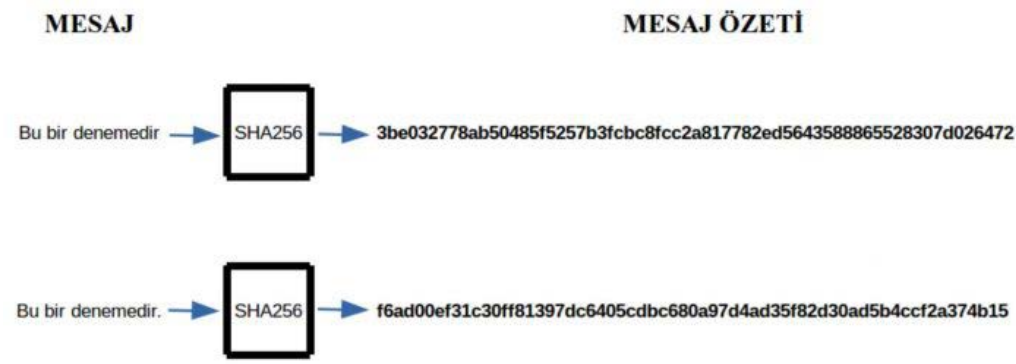
Bu rastgele gibi görünen veri kümesi, ancak şifreleme yapılırken kullanılan anahtara sahip olanlar tarafından **orijinal** ve **anlamlı** hâline geri dönüştürülebilir. Bu anahtara sahip olmayanlar için ise bir anlam ifade etmez. Böylece şifrelenmiş veri nerede ve ne şekilde depolanırsa depolansın, sadece anahtar sahibi tarafından anlamlı kalmaya devam eder. Bu kısımdan anlayabileceğiniz üzere kriptografi yöntemleri, Blockchain üzerinde yapılan işlemlerin değiştirilmesi ve kripto paraların **çift harcanma riski**ni ortadan kaldırmak için kilit rol oynamaktadır.

Kriptolojinin teknik detaylarına ilerleyen bölümlerde değineceğim.

HASH

Konu Blockchain ve kripto paralar olduğunda, kaynakların birçok yerinde "**hash**" ve "**hash fonksiyonu**" kavramlarına değinildiğini fark etmişsinizdir. Uygulamalı olarak workshop bölümünde değineceğim hash kavramının ne olduğunu ve nerelerde ihtiyaç duyduğumuzu kısaca anlatmak istiyorum.

Hash fonksiyonlarını genellikle kriptoloji tabanlı uygulamalarda kullanıyoruz. Projemizde yer alan herhangi bir metnin şifrelenerek okunamaz veya önceden tahmin edilemez hâle getirilmesini istediğimizde başvurduğumuz ya da tasarladığımız algoritma ve dönüştürme işlemlerine kısaca **hash** adını veriyoruz. Hash sistemi daima bir algoritmaya bağlı olarak çalışır ve günümüzde hash'ı kullanmak için **MD2, MD5, SHA** ve **SHA-1** benzeri algoritmalardan faydalanılır. Mesela, Bitcoin, SHA algoritmasını kullanarak hash değerleri üretir ve kendi Blockchain'i üzerindeki işlemleri şifreleyerek güven altına alır. Kendi Blockchain yapınızı kurarken ya da kripto paranızı oluştururken **SHA** algoritması dışında bir algoritma tercih edebilirsiniz. Burada önemli olan bir hash fonksiyonu kullanıyor olmanızdır. Ancak Bitcoin sonrası geliştirilen birçok kripto para projesinde bir kuralmış gibi SHA fonksiyonun kullanıldığını gördük. Hash fonksiyonu seçerken ağın genelinde ne kadarlık işlem gücü ve kaynak ihtiyacı doğuracağını göz önünde bulundurmanız gerekiyor.



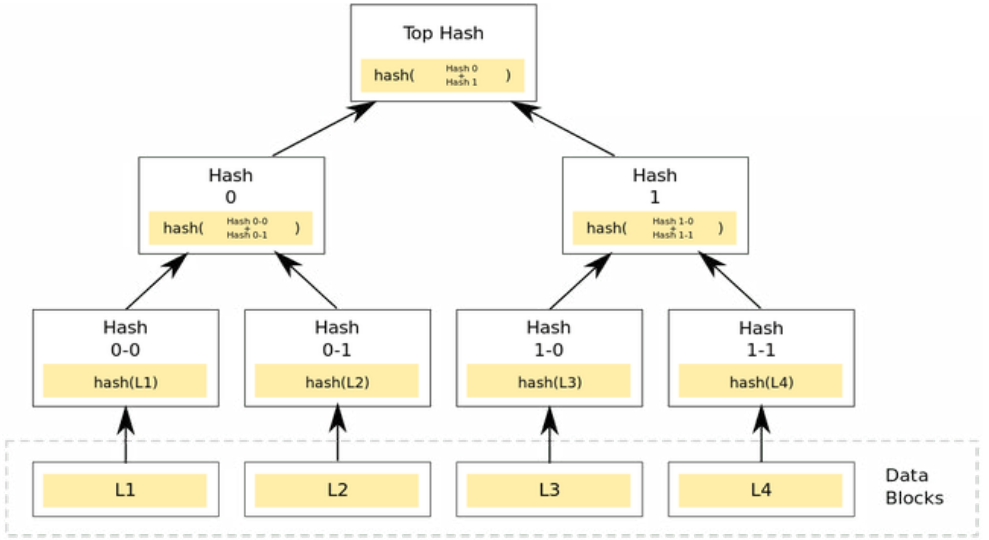
Şekil 1.10: Hash fonksiyonları sayesinde verideki en küçük değişikliği tespit etmemiz çok basit oluyor. Şekilde, cümlenin sonundaki nokta işaretinin hash sonucunu baştan aşağıya değiştirdiğini görüyoruz.

Hash fonksiyonları, genellikle kripto paralardan sonra yazılım sektörü dışındakiler tarafından duyulmaya başladığı için bir başka yanılgı da sadece kripto para ve Blockchain alanında kullanıldığı yönündedir. Ancak hash yapıları yıllardır birçok farklı alanda kullanılıyor. Örneğin, GNU/Linux dağıtımları her **ISO** için (*dağıtımların disk imajları .ISO uzantısı taşır*) bir **MD5** değeri dağıtırlar. Böylece indirdiğiniz dağıtımın, geliştiricilerin elinden çıkmış olan orijinal disk kalıbıyla **aynı olduğunu** ve **değiştirilmediğini** doğrulama şansınız olur. Bu yöntemi, doğrulama gerektiren birçok farklı alanda kolayca kullanabilirsiniz. Veritabanlarıyla yoğun olarak uğraştığım yıllarda verilerin uzunluklarıyla ilgili çok fazla sıkıntı yaşardım. Bu sıkıntıları çözmek için kullandığım yöntemlerden birisi de her veriyi hash fonksiyonu ile sıkıştırıp aynı karakter sayısına getirerek kaydetmekti. Böylece tablolarda ve hücre yapılarında hiç sıkıntı yaşamazdım. Ayrıca **imaj dosyaları** gibi veri tipleriyle çalışırken hem gizliliği sağlamak için hem de boyut sıkıntısı yüzünden veritabanında sorguları yavaşlatmasını engellemek için hash ile sıkıştırıp veritabanına metin olarak yazmak benim için çok pratik olurdu. Anlayacağınız üzere, hash fonksiyonları yazılım dünyasının her alanında oldukça uzun yıllardır kullanılmakta. Blockchain ve kripto paralar ise bu fonksiyonların yoğun olarak kullanıldığı yeni bir ürün sınıfı diyebiliriz.

Hash fonksiyonlarıyla çalışırken bir uygulamanın ötesine geçerek bir ağa yayılacaksak (*mesela P2P ya da Blockchain gibi*) kontrol tarafında işimizi kolaylaştıracak bir yönteme ihtiyaç duyarız. Bu da **Hash Ağacı** (*Hash Tree*) ya da **Merkle Ağacı** (*Merkle Tree*) adıyla anılan yapılardır.

MERKLE AĞACI (MERKLE TREE)

Satoshi Nakamoto Bitcoin mantığını kurgularken nihai hedefi "**P2P Para**" yaratmaktı, demiştik. P2P mantığını önceki bölümlerden hatırlarsanız bilgisayardan bilgisayara aracısız bir şekilde dosya paylaşmamızı sağlayan **dağıtık bir ağ yapısı** sunuyordu. Bu mantıkta para transfer etmek istediğimizde paramızı istediğimiz bir kişi yerine **ağın tamamına** yollamış oluyoruz. Tabii, amacımız bu değil. Satoshi, dağıtık ağ yapısını bozmadan hedeflenen bir cüzdana para göndermek için şifreleme yöntemi kullanarak sadece işlem için üretilen şifre anahtarına sahip cüzdanın parayı almasını sağladı. Bu yapının temelinde hash fonksiyonlarını kullandı. Yazılım dünyası hash fonksiyonlarına Bitcoin'den çok daha önceleri de aşinaydı ve verileri gizlemek için (*özellikle de veritabanlarını*) bu fonksiyonlardan yararlanılıyordu. Ancak Satoshi, Bitcoin'i tasarlarken 1991 yılında **Stuart Haber** ve **W. Scott Stornetta** tarafından ortaya atılan "**Merkle Ağacı**" yöntemini Bitcoin'in temel yapısı içine entegre ederek günümüzün Blockchain tabanlı ilk kripto parasını çifte harcama başta olmak üzere birçok riskten kurtarmış oldu. Blockchain tarafında özerk çalışmalar yapmak gibi planlarınız varsa Merkle yapısını iyi anlamanız gerekiyor. Şimdi gelin bu yapıya yakından bakalım.



Şekilde 1.11: Hash dallarından oluşan tipik bir Merkle ağaç yapısı

Kriptografi ve bilgisayar bilimlerini buluşturan projelerin vazgeçilmezi diyebileceğimiz hash ağacı ya da Merkle ağacında her yaprak düğümü, veri bloğunun özet değerini; her yaprak olmayan düğüm ise kendi alt düğümlerinin kriptografik özet değerlerini içerir. Bu sayede, elimizdeki çok büyük boyutlu verilerin hızlıca organize edilmesi, sıralanması, ilişkilendirilmesi, kontrol edilmesi ve değiştirilmediğinden emin olunarak **güvenli bir şekilde doğrulanması** oldukça kolay bir hâl alır. Bir başka deyişle, Merkle ağacı büyük veri yapılarının verimli ve güvenli bir şekilde doğrulanmasını sağlar. Merkle ağaçları, özet listeleri ve özet zincirlerinin **genelleştirilmiş** hâlidir. Bir yaprak düğümünün, elimizdeki Merkle ağacının bir parçası olduğunu göstermek için, ağaçtaki yaprak düğümü sayısının logaritması kadar birtakım hesaplama yapılması gerekir; bu da yaprak düğümü sayısı kadar işlem gerektiren özet listeleriyle çelişir. Merkle ağacı kavramı, adını **1979** yılında **Ralph Merkle**'nin patentiyle almıştır.

Merkle ağaçları bilgisayarlar arası transfer edilen ya da bilgisayarlarda saklanan, işlenen her türlü verinin doğrulanmasında kullanılabilir. Kişiden kişiye (**P2P**) ağlarda alınan veri bloklarının hasarsız, değiştirilmemiş ve hatta sahte olup olmadığını anlamamıza yardımcı olurlar. *Merkle ağacının bu fonksiyonel özelliği size de Blockchain teknolojisini çağrıştırmadı mı?*

Merkle ağaçları kriptografik hash fonksiyonlarında kullanılır. Merkle ağaçları aynı zamanda **IPFS, Btrfs** ve **ZFS** dosya sistemlerinde (*veri bozulmalarını saymak için*), **Dat** protokolünde, **Apache Wave** protokolünde, **Git** ve **Mercurial** dağıtık versiyon kontrol sistemlerinde, **Tahoe-LAFS** yedek sisteminde, **Zeronet**, sertifika şeffaflığı sisteminde ve *Apache Cassandra, Riak, Dynamo* gibi birçok **NoSQL** sisteminde hâlihazırda kullanılmaktadır. Tabii bir de Bitcoin ve Ethereum P2P ağlarının temelini oluşturur.

Hash, mantık olarak bir veriyi şifreleyerek sıkıştırmış olur. Bu sıkıştırılmış veriyi saklamak, **daha az yer kullanmak** anlamına gelir. Ancak veriye ihtiyaç duyduğumuzda bu fonksiyonu tersine çevirerek veriyi tekrar açarız. Ancak bu tarz işlemler (*verinin boyutuna göre*) oldukça işlem gücü gerektirir ve bu da **zaman** olarak bize geri döner. Bitcoin ve Ethereum ağlarındaki toplam verinin GB'larca büyüklükte veri olduğunu unutmayalım. Bu durumun ileride (*Bitcoin ağındaki kullanıcı sayısı arttığında*) sorun oluşturabileceğini öngören Satoshi Nakamoto, Bitcoin uygulamaya geçirildiği aşamada Merkle ağaçlarını aşırı boyutlu özet fonksiyonlarının **Hızlı Merkle ağaçları** kullanarak sıkıştırma aşamasında hafifletmiştir.

Merkle ağaçları bir dosya veya dosya grubu örneğinde olduğu gibi yaprakları veri bloklarının özeti olan bir **özet ağacı**dır. Ağaçta bulunan düğümler, altlarında yer alan ve kendileriyle ilişkili çocuk düğümlerin özet değerleri alınmış hâlidir. Örneğin, özet değeri 0, altında barındırdığı özet değeri 0-0 ve özet değeri 0-1 olan düğümlerden oluşan veri grubunun, her bir düğümün birleşiminin özet değerini hesaplayıp bir diğer düğüme aktararak Merkle ağacı yapısını oluşturmuş oluruz. Yani, **+** işaretinin birleştirme anlamı taşıdığını varsayarsak; **özet 0 = özet (özet 0-0 + özet 0-1)** ifadesi ile anlatımı matematiksel olarak yazabiliriz.

Merkle ağacı uygulamaları genellikle (*Merkle mantığı gereği*) **ikili** şekilde üretilir (*her düğüm altındaki iki alt düğüm*), ancak her düğümün altında çok daha fazla alt düğüm de kullanılabilir. Genellikle, özet değeri için **SHA-2** gibi bir kriptografik özet fonksiyonu kullanılır. Merkle ağacının yalnızca istem dışı hasarlara karşı korunması gerekiyorsa CRC'ler gibi daha az güvenli **checksum'**lar kullanılabilir. Bir Merkle ağacının tepesinde bir üst özet değeri (*veya kök özet değeri veya ana özet değeri*) bulunur. Bir P2P ağında bir dosya indirmeden önce, çoğu durumda üst özet değeri indirilecek dosyalara iyi tavsiyelerde bulunduğu bilinen bir arkadaş veya bir internet sitesi gibi **güvenilir bir kaynak**tan edinilmiş olur. Üst özet değeri varsa Merkle ağacı, P2P ağındaki herhangi bir eş gibi güvenilir olmayan herhangi bir kaynaktan alınabilir. Daha sonra alınan özet değeri ağacı **güvenilir üst özet değeri** ile karşılaştırılır ve özet değeri ağacı hasar görmüş veya sahte ise program en iyi özet değeri ile eşleşene kadar başka bir kaynaktan başka bir Merkle ağacı deneyecektir. Özet değeri listesinden temel farkı, aynı anda her ağacın bir dalının indirilebilmesi ve anında kontrol edilebilmesidir. Örneğin, yukarıdaki örneğimizde, veri bloğu 2 bütünlüğü, ağaçta özet 0-0 ve özet 1 içeriyorsa veri bloğunun özet işlemi yapılarak sonucu özet 0-0 ve ardından özet 1 ile tekrarlı birleşimiyle hemen doğrulanabilir. Benzer şekilde, ağaçta özet 1-1 ve özet 0 varsa veri bloğu 3'ün bütünlüğü doğrulanabilir. Bu, çok küçük veri bloklarında dosyaları ayıracak kadar verimli olduğu için bir avantaj olabilir. Böylece yalnızca küçük bloklar olması gerekir ve zarar görürlerse yeniden indirilirler. Özet değeri dosyası çok büyükse böyle bir özet ağacı veya özet listesi oldukça büyük olur. Ancak bu bir ağaçsa küçük bir dal hızlı bir şekilde indirilebilir, dalın bütünlüğü kontrol edilebilir ve daha sonra veri bloklarının indirilmesi başlatılabilir.

İKİNCİ PREIMAGE (TERS GÖRÜNTÜ) SALDIRISI

Güvenlik söz konusu olduğunda Merkle ağacı kırılamaz gibi görünebilir. Hele ki ağda kontrolü destekleyen çok fazla kaynak varsa... Ancak Merkle ağacı da teoride birtakım saldırılara açıktır. Mesela, **ikinci preimage** (*ters görüntü*) **saldırısı:** Merkle **özet kökü** (*hash root*), bir saldırganın aynı Merkle özet köküne sahip olan orijinalden farklı bir belge oluşturduğu ikinci **preimage saldırısı** sağlayan ağaç derinliğini göstermez. Yukarıdaki örnekte bir saldırgan, iki veri bloğu içeren yeni bir belge oluşturabilir; ilk özet 0-0 + özet 0-1 ve ikinci özet 1-0 + özet 1-1 şeklinde olacaktır. *Bu gibi bir durumda ağdaki verinin doğruluğu nasıl saptanabilir?* Bu sorunun cevabı, Merkle ağacı içerisindeki **Sertifika Şeffaflığı Sistemi** dediğimiz yapıda saklı. Bu tarz saldırıları öngören geliştiriciler, Sertifika Şeffaflığı Sisteminde basit bir düzeltme tanımlamıştır: Yaprak düğüm özet değerleri hesaplanırken özet verilere otomatik olarak 0x00 baytlık bir ön ek eklenir ve iç düğüm özet değerleri hesaplanırken de benzer şekilde 0x01 baytlık bir ek eklenir. Bu sayede ağda düğüm kontrolleri yapılırken art niyet içeren bir değişiklik hemen tespit edilebilir. Ancak bu gibi önlemler ağaç boyutunda **büyümeye** neden olur. Gerekli durumlarda Merkle ağacının boyutunu kısıtlama yoluna gidilerek bu sorun aşılır. Örneğin Merkle ağacının boyutunu kısıtlamak bazı resmî güvenlik belgelerinde bir ön şarttır ve bazı kanıtları sıkılaştırmaya yardımcı olur. Bazı uygulamalar özet ağaç derinlik ön eklerinin özet değerini alır ve kendinden öncekileri kullanarak ağaç derinliğini sınırlar; böylece ayıklanan özet zincir yalnızca ön ek her adımda azalırsa ve yaprağa ulaşıldığında hâlâ pozitif ise **geçerli** olarak tanımlanır.

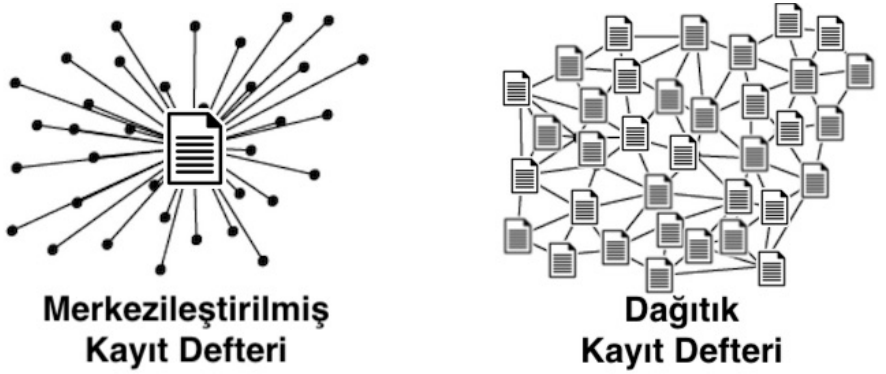
KAPLAN AĞACI ÖZETİ

Kaplan ağaç özeti (*Tiger Tree Hash*) yaygın olarak kullanılan bir özet ağacı şeklidir. Genellikle 1024 bayt veri bloğu boyutları ve ikili özet ağaç (*düğümün altında kendine ait iki düğüm*) yapısıyla kullanılır. Kaplan ağaç özetleri, **Gnutella, Gnutella2 , Direct Connect P2P** dosya paylaşım protokollerinde ve **Phex, BearShare, LimeWire, Shareaza, DC++** ve **Valknut** gibi dosya paylaşım uygulamalarında kullanılmaktadır. Merkle ağacı, kripto para ve Blockchain dünyasının vazgeçilmezi pozisyonundadır. Aslında Blockchain öncesi yakın geçmişe baktığımızda hash tabanlı çalışan harika uygulamalar görüyoruz. Bunlardan en dikkat çekeni kuşkusuz **Shazam**.

Apple tarafından ütopik rakamlara satın alınan uygulama, bildiğiniz üzere, dinlemekte olduğunuz şarkının adı, kimin söylediği ve hangi albümde olduğu gibi bilgileri hızlıca telefon ekranımıza yazabiliyor. Bunu yaparken de veritabanında yer alan şarkıların hash değerlerini, ortamda çalan ve adını merak ettiğiniz şarkıyı hash ile de sıkıştırıp daha sonra o hash değerini veritabanındaki hash ile karşılaştırarak yapıyor. Üstelik tüm bu işlemleri saniyeler içerisinde yapabiliyor.

MERKEZSİZ YAPILAR

Kitabın başında muhasebe ve paranın tarihsel gelişimini kısaca anlatmaya çalışmıştım. Günümüz muhasebe yapısı basitçe **çift kayıt tutma** esasına göre çalışır. Bunun amacı kayıtların değiştirilmediğinden emin olmak için sağlama yapabilmektir. Blockchain tabanlı bir kayıt sisteminde ise veri, ağa bağlı her bir düğüm ile hash kontrolüne tabii tutulduğundan sağlaması **çok daha hızlı** ve **kolay** bir şekilde yapılabilir. Üstelik çift kayıt tabanlı muhasebe sisteminde, kayıtlardan biri veya birkaçı değiştirilmek istendiğinde (*art niyetli bir şekilde*), iki tarafta da yapılacak değişiklik yeterli olacaktır. Artık kaydın orijinal hâline ulaşılamayacağından yeni hâli kabul edilecektir. Hatta tek tarafta yapılacak bir değişiklik bile, sağlamayı kolaylaştıracak üçüncü tip ispat kaynakları olmadığında hangi kaydın doğru olduğu konusunda karar vermemizi imkânsız hâle getirir. Ancak Blockchain gibi ağlarda veri tüm düğümler tarafından kontrol edildiğinden art niyetli bir değişiklik yapmak neredeyse imkânsızdır.



Şekil 1.12: Merkezî ve dağıtık kayıt defteri temsilleri

Merkezsiz yapıların hızlı oluşu, güvenlik algoritması sayesinde her türlü müdahaleye kapalı oluşu, ağ dışındakiler için bile işlemlerin takibini mümkün

kılması (*şeffaflık ilkesi*), muhasebe kayıt esaslarımızın yakın gelecekte **merkezsiz kayıt defteri** esasına dönüşeceğini gösteriyor. Eski ekonominin duayen kaleleri olan bankalar, çoktan dağıtık kayıt sistemleri üzerine çalışmalara başladılar bile. *Peki, nasıl işliyor bu düzenek?*

Basit bir şekilde anlatmak istersek; elimizde bulunan veriyi internet üzerinden (*herhangi başka bir iletişim teknolojisi de olabilir*), önceden kurgulanmış bir ağa yollayarak ağa bağlı kullanıcıların (*bilgisayar, cep telefonu ya da ağa bağlanabilen herhangi bir cihaz, ağımızdaki düğümleri oluşturur*) bu veriyi indirip doğrulayarak kayıt altına almasını sağladığımız yapılara **dağıtık kayıt sistemi** ya da **merkezsiz kayıt sistemi** diyoruz. Bu noktada kayıtlarımız tüm kullanıcılara kopyalanmış oluyor.

Kitabımın önceki bölümlerinde dağıtık kayıt defteri kavramının yeni bir kavram olmadığını ve geçmişte *Kazaa, eDonkey* veya *Bitorrent* gibi **P2P dosya paylaşım ağları** tarafından da kullanıldığından bahsetmiştim (*kripto para dünyasındaki birçok kişi Bitcoin adının Bitorrent adından esinlenildiğini düşünür ve buna inanır*). Ancak bu ağ yapılarının çalışma mantığına yakından baktığımızda küçük bir fark olduğunu görüyoruz. Bu ağlar üzerinde tutulan veriler genellikle şifrelenmeden paylaşıldığından, Blockchain sonrası nihai hâlini alan "**dağıtık kayıt defteri**" mantığından bir hayli farklıydılar diyebilirim. Ancak o dönemlerde bu durum bir eksik ya da hata değildi. Aksine dileyen herkesin verilere erişip paylaşım aşamasında bir düğüm olabilmesi için verinin şifrelenmemesi büyük önem taşıyordu. Bu sayede dileyen herkes ağa dâhil olup bu verilere erişebiliyordu. Bir başka şekilde ifade etmem gerekirse; Bitorrent ağında paylaşılan verileri şifreleyerek (*kriptografi ile*), Blockchain mantığındaki dağıtık kayıt defterlerine aktarmak mümkün olabilir. Ancak bu durumda veriyi şifreleyen kişi/taraf dışında hiç kimse bu veriden fayda göremeyecektir. Üstelik, herhangi bir şekilde ağ noktalarının birisinde veri üzerinde bir değişiklik meydana gelirse verimiz şifrelenmiş olsa bile tutarsız durumlar ortaya çıkabilir. Anlayacağınız üzere, dosya paylaşım ağlarında verinin şifrelenmesi durumu bilerek uygulamaya alınmamıştır.

P2P ağlarında olduğu gibi birden fazla tarafın bulunduğu bir sistemde, sisteme eklenmesi istenen her verinin geçerli bir standarda sahip olması gerekir. Böylece sistemin bütünlüğü korunabilir. Ancak, dağıtık yapılar birbirini tanımayan taraflardan oluşabileceği için, sistemin geneli tarafından kabul edilmiş **kurallara bağlı bir yapı** kurgulamak şarttır.

Bu kurallar, belli bir amaca hizmet edecek sistemlerin tasarım aşamasında belirlenir ve farklı ihtiyaçlara göre değişiklik gösterebilir. Bu kurallar topluluğunun yapısına ve çalışma sürecine, her bir uygulamanın kendisine özel olmak üzere, "**uzlaşma yapısı**" adını veriyoruz.

Belli bir uzlaşma yapısı ile kurgulanan sistemler içerisinde, sisteme bağlı tüm paydaşların uyduğu kuralların işleyiş yapısına da "**uzlaşma süreci**" adını veriyoruz. Sınırlı sayıda birey bir araya geldiğinde el sıkışarak sözlü veya yazılı bir kayıt ile bir uzlaşma sağlayabilirler. Dijital bir sistem üzerinde uzlaşma yapısının sağlanması için, bunun yazılım kodları kullanılarak garanti altına alınması gerekir. **Satoshi Nakamoto**, Blockchain teknolojisini ortaya koyarken tüm bu sorunları çözeceği bir model kurgulamayı başarıyor. Baştan uzlaşma yapısı (*kuralları*) belirlenmiş şekilde veriyi kaydetmemizi mümkün kılıyor. Üstelik, daha sonra bu kayıtları iletişim ağları üzerinden, pek çok noktaya dağıtarak verinin tüm noktalarda aynı kaldığına dair güvenilir bir uzlaşma süreci başlatıyor. Hatta bununla da kalmıyor ve tüm kullanıcıların verilerini şifreleyeceği bir çözüm de sunmayı başarıyor. Bu aşamada **Blockchain teknolojisi**, dijital dünyada artık kolaylıkla oluşturabildiğimiz, güncellediğimiz ve hatta silebildiğimiz veri kullanım şekline farklı bir bakış açısı getiriyor; dağıtık bir veri kayıt sistemi şeklinde karşımıza çıkarak kaydedilen bir verinin sonsuza kadar değiştirilemez olduğunu garanti eden güvenilir bir yapı olduğunu gösteriyor.

BLOCKCHAIN NEDİR?

Buraya kadar olan bölümlerde Blockchain yapısının kurgulanması, hayata geçirilmesi ve kullanım alanlarıyla ilgili evrimsel sürece değindik. Üstelik, Blockchain tabanlı uygulamalar geliştirirken ihtiyaç duyacağımız kavram ve bilgilere de sahibiz. Artık Blockchain yapısını ameliyat masasına yatırıp daha teknik olarak incelemeye başlayabiliriz.

Blockchain yapısı, birçok bilgisayardaki işlemleri kaydetmek için kullanılan, merkezî olmayan, dağıtık ve ağa dâhil olan ya da olmayan herkesin inceleyebileceği **açık bir dijital defter**dir. Böylece, yapılan işlemden sonraki tüm blokların değişmesi ve ağın anlaşılması dışında kayıt geriye dönük olarak değiştirilemez bir hâl alır. Ayrıca bir denetleme ve doğrulama işleminin gerçekleştirilmesi için, ağdaki kullanıcılara izin ve ödül verme imkânı sağlar.

Bir blok veritabanı, bir P2P ağı ve dağıtılmış zaman damgası sunucusu kullanılarak bağımsız olarak yönetilir. Ağdaki hemen herkesin desteklediği, kitlesel işbirliği ile blokların değiştirilmediği doğrulanarak onaylanır. Sonuç, katılımcıların veri güvenliği konusundaki belirsizliğinin marjinal olduğu **sağlam ve sürekli bir iş akışı** olarak karşımıza çıkar. Blockchain ağına dâhil olan dijital bir varlık, artık değiştirilebilirlik, sonsuz yeniden üretilebilirlik ve silinebilirlik özelliklerini tamamen kaybeder. Her bir birim değerin ağda sadece bir kez aktarıldığından, uzun süredir devam eden **çift harcama problemi** çözülmüş olur. Blockchain, Nakamoto tarafından aynı zamanda bir değer değişimi protokolü olarak da tarif edilmiştir. Bu **Blockchain bazlı değer değişim sistemi**, geleneksel sistemlerden daha hızlı, daha güvenli ve daha ucuz bir şekilde tamamlanabiliyor. Üstelik Blockchain, mülkiyet haklarını atayabiliyor. Çünkü teklif ve kabulü zorlayan bir kayıt yapısına sahip.

BLOK

Blockchain, adından da anlaşıldığı üzere bloklardan oluşan bir yapı. Bloklar veriyi içerir ve veriyle ilişkiye giren her işlemi kaydedip kontrol ederek onayını veren kullanıcılar arasında **iletişimi yönetir**. Bloklar, bir Merkle ağacına hash'lanmış ve kodlanmış geçerli işlemlerin yığınlarını tutar. Her blok, blok zincirindeki önceki bloğun **kriptografik hash fonksiyonu**nu içerir ve ikisini birleştirir. Böylece birbirine bağlanan bloklar bir zincir oluşturur. Bu süreç ağda sürekli bir döngü içerisinde devam ederken önceki bloğun bütünlüğünü, orijinal başlangıç bloğuna (*Genesis bloğu*) geri döndüğünü doğrular.

Bazı durumlarda eş zamanlı olarak ayrı bloklar üretilebilir ve ağ üzerinde **geçici bir çatal** oluşturulabilir. (*Kripto para dünyasında bu işlemlere **fork** deniyor. Ancak fork işlemleri genellikle kalıcı olarak ilgili Blockchain ağına uygulanır.*) Güvenli bir hash temelli tarihe ek olarak herhangi bir blok zinciri, tarihin farklı sürümlerini puanlamak için belirli bir algoritmaya sahiptir, böylece daha yüksek bir değere sahip olan biri diğerlerine göre seçilebilir. Zincire dâhil edilmek üzere seçilmeyen bloklara, **yetim bloklar** denir. Veritabanını destekleyen eşler, zaman zaman tarihin farklı versiyonlarına sahiptir. Sadece kendileri tarafından bilinen veritabanının en yüksek puanlı sürümünü tutarlar. Bir eş, daha yüksek puanlı bir versiyon aldığında (*genellikle tek bir yeni blok eklenmiş olan eski versiyondan bahsediyoruz*) kendi veritabanlarını genişletir veya üzerine yazar ve geliştirmeyi eşlerine tekrar iletirler. Herhangi bir girişin, tarihin en iyi versiyonunda sonsuza kadar kalacağıyla ilgili kesin bir garanti asla yoktur.

Çünkü Blockchain tipik olarak yeni blokların reytingini eski bloklara eklemek için inşa edilir ve yeni blokları eski blokların üzerine yazmak yerine zincire yeni bloklar ekleyerek yapıyı genişletmeyi tercih eder.

BLOK SÜRESİ

Blok süresi, Blockchain tasarlanırken belirlediğimiz bir periyottur. Ağın, blok zincirinde bir ilave blok üretmesi için gereken süredir. Blok süresini belirlemek bizim elimizdedir. Ancak blokların işlemleri ve veriyi içerdiğini unutmamak gerekir. Blok üretme süresini ne kadar uzun tutarsak blok içerisine o kadar fazla veri alabiliriz ve bloğa girmek için sıra bekleyen işlemler kuyruğundan kurtuluruz. Ancak blok süresini kısa tutacak olursak işlem hızımız artacak ve eğer bir para birimiysek arzımız oldukça hızlanacaktır. Yalnız bu noktada yanlış anlaşılmaya mahal vermemek gerekiyor. Blok süresini aynı zamanda blok boyutuyla **doğru orantılı** olarak belirlememiz gerekiyor. Örneğin blok boyutunu 1024 KB olarak belirlediğimizde blok içerisine ancak 1024 KB büyüklüğünde veri koyabiliriz. Eğer blok süresini 5 dakika olarak belirlersek bloğumuzun kapasitesi dolduğunda yeni blok üretmemiz gerekir. Yoksa işlemler kuyruk hâline gelir ve ağımızda noktadan noktaya hareket etmek bir hayli güçleşir. Eğer blok süresini kısa tutmayı hedefliyorsanız blok boyutunu da ona göre ayarlamanız gerekir. Blok tamamlandığı zaman, içerilen veriler **doğrulanabilir** hâle gelir.

Kripto para birimlerine bakacak olursak ödeme işlemi gerçekleştiği zaman bu hızlı bir şekilde gerçekleşmelidir. Dolayısıyla daha kısa blok süresi daha hızlı işlem anlamına gelir. **Ethereum** için blok süresi 14 ila 15 saniye arasında belirlenmişken **Litecoin** için 2.5 dakika ve **Bitcoin** için 10 dakika olarak ayarlanmıştır. Anlayacağınız üzere en yavaş ağ Bitcoin ağıdır.

Yalnız bu noktada, açıklamam gereken bir başka faktör devreye giriyor. O da ağdaki işlemi onaylamak için gereken **reyting *(blok puanı)* sayısı**. Yani bir işlem gerçekleştiğinde bu işlemin doğruluğu onaylanana kadar bloğun reyting alması beklenir (*onay*), bu da ağın hızı için önemlidir. Örneğin, A cüzdanı B cüzdanına 0.05 BTC yollamak istediğinde, işlemi gerçekleştirir ve bu işlem sıradaki bloğun içerisine girmek için beklemeye başlar. Ağın yapısı gereği büyük boyutlu ödemeler her zaman küçük boyutlu ödemelerden önce işleme alınır. Eğer sıradaki bloğa girmeyi başaramazsa bir sonraki bloğu bekler. Bu da kabaca **20 dakika** gibi bir zaman alır (*hatırlarsanız Bitcoin'in blok üretme zamanı 10 dakikadır*).

Bloğun içerisine girdiğini varsayarsak da en az 3 onay alması gerekir ki bu ağdaki kullanıcılardan en az üçünün bu yeni bloğu indirip kontrol ederek Genesis bloğa onay göndermesi anlamına gelir. Uzun lafın kısası işlemimizin gerçekleşmesi bu saydığım adımlara bağlı olmak şartıyla 30 dakika ile 60 dakika arasında bir süreye ihtiyaç duyar. Ancak kurumuz **BTC** (*Bitcoin'in kur kısaltması BTC'dir*) yerine **ETH** olsaydı (*Ethereum'un kur kısaltması ETH'dir*) işlem 1–2 dakika içerisinde tamamlanabilirdi.

HARD FORK (AĞI ÇATALLAMAK)



Şekil 1.13: Çatallanmalar Bitcoin'in değerinde negatif etkilere sahip olmakla birlikte; borsalarda parasını katlamaya çalışan "al-sat"çılara daha çok emtia sağlayarak piyasaları canlandırıyor.

Hard fork tabiri Blockchain ve kripto para dünyasında kullanılan önemli terimlerden biridir. Tam anlamı, Blockchain'i kurgularken kullanılan kurallarda değişikliğe gidilmesidir. Değişiklik sonrası Blockchain artık yeni kurallar doğrultusunda blok üretmeye başlayacağı için, eski kurallara göre onaylanan yazılımlar, yeni kurallara göre üretilen blokları geçersiz olarak göreceklerdir. Hard fork durumunda, yeni kurallara uygun olarak çalışacak tüm düğümlerin yazılımlarını yükseltmeleri gerekmektedir. Buradaki "**yazılım**"dan kastettiğim, "**core**" diye tabir ettiğimiz ve Blockchain ağına erişmemizi sağlayan **cüzdan uygulamaları**dır.

Bir düğüm grubu eski yazılımı kullanmaya devam ederken diğer düğümler yeni yazılımı kullanıyorsa **bölünme** gerçekleşebilir. Örneğin, Ethereum, kodundaki bir güvenlik açığından istifade ederek saldırıya uğramış olan **The DAO**'daki (*Decentralized Autonomous Organization*) yatırımcıları "**bütünleştirmek**" için hard fork yaptı. Bu durumda, **çatallanma** işlemi, bölünmüş bir Ethereum ve Ethereum Classic zincirleri ile sonuçlandı.

2014 yılında, **NXT** (*açık kaynak kodlu, Bitcoin benzeri, anonim bir kripto para ve ödeme sistemi projesi*) topluluğunun, 50 milyon NXT'lik bir hırsızlık olayının büyük bir kripto para değişimi işleminin etkilerini hafifletmek için blok zincir kayıtlarının geri dönüşüne yol açacak hard fork'u düşünmesi istendi. Hard fork teklifi reddedildi ve fonların bazıları müzakereler ve fidye ödemeleri sonrasında geri alındı.

Alternatif olarak kalıcı bir bölünmeyi önlemek için, yeni yazılımı kullanan düğümlerin çoğu, 12 Mart 2013'te Bitcoin bölünmesi durumunda olduğu gibi eski kurallara geri dönebilir. Bir ağ çatallandıktan sonra eski sürüm yazılım kullanan düğümlerin blok üretimi devam edecek olursa eski ağ devam ederken yeni ağ tamamen farklı kurallarla blok üreterek yoluna devam eder. Bitcoin çatallandığında **Bitcoin Cash, Bitcoin Diamond, Bitcoin Gold** gibi birçok çatallanmış zincir hayatımıza girdi. Ancak zaman içerisinde orijinal Bitcoin ve Bitcoin Cash dışında kalan ağlar düğümlerini hızla kaybetmeye başlayarak piyasada erimeye başladılar.

MERKEZSİZLEŞTİRME

Blockchain teknolojisi, verileri **P2P** (*peer-to-peer*) ağ üzerinden depolayarak verilerin merkezî olarak tutulması durumunda ortaya çıkabilecek bir dizi riski ortadan kaldırmış olur. Merkezî olmayan Blockchain teknolojisi, geçici mesaj iletimi ve dağıtılmış ağ iletişimi protokollerini kullanabilir.

P2P Blockchain ağları, bilgisayar korsanlarının yararlanabileceği merkezî güvenlik açığı bulundurmaz. Aynı şekilde, merkezî bir arıza noktası yoktur. Blockchain kodlarında saklı olan güvenlik yöntemleri, **açık anahtar şifrelemesi** kullanımını içerir. **Açık anahtar** (*uzun, rastgele görünen bir sayı dizisidir*) Blockchain ağı üzerinde bir adrestir. Ağ üzerinden gönderilen değer belirteçleri bu adrese ait olarak kaydedilir. Ağdaki her düğüm bu anahtara sahiptir. Bu şekilde ağdaki her hareket takip edilebilir. Gizli anahtar ise sahibinin dijital varlıklarına erişim sağlayan bir parola gibidir. Bir başka deyişle blok zincirinin şu anda desteklediği çeşitli yeteneklerle etkileşimde bulunma araçlarıdır. Blockchain ağında yer alan bloklar içerisinde saklanan veriler, genel olarak **bozulmaz** olarak nitelendirilir. Dolayısıyla gizli anahtarı unutmanız ya da kaybetmeniz durumunda geri getirme şansınız yoktur. Web sitelerinden alışageldiğimiz şekilde, "**parolamı unuttum**" seçeneği Blockchain dünyasında yer almıyor ne yazık ki.

Merkezî veriler daha kolay kontrol edilirken bilgi ve veri manipülasyonu mümkündür. Erişilebilir bir defterdeki verileri merkezsizleştirerek ve herkese açık olan blok düzeyindeki şeffaf veriler formuna sokarak ağda olmayan kişiler tarafından bile incelenebilir hâle getirilir.

Merkezî olmayan bir sistemdeki her düğüm, Blockchain ağındaki tüm blok verisinin bir kopyasına sahiptir. Veri kalitesi ve güvenliği, Blockchain tasarlanırken kodlanmış olan hash algoritması ile sağlanan kontrol ve doğrulama yapısı sayesinde sürdürülür. Blockchain mantığında hiçbir şekilde "**orijinal**" ya da "**resmî**" kopya yoktur ve hiçbir kullanıcı diğerlerinden daha fazla "**güvenilir**" değildir. İşlemler, yazılım kullanılarak ağa yayınlanır. Mesajlar en iyi çaba esasına göre iletilir. Madencilik düğümleri işlemleri kontrol ederek doğrular (*madencilik kavramına ilerleyen bölümlerde değineceğim*) ve oluşturdukları yeni bloğa ekledikten sonra, tamamlanmış bloğu diğer düğümlere yayınlar. Blok zincirleri, değişikliklerin serileştirilmesi için **proof-of-work** (*yapılan işin kanıtı – ki burada işten kastımız bloğu çözüp içerisindeki ödül kripto parayı kazanabilmek için yapılan hesaplamadır*) gibi çeşitli zaman damgası şemaları kullanır. Alternatif konsensüs yöntemleri **proof-of-stake** içerir.

AÇIKLIK

"**Açık blockchain ağı**" mantığı, kamuya açık olsa da hâlâ görünmesi için fiziksel erişim gerektiren geleneksel Blockchain yapısına göre daha kullanıcı dostudur. Tüm erken Blockchain ağ yapıları "**izinsiz erişim**" mantığı içerdiğinden, blok zinciri tanımlaması üzerinde zaman zaman tartışmalar ortaya çıkmıştır. Bu süregelen tartışmadaki en önemli konu, merkezî bir otorite tarafından görevlendirilen ve yetkilendirilen (*onaylanan*) doğrulayıcılara sahip özel bir sistemin, bir Blockchain ağı olarak kabul edilip edilmemesi gerektiğidir.

İzin verilen veya özel Blockchain yapılarının savunucuları, "**Blockchain**" teriminin, verileri zaman damgalı bloklara bölen herhangi bir ağ yapısına uygulanabileceğini iddia etmekteler. Bu bloklar, veritabanlarında **MVCC**'nin (*Multiversion Concurrency Control*) dağıtılmış bir versiyonu olarak hizmet eder. **MVCC**, bir veritabanındaki tek bir nesneyi aynı anda değiştiren iki işlemi önlediğinden, blok zincirleri, iki işlemin bir blok zincirinde aynı tek çıktıyı harcamalarını önler.

Muhalifler, izin verilen sistemlerin geleneksel kurumsal veritabanlarına benzediğini, merkezî olmayan veri doğrulamasını desteklemediğini ve bu tür sistemlerin operatörün onaysız değişiklik yapmasına ve revizyonuna karşı zorlaştırmadığını söylüyor.

Şimdi, bu son kısmın biraz karmaşık geldiğini ve aslında neyi anlatmaya çalıştığımı anlamadığınızı tahmin edebiliyorum. Fakat bu son paragraftan da anlayabileceğiniz üzere, Blockchain yapılarının **izin alarak** bağlanabildiğimiz ve **izin almadan** bağlanabildiğimiz türleri mevcut. İzin almaya ihtiyaç duymadığımız yapılar için kripto para ağlarını düşünebilirsiniz. İzin almamız gereken yapılar özel yapılardır ve her düğüm işlemleri kontrol etme ve onaylama yetkisine sahip olmayabilir. Gelin bu yapılara yakından bakalım.

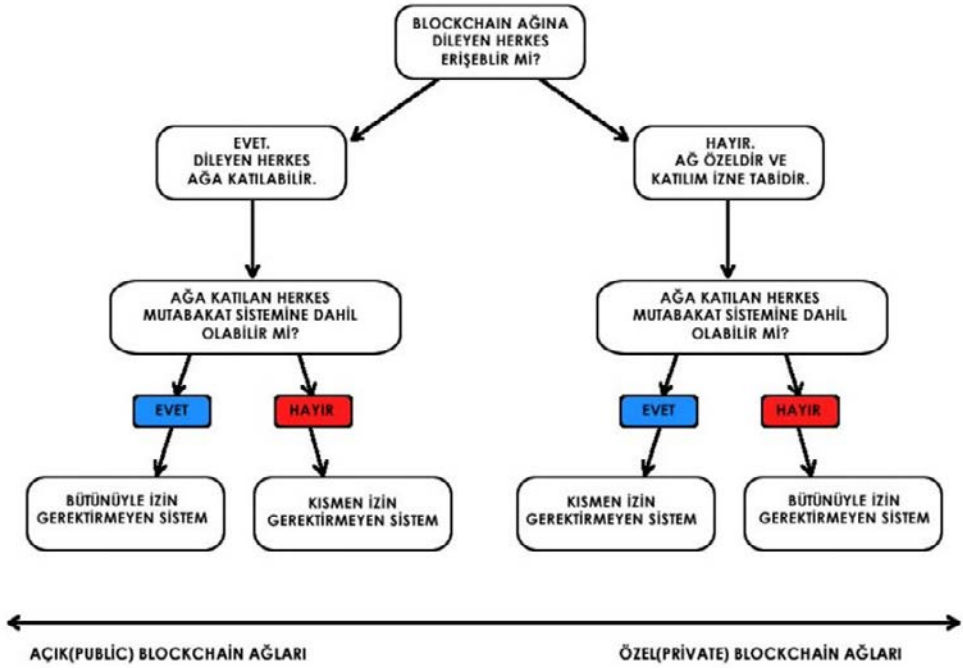
İZİN GEREKTİRMEYEN BLOCKCHAIN

Açık, izinsiz ya da **halka açık** bir Blockchain ağının en büyük avantajı, kötü oyunculara karşı korunmanın gerekli olmaması ve erişim kontrolüne ihtiyaç duyulmamasıdır. Bu mantık, uygulamaların başkalarına onaya veya güvene gerek kalmadan ağa eklenebildiği ve blok zincirinin bir taşıma katmanı olarak kullanılabileceği anlamına gelir.

Bitcoin ve diğer kripto para birimleri hâlihazırda bir **proof of work** (*iş kanıtı*) içerecek yeni girişler gerektirerek blok ağını güvence altına alır. Blockchain ağının blok sayısını artırmak, yani zinciri uzatmak için Nakamoto, 1990'larda **Adam Back** tarafından geliştirilen **Hashcash bulmacaları**nı kullanır.

Blockchain teknolojisi en çok ekonomi ve finans alanlarında fayda sağlıyor demiştim (*ve demeye devam edeceğim*); fakat ne yazık ki finans şirketleri, merkezî olmayan Blockchain ağ yapılarına öncelik vermiyorlar. 2016 yılında ABD'de Blockchain ile ilgili projeler için risk sermayesi yatırımı zayıflamakta, ancak Çin'de ilginç şekilde artmaktaydı. Şimdilerde ise finansal kuruluşlar Blockchain teknolojisini altyapılarında ya da mevcut finansal ağların yerine kullanmaya çalışmak yerine, kripto paraları yatırım araçları hâline getirmeye çabalıyorlar. Bu girişim sayesinde 2017 Nisanı'ndan sonra Bitcoin 20 bin USD değerine kadar yükselmiş ancak sonrasında 3400 USD seviyesine kadar durmadan gerilemiştir. İzin gerektirmeyen ağlara en iyi örnek olarak Bitcoin ve temel diğer birçok kripto para birimi verilebilir.

İZİN GEREKTİREN (ÖZEL) BLOCKCHAIN



Şekil 1.14: Açık ve Kapalı Blockchain yapılarının temsili

İzin gerektiren Blockchain ağ yapıları, gerekli izin ya da onayı almayan kullanıcıların erişimine kapalı olan yapılardır. Ayrıca ağa erişimi olan kişileri yönetmek için bir erişim kontrol katmanı kullanırlar. **Açık *(izinsiz)* Blockchain ağları**nın aksine, özel blok zinciri ağlarındaki doğrulayıcılar ağ sahibi tarafından kontrol edilir. İşlemleri doğrulamak için anonim düğümlere güvenilmez ve ağ etkisinden yararlanılmaz. İzin gerektiren Blockchain ağ yapıları, ayrıca "**konsorsiyum**" veya "**hibrit**" blok zincirleri adıyla da anılırlar.

New York Times, 2016 ve 2017 yıllarında pek çok şirketin Blockchain ağlarını "**kamu sisteminden bağımsız özel blok zincirleriyle**" kullandığını belirtmiştir.

Buraya kadar, "*Blockchain nedir? Nasıl çalışır? Hangi anatomik parçalardan oluşur? Erişim bakımından türleri nedir?*" gibi sorulara yanıt bulduk. *Ancak Blockchain ağ yapılarını geliştirmemize sebep olan "**eureka**" aşaması neydi?*

Aslında buradaki "**eureka**"mız büyük ekonomik buhranlardan birisiydi diyebilirim. Gelin o günlere dönerek neden Blockchain yapısına ihtiyacımız oldu, ona bakalım.

Blockchain'i Doğuran Sebepler

Kitabımın girişinde, muhasebe ve paranın kısa tarihini anlattığım bölümde de bahsettiğim gibi insanoğlu tarihi boyunca veri kayıtlarını tutarken gelişmekte olan toplumları içerisinde düzenin sağlanmasını amaçlamıştır. Büyümekte olan toplumlarda, birbirini tanımayan kişi ve yapılar arasındaki ilişkilerin kurallara bağlı hâle getirilmesi ve bu kuralların kayıtlara geçirilmesi her zaman büyük önem taşımıştır. Temel olarak günümüze dek kurulan pek çok organizasyon, dernek, vakıf, şirket ve hatta devletler bu kayıtları oluşturmak, düzenlemek ve işletmek için yine topluluklar ve toplumlar tarafından kurulmuştur. 2008 yılında, global krizin patlak vermesinden hemen önce, ABD ekonomisinin ve dünyanın en büyük finans kurumlarından birisi olan **Lehman Brothers**'in iflası dengeleri yerinden sarsmıştı. Üstelik bu, gölün durgun sularına atılan taşın çıkarttığı ilk dalgaydı. Bu iflasın ardından önce yıkılmaz diye tabir edilen bankalar, ardından çok uluslu şirketler ve sonrasında da ülkeler ciddi bir ekonomik krizin avcuna düştü. İşinden olan binlerce kişi yeni bir iş bulamadığından kredilerini ve ipoteklerini ödeyemez hâle gelerek aileleriyle birlikte dipsiz bir kuyunun içerisine düşüverdiler. **Mortgage krizi** de denilen bu süreç, insanların bankalara karşı tavır almasına neden oldu. Bu resimde her ne kadar bankalar insanları ve şirketleri batışa sürüklemiş gibi görünse de aslında onlar da batmamak için mudilerini ve kredi dağıttıkları şirketleri dibe çekmeye başlamışlardı. Bu zorlu sürecin ilk aşaması olan Lehman Brothers'in iflasını izleyen süreçte (*aslen sadece iki ay sonra*) ortaya çıkan ve daha önce de belirttiğim gibi kimliği hâlâ gizliliğini koruyan **Satoshi Nakamoto** takma adlı kişi veya grubun yayınladığı "**Bitcoin: Eşten Eşe Elektronik Nakit Ödeme Sistemi**" başlıklı teknik çalışma, her ne kadar bir dizi matematiksel ve teknolojik uygulamayı bizlere sunsa da aslında makalede verilen temel mesaj gayet insani ve basitti: **aracıların olmadığı bir ekonomi yapısı kurmak!**

Bu durum aslında son kullanıcıya zaman kısıtlaması olmadan, 7 gün/24 saat, uçtan uca para gönderme imkânı sunan bir yapı olmakla birlikte; kişi ve/veya kurumlara **kendi kaynaklarını üretme** imkânı sunuyordu. Daha da iyisi bu yeni sistem sayesinde, devletler de en büyük maliyet kalemi olan para basma olayından kurtulabilirlerdi.

Bir düşünsenize; özel kâğıt, özel mürekkep, filigran ve üç boyutlu güvenlik önemleri, baskı maliyetleri, basılan paranın kontrolü ve fizikî güvenlik önlemleri eşliğinde basılan paranın piyasaya arz edileceği banka şube ve ATM'lerine ulaştırılması... Bu sürecin sıkıntılı bir başka yanı da 1 USD basmak ile 5 USD hatta 20, 50 ve 100 USD basmak aynı maliyete sahip. *Bu noktada devlet olarak 1 USD basmak mı isterdiniz yoksa 100 USD mi?* Elbette cevabınız 100 USD olabilir. Ama bu kolay bir süreç değildir. Hele ki mevcut ekonominiz içerisinde 1 USD hâlâ alım gücünü kaybetmemişse bu kararı kolay kolay alamazsınız. Yine de 1 USD basmak yerine 100 USD basmak istiyorsanız enflasyon yaratmanın yollarını bulmanız gerekir. Bildiğiniz üzere enflasyon bir ülkedeki çalışanların alım gücünü eriten güçlü bir etken olduğundan pek sevilmez. İşte bu tip senaryoların düğümlediği ekonomik çıkmazları aşmanın en kolay ve maliyetsiz yolu **Blockchain teknolojisi** olarak karşımıza çıkıyor. Yani, Blockchain teknolojisini doğuran sebeplerin başında enflasyon, global ekonomik krizler ve bunların sonucunda doğan mali kaynaklara erişim sıkıntıları, para basma maliyetleri, paranın güvenliğini sağlama ve dağıtma maliyetleri geliyor. İkincil sebep ise kuşkusuz bankacılık sisteminde artan komisyon oranları, para transferlerindeki maliyet artışları ile finansal dünyada "**ortadaki adam**" (*middle man*) diye tabir ettiğimiz aracı kurumlardan kaynaklı üçüncü parti maliyetler diyebiliriz. Blockchain teknolojisi ve kripto para yapıları bu sıkıntıların tamamını ortadan kaldırmayı vaat ediyor. Bitcoin modelinde gördüğümüz **sınırlandırılmış toplam arz miktarı** (*Bitcoin modelinde toplam 21 milyon BTC bulunur ve bu rakamın tamamı arz edilerek dolaşıma katıldığında arza ek yapılamaz*) devalüasyon ve enflasyon risklerinin oluşumunu engeller.

Artık Blockchain teknolojisi neden doğdu ve bu teknolojiye neden ihtiyacımız var anlayabiliyoruz. Ancak Blockchain tarafında kuramsal bazı tartışmalar da mevcut. Özellikle Blockchain teknolojilerine büyük yatırımlar yapan ülkelerin konu kripto paralara geldiğinde çok agresif bir tavır sergileyerek yasaklama yoluna gittiklerini görebiliyoruz. Hatta ülkelerin bu teknolojiye bakış açısını üç gruba ayırabiliriz: **pozitif yaklaşanlar, nötr duranlar** ve **negatif yaklaşanlar**. Hâlâ eski nesil ekonomik yapılar ve modellere sıkı sıkıya bağlı kalan ülkelerin bu konuda negatif bir duruş sergileyerek regülasyonlar bağlamında yasaklama ya da engel koyma yoluna gittiğini görüyoruz. Bu ülkelerin birçoğunun eski ve köklü finans kartellerine sahip olmasının yanı sıra petrol, altın ve Keynesçi emtiaların borsalarını yönetiyor olması sizleri şaşırtmaz herhâlde.

Nötr duruşa sahip olan ülkeler ise ticaret yaptıkları müttefik ülkelerin kararlarını desteklemek ya da tepkilerini çekmemek adına çekimser kalmayı tercih ediyorlar. Ancak üreten ve özellikle de teknolojiye yön vermeyi hedefleyerek gelecekte oyun kurucu olma hedefi kurgulayan ülkeler tam teşekküllü yatırımlarla altyapılarını Blockchain ve kripto para teknolojilerine geçirmeye başladılar bile. Merkez bankası karşılıklarını Bitcoin'e endekslemeyi tercih eden ülkelerin sayısı her geçen gün artıyor. Fakat basını takip eden okurlar hemen hatırlayacaktır ki bazı ülkeler (*mesela Hindistan ve Çin*) Blockchain teknolojisine büyük yatırımlar yaparken **kripto paraları** ve **ICO *(Initial Coin Offering)* süreçlerini** sert bir şekilde yasaklama yoluna gidiyor. Bu noktada akıllarda şu soru uyanıyor: *Acaba Blockchain teknolojisi kripto parasız olur mu? Ya da Blockchain altyapısı olmadan kripto para kurgulanabilir mi?* Hadi bu sorunun cevabını bulalım.

Blockchain Kripto Parasız Olur Mu?

Kitabımın önceki bölümlerinde Blockchain teknolojisinin geliştirilme amacının; daha kusursuz, daha güvenli ve **ortadaki adam** ihtiyacını ortadan kaldırabilmek olduğunu söylemiştim. Bitcoin ve sonrasında hayatımıza giren diğer kripto para projelerinden görebildiğimiz kadarıyla da bu misyonunu başarıyla yerine getirebiliyor. Ancak Blockchain altyapısının olmadığı bir kripto para kurgulamak istediğimizde **çift harcama sorunu** hemen karşımıza çıkıyor. Bunu engellemek için A cüzdanıyla B cüzdanı arasına bir kontrol ya da emanet otoritesi konumlandırmamız gerekiyor ki o zaman günümüz fiyat kurları ve bankacılık sisteminden farklı bir şey yapmamış oluyoruz. Ayrıca blok mantığından çıkıp muhasebeleştirme anlamında kayıtları hâlihazırda yaptığımız gibi çift kayıt esasında tutmamız gerekiyor ki bu da zaten güven konusunda kayıtları tutan kişi ve/veya kurumların insafına kaldığımız anlamını taşıyor. İşte, bu perspektiften baktığımızda Blockchain teknolojisi olmadan kripto para kurgulayamayacağımızı anlıyoruz. Yani, **Blockchain olmadan kripto para olamaz** diyebiliriz.

Peki, perspektifi değiştirelim ve kripto para olmadan Blockchain olabilir mi, ona bakalım. Blockchain yapısını oluşturan bloklar ve bloklara giren işlemleri onaylayan düğümlerin mantığını önceki bölümlerde anlatmıştım.

Şimdi şu noktaya odaklanalım: Bir bilgisayarınız var ve çalışmak için elektrik, internet bağlantısı, artı sizin eforunuza ihtiyacı var. Bu üç ögeyi harcayarak bağlı olduğunuz ağdaki bir para transferini onaylıyorsunuz. Parayı gönderen ve alan cüzdanların sahipleri, işlemleri gerçekleştiği için mutlular. *Peki, işlemin onay sürecini ve maliyetini karşılayan siz mutlu musunuz? Yani sarf ettiklerinizin yerini dolduracak ne var elinizde? Hiç mi? O zaman gönüllülük esasıyla bu işlemi kaç kez daha yapardınız?* Bu sorunun cevabını biliyoruz sanırım. Fakat insanlar bu tarz işlemleri karşılıksız yapmadığında Blockchain teknolojisi çaresiz kalmıyor. Ağın kontrol, doğrulama ve onaylama yükünü ağı kurgulayanların **düğüm sunucu**larına atabiliyoruz. Blockchain teknolojisine ilham veren **P2P dosya paylaşım ağları**nın yapısını hatırlayın. Dosya direkt olarak kullanıcının bilgisayarından diğer kullanıcının bilgisayarına paylaşılıyordu. Ancak tüm kullanıcılar **offline** ise ağda daima **online** olan sunucular dosyaları paylaşmaya devam ediyordu. Bu mantık, Blockchain içerisinde kurgulanabilir. Ağdaki düğümlerin blok onaylama işlemini yapmadıkları zaman bu işi üstlenecek düğüm sunucuları devreye alınabilir. Ancak bu bizim güven esasımızı sarsar ve ağda açık oluşmasına neden olur. *Nasıl mı?* Basit! Ağın kurucularının sahip olduğu ve yönetebildiği düğümlerden bahsettiğimizden, aslında onaylanacak veriye önce müdahale ederek değiştirip sonra da veriyi onaylama imkânları doğar. Bu da verinin değiştirilemez olduğu kuralı çiğnemekle birlikte, ağın merkezsizliğini bozar. Ağımız artık merkezî bir otoritenin kontrolünde diyebiliriz. Yani, teknik olarak elimizde dağıtık yapıda ama merkezî şekilde yönetilen bir Blockchain türevi olmuş oluyor. Ayrıca **kripto para** dediğimiz **hash değerleri** ağımız içerisinde bloktan bloğa hareket etmemizi de sağlıyor. Bu noktada Blockchain ağımıza yeni bloklar ekleyebilmemiz için ağda hareket eden kripto paraya ihtiyacımız var. Bu noktada aksini savunabilecek meslektaşlarım olabileceğini de belirterek söylemek isterim ki kripto parasız bir Blockchain ağı kurgulamamız maalesef mümkün değildir.

Öyleyse regülasyonlar aracılığıyla kripto paraya, tabiri caizse savaş açan ülkeler için bir çıkar yol var mı? Aslında bu sorunun cevabı ayrı bir kitap konusu diyebilirim. Çünkü, **para politikası** dediğimiz yapı ve içerisinde barındırdığı para tanımı değiştirilmeden bu sorun net bir şekilde çözülemez. Ancak Blockchain üzerinde hareket eden yapıya **kripto para** demek zorunda değiliz. *Jeton, kontör, dijital senet* vb. isimler vererek para sıfatını kullanmayabiliriz. *Dijital değer deposu, dijital finansal aset* gibi kavramlar da uydurabiliriz.

Sonuçta bu yapıya ne dediğimizin bir önemi olmamakla birlikte Blockchain'den ayıramayacağımız bir gerçek. O yüzden içerisinde kripto paranın olmadığı bir Blockchain ağı tasarlamamız ve hayata geçirmemiz pek mümkün değildir. Böylelikle sektöre girmeyi hedefleyen hemen her kesimden kişinin aklını kurcalayan sorunun cevabını vermiş olduk.

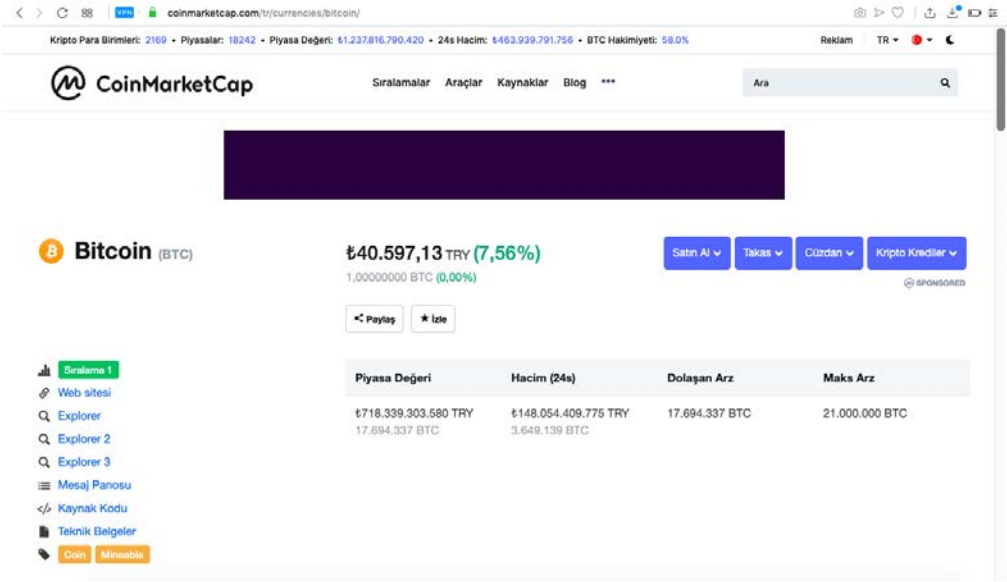
Buraya kadar olan bölümleri okuyan okurlarımızın kafasında Blockchain ağı, teknik ve kuramsal olarak tek tipmiş gibi bir algı oluşmuş olabilir. Ancak bir programlama dili ya da tek tip bir matematiksel fonksiyondan bahsetmediğimizi hemen belirteyim. Blockchain dediğimiz yapı aslında bir kayıt yöntemi ve bu yöntemin tasarıma dökülmesi aşamasında farklı amaçlara yönelik özelleşmiş yorumları mevcut. Bu yorumlar bize temelde aynı amaca hizmet eden farklı teknik altyapılara ve geliştirme süreçlerine sahip mimariler olarak dönüyor. Şimdi gelin, Blockchain teknolojisinde kaç farklı mimari var, yakından bakalım.

BLOCKCHAIN MİMARİLERİ

Blockchain ağ mimarilerinin ilki, bu teknolojiyle tanışmamızı sağlayan **Bitcoin ağı**dır. Bitcoin'i temel alan hemen tüm kripto para ağları blok zamanı, toplam para arzı miktarı, onay adetleri ve hash algoritmasındaki değişiklikler dışında Bitcoin ağından farksızdır. O yüzden Bitcoin ağını oluşturan Blockchain mimarisini (*çatalları ayrı tutmadan*) **öncü mimari** olarak tanımlıyorum. İkinci ağımız akıllı kontrat mantığını kurgudan çıkartıp hayata geçirmemizi sağlayan **Ethereum ağı**dır. Ethereum da gün yüzüne çıkıp kendini ispatladıktan sonra *EOS, NEO, TRON* gibi türevleri sektöre girdi ancak bunların da Bitcoin türevlerinden farkı olmadığından **ikinci özgün mimari** olarak Ethereum ağını ele alacağız. Üçüncü mimarimiz ise IOTA ile tanıştığımız **TANGLE** ağıdır. Birçok meslektaşım TANGLE ağını **gerçek merkezsiz Blockchain ağı** olarak tanımlamaktadır. Son yıllarda Blockchain gibi gündemi meşgul etmeyi başaran bir diğer çığır açan teknoloji de **IoT** olarak tanımladığımız **Internet of Things** ya da Türkçeye geçmiş hâliyle **Nesnelerin İnterneti** kavramı. Bu yapıyı taban alan bir Blockchain ağ sistemi olarak karşımıza çıkan TANGLE, gelecekte çok konuşulacak diyebilirim.

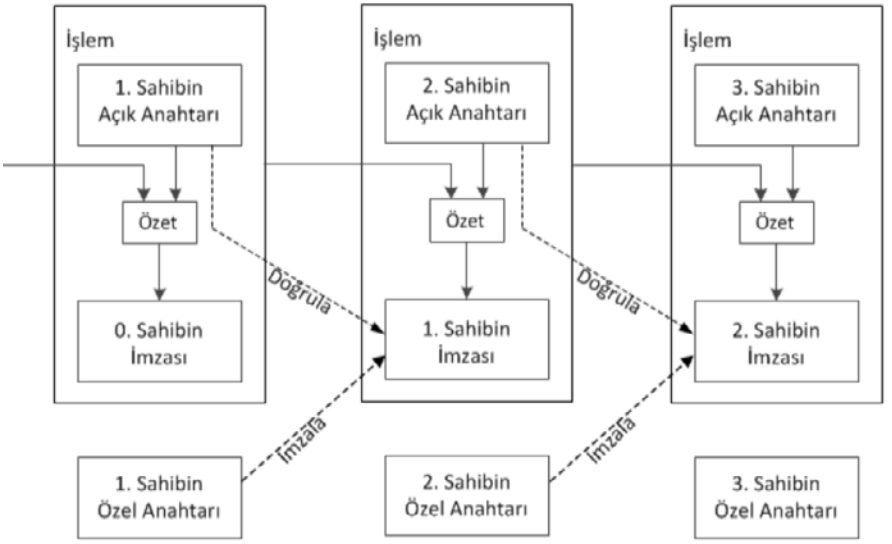
Şimdi Bitcoin ağıyla başlayalım.

BITCOIN AĞI (BLOCKCHAIN 1.0)



Şekil 1.15: Bitcoin anonim bir yapı olmasaydı, sahibi olan şirket ya da ülkeye 718 milyar TL'den fazla kaynak yaratmış olurdu.

Bitcoin'in Blockchain ağında, kripto para bir **dijital imza zinciri** olarak tanımlanır. Paranın el değiştirmesi sırasında her sahip, parayı bir sonrakine gönderirken kendi dijital imzasıyla **bir önceki işlemin özetini** (*hash*) ve **bir sonraki sahibin açık anahtarını** imzalar. Bu imzayı da ödemeye konu olan paranın sonuna ekler. Ödeme alan cüzdan ise sahiplik zincirini ispatlamak için imzaları kontrol eder ve doğrular.

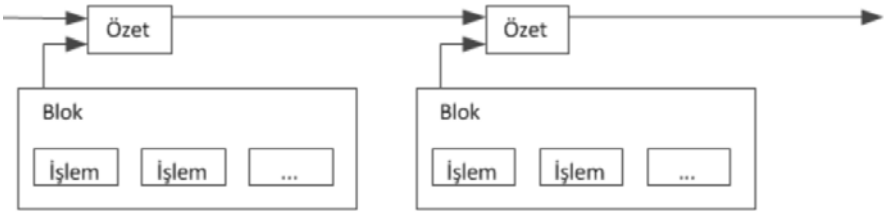


Şekil 1.16: Bitcoin ağında açık anahtarların kontrolü ve doğrulama şeması

Ağda sadece en eski işlem önem taşır. Yani daha sonraki harcama girişimlerini dikkate almıyoruz. Zaten Bitcoin ağının temel prensibinde işlemin gerçekleşmediğini kanıtlamanın tek yolu, tüm işlemlerden haberdar olmaktan geçiyor. **Merkeze dayalı model**de, merkez tüm işlemleri bildiği için hangisinin önce geldiğine karar verebilir. Güvenilen bir taraf olmadan bunu başarabilmek için işlemler ağda açıkça ilan edilmelidir ve katılımcıların işlemlerin gerçekleşme sırası konusunda hemfikir olacağı bir yapı kurgulanmalıdır. Ödeme alanın, her işlem sırasında, harcamanın ilk kez yapıldığı taraf olduğunun diğer düğümlerin çoğu tarafından onaylandığı bir kanıta ihtiyacı vardır. İşte Bitcoin ağının temelde yaptığı iş aslen budur. Bunun için de **zaman damgası** prensibi kullanılır.

ZAMAN DAMGASI SUNUCUSU

Satoshi'nin bu noktada kurguladığı yapı, bir zaman damgası sunucusuyla başlıyor. Bir zaman damgası sunucusu, damgalanmayı bekleyen bir işlem bloğunun özetini alarak bu özeti ağda yayınlar. Zaman damgası, mesaj yayınladığı anda verinin özete girdiğinin, yani var olduğunun kanıtıdır. Her zaman damgası, özetinin içinde bir önceki zaman damgasını barındırarak bir zincir oluşturur ve her eklenti öncekileri güçlendirir.

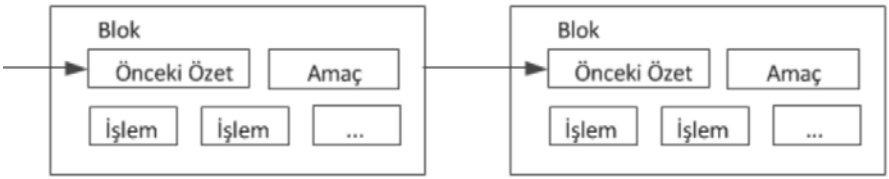


Şekil 1.17: Zaman Damgası özet yapısı. Her özet kendinden önceki zaman damgası özetini bünyesinde bulundurur.

İŞ KANITI (PROOF OF WORK)

Eşten-eşe dağıtık bir zaman damgası sunucusu uygulaması için Adam Back'ın **Hashcash** sistemine benzer bir iş-kanıtı sistemine ihtiyacımız var. İş-kanıtının temelinde SHA-256 özet fonksiyonu ve özet çıktısının belli sayıda 0 biti ile başlamasını sağlayacak bir amaç değerin aranması yatar. Ortalama gerekli iş yükü özette amaçlanan ve tek bir özet hesaplamasıyla sınanabilen 0 bitlerinin sayısı ile **üssel olarak orantılı**dır.

Zaman damgası ağımız için iş kanıtı modelini, blok özeti, istenen sayıda 0 biti ile başlayıncaya kadar bloğun içindeki bir amaç değerinin değiştirilmesi üzerine inşa ediyoruz. İş kanıtını elde edecek **CPU** tüketimi gerçekleştikten sonra, blok aynı işi tekrar yapmadan değiştirilemez. Zincire sonradan eklenen bloklar öncekilere bağlı olduklarından bir bloğu değiştirmek kendinden sonra gelen tüm blokları da tekrar hesaplamayı gerektirir ki bu, çok sayıda düğümün olduğu ağlarda imkânsıza yakındır.



Şekil 1.18: İş-kanıtı (Proof-of-Work)

İş-kanıtı aynı zamanda çoğunluk kararının temsil edilmesi problemini de çözer. Eğer çoğunluk kararı IP adreslerine dayalı bir yöntemle hesaplanıyor olsaydı birçok IP adresine sahip olan kişiler tarafından suistimal edilebilirdi. İş-kanıtı yönteminde **1 CPU, 1 oy**a eşittir. Çoğunluk kararı, en fazla iş kanıtının yapıldığı en uzun zincir tarafından temsil edilir.

CPU işlem gücünün çoğunluğu, dürüst düğümlerin elindeyse dürüst zincir diğerlerinden daha uzun olacak ve rakip zincirleri alt edecektir. Geçmiş bloklardan birini değiştirmek için saldırganın sonrasında gelen tüm bloklardaki işi tekrar yapması ve dürüst düğümlerin yaptığı işi yakalayıp geçmesi gereklidir. Tabii ağa yeni bloklar eklendikçe, daha yavaş bir saldırganın bunlara yetişme olasılığının üssel olarak azaldığı aşikârdır. Artan donanım hızlarına ve değişken aktif düğüm sayısına ayak uydurmak üzere iş-kanıtı zorluk seviyesi saatte ortalama blok hedefini tutturacak şekilde tekrar ayarlanır. Eğer çok hızlı blok üretiliyorsa zorluk seviyesi yükselir.

AĞIN İŞLETME ADIMLARI

Bitcoin ağının işler hâle geçtiği an itibariyle, blok oluşturup onay süreci sonrası zincire ekleme prensibi belli başlı aşamalardan oluşur. Bu aşamaları şu şekilde sıralayabiliriz:

1) Yeni işlemler tüm düğümlere yayılır.

2) Her bir düğüm yeni işlemleri bir blok içinde toplar.

3) Her bir düğüm kendi bloğu içinde bir iş-kanıtı bulmak için çalışır.

4) İş-kanıtını bulan düğüm, bunu tüm diğer düğümlere yayar.

5) Diğer düğümler blok içindeki tüm işlemler geçerliyse ve daha önceden harcanmadıysa bloğu kabul ederler.

6) Düğümler bir sonraki bloğu çözmek için çalışmaya koyulduklarında önceki bloğun özetini de yeni bloğa dâhil ederler, böylece bloğu kabul ettiklerini göstermiş olurlar.

Düğümler her zaman en uzun zincirin doğru olduğunu kabul ederek uzatmaya çalışırlar. Eğer iki düğüm aynı sıradaki bloğun farklı versiyonlarını **aynı anda** bulup yayınlarlarsa bazı düğümler birini diğerinden önce alabilir. Bu durumda düğümler ilk aldıkları bloğu **doğru** kabul ederler ve üzerinde çalışırlar, ama diğer dalı da sonradan daha fazla uzama ihtimaline karşılık saklarlar. Beraberlik bir sonraki iş-kanıtı bulunduğunda ve dallardan birisi daha uzun hâle geldiğinde bozulmuş olacak, diğer dal üzerinde çalışan düğümler de uzun tarafa geçeceklerdir.

Yeni işlemlerin tüm düğümlere ulaşması gerekmez. Yeteri kadar düğüme ulaştıklarında çok geçmeden bir blokta yer alırlar. Blok yayınları mesaj kayıplarına karşı da dayanıklıdır.

Eğer bir düğüm bir bloğu alamadıysa bir sonrakini aldığında kayıp olanı fark eder ve istekte bulunur.

TEŞVİK

Kural gereği bir bloktaki ilk işlem bu bloğu yaratanın sahip olacağı yeni bir parayı dolaşıma çıkardığı özel bir işlemdir. Bu, düğümlerin ağı desteklemelerini teşvik eder. Aynı zamanda, para basacak merkezî bir kurum olmadığından paranın dolaşıma katılmasını da sağlar. Sabit bir miktardaki para miktarının sürekli dolaşıma girmesini, altın madencilerinin piyasaya yeni altın sürmesine benzetebiliriz. Bizim durumumuzda tüketilen şey **CPU işlem gücü** ve **elektrik**tir.

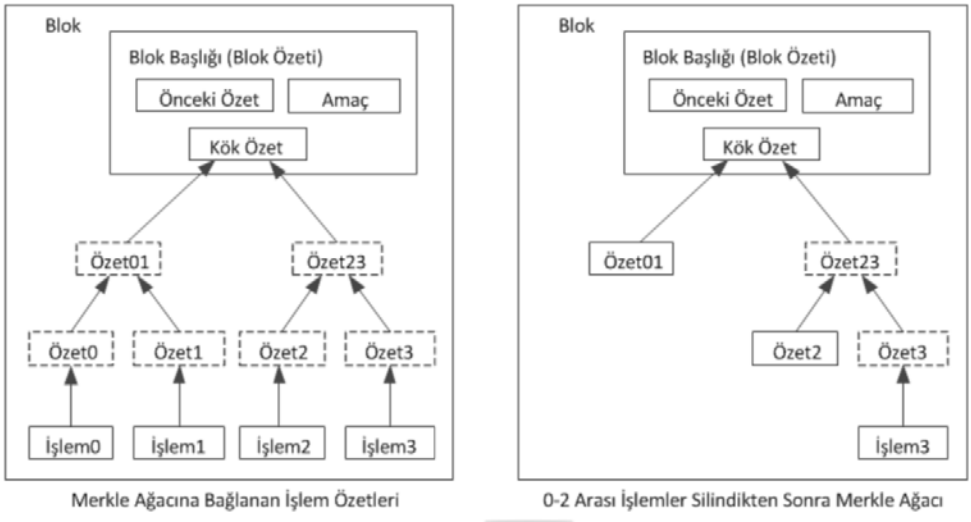
Teşvik işlem ücretlerinden de elde edilebilir. Bir işlemin çıktısı girdilerinden küçükse aradaki fark **işlem ücreti** olarak işlemi ihtiva eden bloğun teşvik miktarına eklenir. Önceden belirlenmiş **toplam arz miktarı**nı (*Bitcoin için bu değer 21 Milyon adettir*) kapsayan paranın tamamı tedavüle çıktıktan sonra, teşvik, işlem ücretinden elde edilir ve enflasyon sıfıra iner.

Teşvik yapısı aynı zamanda ağdaki düğümlerin dürüst kalmalarını sağlar. Eğer art niyetli bir bilgisayar korsanı bütün dürüst düğümlerden daha fazla işlem gücünü toplamayı başarabilirse bu gücü ne için kullanacağına karar vermesi gerekir. İlk seçeneği tabii ki ağdaki tüm ödemeleri çalarak insanları dolandırmak olabilir. Ancak bunu yapmak yerine yeni para üretmeyi de tercih edebilir. İlk seçeneği tercih ederse kısa süre içerisinde fark edileceğinden dolayı itibarını ve belki de tüm kazancını kaybetme riskine sahiptir. Düşünsenize, Bitcoin gibi bir yapının kırılabileceğinin ispatlanması değerini bir anda sıfıra yakın bir noktaya çekebilir. Bu durum bilgisayar korsanının istemeyeceği bir sonuçtur. İkinci yolu tercih etmesi durumunda ağ çalışmaya devam edecek ve sürekli yeni para arzı olacağından korsanımız daha çok kazanma şansına sahiptir. Bir başka deyişle, oyunu kurallarına uygun oynamanın daha kârlı bir yol olduğunu anlaması gerekir.

DİSK ALANI

Bir paranın kullanıldığı son işlem, yeteri kadar bloğun altında kaldığında, öncesindeki harcama işlemleri saklama alanından tasarruf etmek üzere silinebilirler. Bloğun özetini bozmadan bunu sağlamak için işlemler bir Merkle ağacında tutulur ve sadece kökü bloğun özetine dâhil edilir. Eski bloklar ağaç dalları budanarak sıkıştırılabilir. Ara özetlerin saklanmasına gerek yoktur. Bu mantık bize ağda dolaşan özet verinin boyutunu aşırı büyük değerlere gelmemesi için kontrol imkânı sunar.

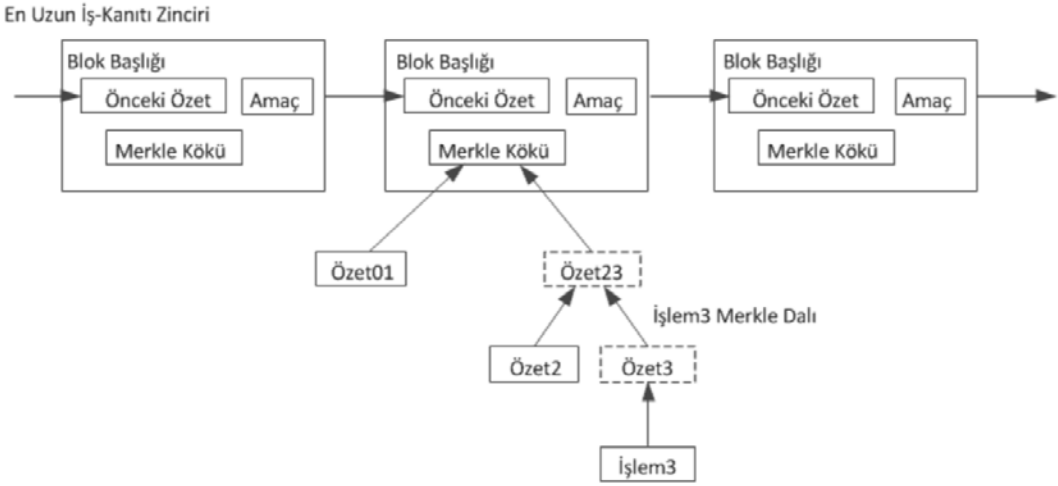
Bitcoin ağındaki her bir blok başlığı işlemler hariç yaklaşık **80 bayt** uzunluğundadır. Her 10 dakikada bir yeni blok üretildiğini varsayarsak 80 bayt * 6 * 24 * 365 = **4.2 MB/yıl** eder. 2008 yılında tipik bilgisayar sistemlerinin 2 GB RAM ile satıldığı ve Moore yasasına dayanarak senede 1.2 GB büyüme öngörüsüyle, blok başlıkları hafızada tutulsa dahi, saklama bir sorun olmayacaktır. Yıl 2019'u gösterdiğinde Bitcoin'in Blockchain boyutu 65 GB'tı geçmiş bulunuyor.



Şekil 1.19: Merkle Ağacı yapısı sayesinde ağda saklanan veri boyutu daha düşük oluyor.

ÖDEME DOĞRULAMASI

Bitcoin ağında ödemelerin doğrulanması için basitleştirilmiş bir yöntem de bulunur. Bu yöntem sayesinde, tam bir düğüm işletmeden de ödemeleri doğrulamak mümkün olur. Basit ödeme doğrulaması metodunda kullanıcının, sadece en uzun iş-kanıtı zincirine dâhil olan bloklara ait başlıkların **bir kopyasını** saklaması yeterlidir. En uzun zincire sahip olduğuna kanaat getirinceye kadar ağdaki düğümleri sorgulayarak bu başlıkları toplar ve işlemi, damgalandığı bloğa bağlayan Merkle dalını oluşturmuş olur. Bu yöntemde işlemi kendisi doğrulayamaz ama zincirdeki bir yer ile ilişkilendirmekle bir ağ düğümünün kabul ettiğini görür. Ardından eklenen bloklar da ağın işlemi kabul ettiğini onaylarlar.



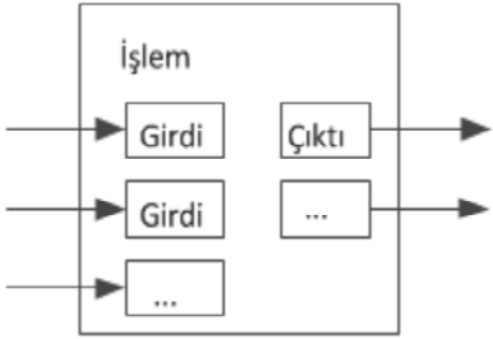
Şekil 1.20: Basitleştirilmiş Ödeme Doğrulama yöntemi akış şeması.

Bu yöntemde doğrulama işlemi, dürüst düğümler ağ kontrolünü ellerinde bulundurdukları sürece güvenilirdir. Ancak ağdaki hesaplama gücünün çoğunluğu art niyetli bir bilgisayar korsanının elindeyse bu yöntem **savunmasız** hâle gelir. Ağ düğümleri işlemleri kendileri doğrulayabilirken basitleştirilmiş yöntem, saldırganın ağa hâkim olduğu süre boyunca ürettiği sahte işlemler ile kandırılabilir. Bu soruna karşı korunma stratejilerinden biri ağ düğümlerinin geçersiz bir bloğu fark ettiklerinde, alarm üreterek kullanıcıdan tüm bloğu ve şüpheli işlemleri yüklemesini istemesi olabilir.

Sık ödeme alan cüzdanlar muhtemelen güvenliklerini sağlamada daha bağımsız olmak ve daha hızlı doğrulama yapabilmek için kendi ağ düğümlerini işletmek isteyeceklerdir. Bitcoin ağına bağlı bir şekilde ödeme ve tahsilat yapılıyorsa en doğru seçenek, **Bitcoin Core** uygulamasını kurarak kendi düğümünüzü yönetmenizdir.

DEĞERLERİ BİRLEŞTİRME VE BÖLME

Bitcoin ağında her bir Bitcoin ödemesini teker teker takip etmek mümkün olsa da havale edilecek her bir **Satoshi** (*1 BTC = 100 Milyon adet Satoshi*) için ayrı ayrı işlem açmak pratik bir yöntem değildir. Değerin bölünebilmesi ve birleştirilebilmesi için işlemler birden fazla girdi ve çıktıdan oluşur. Genellikle ya önceki büyük bir işlemden gelen tek bir girdi ya da küçük miktarları birleştiren birden fazla girdi olur. En fazla da iki çıktı olur: Birisi ödeme tutarıdır, diğeri de varsa parayı gönderene geri dönen para üstüdür. (*Kripto para işlemlerinde genellikle para üstü ödemesi söz konusu değildir. 1 BTC 100 Milyon eşit parçaya bölünebildiğinden her zaman tam tutar ödemek mümkündür. O yüzden para üstü işlemlerine gerek duyulmaz.*)



Şekil 1.21: Ödeme aşamasındaki blok girdileri

Unutulmamalıdır ki **dallanma**, yani her bir işlemin birçok işleme bağlı olması ve o işlemlerin de daha da fazla işleme bağlı olmaları, burada bir sorun değildir. Hiçbir zaman işlem tarihçesinin tam bir kopyasını çıkartmaya ihtiyaç yoktur.

GİZLİLİK

Geleneksel bankacılık modeli bilgiye erişim yetkisini ilgili taraflar ve güvenilir bir üçüncü tarafla sınırlayarak bir noktaya kadar gizliliği sağlar. Tüm işlemlerin açık şekilde ilan edilmesi ihtiyacı bu yöntemi kullanışsız bırakır ama bilgi akışını başka bir noktadan keserek hâlâ gizlilik sağlanabilir. Bunun için Bitcoin ağı, açık anahtarları isimsiz tutarak gizlilik sağlar. Böylece ağdaki birisinin bir başkasına belli bir miktar Bitcoin gönderdiği izlenebilir ancak işlemi kimin yaptığına dair bir bilgi paylaşılmaz. Yani işlemler **şeffaf** olmakla birlikte **anonim** bir şekilde yapılmış olur. Bu, borsaların işlemlerle ilgili açıkladıkları bilgi miktarına benzer. Tek tek her alış-satış işleminin zamanı ve miktarı herkesin görebileceği şekilde ilan edilir fakat tarafların kimlikleri gizlidir.



Şekil 1.22: Bitcoin ağındaki güvenlik modelleri

Bu yapıya ek bir bariyer olarak açık anahtarın sahibi ile ilişkisini gizlemek için her işlemde **yeni bir anahtar çifti** kullanılmalıdır. Çok girdili işlemlerde bir miktar ilişkilendirme kaçınılmazdır ve işlemin girdilerinin aynı kişiye ait olduğunu açığa çıkarırlar. Korkulan olay, anahtarın sahibinin belli olması durumunda aynı kişiye ait diğer işlemlerin de açığa çıkmasıdır. Bunun için Bitcoin ağı, ön tanımlı olarak ödeme almış olan cüzdan adreslerini sürekli değiştirir. Ödeme alan adres değişeceğinden ödeme alan adreslerle kullanıcı ve önceden alınan ödemelerin bilgisi ilişkilendirilemez. Bu durumun ticaret perspektifinde bazı olumsuz yan etkileri vardır. Bunlara ilerleyen bölümlerde değineceğim.

AĞIN GÜVENLİĞİ

Dürüst zincirden daha hızlı şekilde alternatif bir zincir oluşturmaya çalışan saldırganın senaryosuna geri dönelim. Saldırgan amacına ulaşabilse bile, bu olay sistemi yoktan Bitcoin üretmek ya da hiç sahip olmadığı bir Bitcoin miktarını elde etmek gibi keyfî değişikliklere karşı savunmasız hâle getirmez. Düğümler geçersiz bir işlemi ödeme olarak kabul etmeyeceklerdir. Dürüst düğümler de bu işlemleri barındıran blokları asla kabul etmeyeceklerdir. Bir saldırgan sadece kendi işlemlerinden birini değiştirerek yakın zamanda harcadığı parayı geri almayı deneyebilir.

Dürüst zincir ile saldırgan arasındaki yarışı "**Rastgele Binom Yürüyüşü**" şeklinde tarif etmek mümkündür. **Başarılı** olma durumu dürüst zincirin bir blok uzamasıdır ve **+1** puan öne geçirir. **Başarısızlık** durumu da saldırganın zincirini bir blok uzatmasıdır ve aradaki farkı **-1** puan kadar azaltır.

Saldırganın belirli bir gecikme farkını kapatabilme olasılığını "**Kumarbazın İflası**" problemine benzetebiliriz. Sınırsız krediye sahip bir kumarbazın bir borç ile oyuna başladığını ve başa baş noktasına gelebilmek için sonsuz sayıda deneme yapabileceğini varsayalım. Kumarbazın beraberliğe ulaşma olasılığını ya da saldırganın dürüst zinciri yakalama olasılığını şöyle hesaplayabiliriz:

p = dürüst bir düğümün sıradaki bloğu bulma olasılığı

q = saldırganın sıradaki bloğu bulma olasılığı

qz = saldırganın z blok geriden gelerek beraberliği yakalama olasılığı

$$q_z = \begin{cases} 1 & , p \leqslant q \\ (q/p)^z & , p > q \end{cases}$$

p > q varsayımıyla, saldırganın yakalamak zorunda kaldığı blok sayısı arttıkça olasılık üssel olarak azalır. Kendi dezavantajına işleyen olasılıklar karşısında eğer en başlarda şanslı bir dizi hamle yapamazsa daha da geride kaldıkça kazanma ihtimali yok denecek kadar azalır.

Şimdi de yeni bir işlemdeki alacaklının, gönderenin işlemi değiştiremeyeceğinden emin olana dek ne kadar beklemesi gerektiğini ele alalım. Gönderenin alıcıyı para ödediğine bir süreliğine inandırmak isteyen ve bir süre sonra işlemi paranın kendisine geri dönmesi için değiştirmeyi planlayan bir saldırgan olduğunu varsayıyoruz. Bu olay olduğunda alıcı haberdar olacaktır. Ama saldırgan bunun için çok geç olmasını ümit eder.

Alıcı yeni bir çift anahtar üreterek açık anahtarı imzalamadan hemen önce gönderene iletir. Bu şekilde göndericinin önceden bir blok zinciri hazırlaması ve üzerinde çalışarak öne geçtiği şanslı anında işlemi gerçekleştirmesi engellenir. İşlem gönderildiğinde kötü niyetli gönderici gizlice işlemin farklı bir versiyonunu içeren paralel bir zincir üretmek için çalışmaya koyulur.

Alıcı işlemin bir bloğa eklenmesini ve ardından bloğa **z blok** daha bağlanmasını bekler. Saldırganın tam olarak ne kadar ilerleme kaydedebildiğini bilmiyordur ama dürüst blokların ortalama beklenen zamanda üretildiklerini varsayarsak saldırganın potansiyel ilerleme miktarı beklenen değere göre **Poisson dağılımı** olur:

$$\lambda = z\frac{q}{p}$$

Saldırganın hâlâ yetişebilme ihtimalini hesaplamak için her ilerleme miktarının Poisson yoğunluğunu bu noktadan itibaren yetişebilme olasılığı ile çarpıyoruz:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & , k \leqslant z \\ 1 & , k > z \end{cases}$$

Sonsuz dağılım kuyruğunun toplamını almaktan kaçınmak için tekrar düzenliyoruz.

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Şimdi bu formülleri C koduna dönüştürelim:

```
#include

double SaldirininBasariOlasiligi(double q, int z)
{
double p = 1.0 - q;
double lambda = z * (q / p);
double sum = 1.0;
int i, k;
for (k = 0; k <= z; k++)
{
double poisson = exp(-lambda);
for (i = 1; i <= k; i++)
poisson *= lambda / i;
sum -= poisson * (1 - pow(q / p, z - k));
}
return sum;
}
```

Birkaç denemenin sonunda gözlemliyoruz ki olasılık z'ye bağlı olarak katlanarak azalıyor.

q=0.1
z=0 P=1.0000000
z=1 P=0.2045873
z=2 P=0.0509779
z=3 P=0.0131722
z=4 P=0.0034552
z=5 P=0.0009137
z=6 P=0.0002428
z=7 P=0.0000647
z=8 P=0.0000173
z=9 P=0.0000046
z=10 P=0.0000012

q=0.3
z=0 P=1.0000000
z=5 P=0.1773523
z=10 P=0.0416605
z=15 P=0.0101008
z=20 P=0.0024804

z=25 P=0.0006132
z=30 P=0.0001522
z=35 P=0.0000379
z=40 P=0.0000095
z=45 P=0.0000024
z=50 P=0.0000006

P'yi %0.1'den küçük değerler için çözdüğümüzde şu şekilde sonuç alıyoruz:

P < 0.001
q=0.10 z=5
q=0.15 z=8
q=0.20 z=11
q=0.25 z=15
q=0.30 z=24
q=0.35 z=41
q=0.40 z=89
q=0.45 z=340

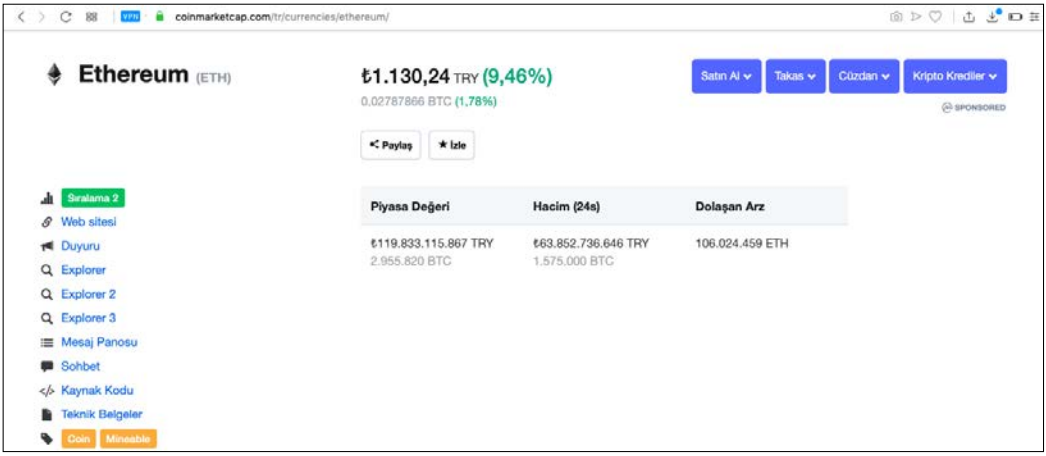
Bu mini uygulamada da gördüğünüz gibi, ağın saldırı stresi altında kırılma imkânı yoktur. Bitcoin ağı bize yepyeni ve kırılmaz bir veri saklama yönteminin yolunu göstermiş oldu. Üstelik otonom ve merkezsiz bir şekilde... Yakın gelecekte veri saklama türleri tamamen bu yapıların üzerine geçiş yapmış olacak.

Bitcoin ağı bize Blockchain teknolojisinin ilk örneğini sunuyor. Yakın bir geçmişte Bitcoin destekçileri ve gönüllü geliştiricileri tarafından desteklenen birkaç **fork** (*çatallanma*) yaşandı. Bu çatallanmalar sayesinde Bitcoin ağı geliştirilirken yeni özellikler kazanmış Bitcoin tipleri hayata **merhaba** dedi. Dikkat ettiyseniz Bitcoin **türevi** demek yerine **tipleri** demeyi tercih ettim. Çünkü bu yeni coinler altcoin olmak yerine bilfiil Bitcoin adını kullanmaya talip olmayı tercih ettiler. Bu aşama sancılı bir süreçti ve Bitcoin, USD değeri tarafında düşüş yaşamaya devam ettiği için çatallanma destekçileri Bitcoin adını telaffuz etmeyi tercih ediyorlardı.

Nihayete erdiğimizde **Bitcoin Cash**, Bitcoin adını kullanmaya başlarken; çatallanmaya eski ağıyla devam eden emektar Bitcoin ise **Core** ya da **Bitcoin Core** olarak anılmaya başlandı. Tabii piyasalarda bomba etkisi yapan bu köklü değişimlerin fitilini ateşleyen yapı, **Ethereum** olmuştu. Önceleri, yatırımcıların ciddi bir kazanç yaşamasına neden olduğu için gündeme oturan Ethereum,

sonrasında güvenlik açığı yüzünden çatallanmaya giderek **Ethereum** ve **Ethereum Classic** adlı iki farklı ağa bölündü. Ethereum bu bölünme sonrasında hızla yayılarak **akıllı sözleşme**lerin (*smart contract*) hayatımıza girmesini sağladı. Şu an, akıllı sözleşmelerden bahsettiğimizde aslında Ethereum hakkında konuşmuş oluyoruz. *Peki, nedir bu Ethereum? Nasıl çalışır? Ne vaat eder?* Hadi yakından bakalım.

ETHEREUM AĞI



Şekil 1.27: 2019'da Ethereum, tek başına (üzerinde kurgulanan token piyasası hariç) 119 Milyar TL değere ulaşarak Bitcoin'den sonra piyasayı domine eden ikinci kripto para olmayı başardı.

Mantık olarak Bitcoin'in açtığı yoldan ilerleyen Ethereum, **Vitalik Buterin** adlı genç bir yazılımcı tarafından, 2013 yılında, **Bitcoin Magazine** (*bitcoinmagazine.com adresli online yayın Vitalik'e aittir*) web sitesinde yayımlanan bir yazı ve ardından **Github** üzerinde paylaştığı **whitepaper** ile doğdu. 2014 yılına geldiğimizde ise Miami'deki **Kuzey Amerika Bitcoin Konferansı**'nda resmi olarak ilan edildi. O andan itibaren kripto para ekosisteminde erken dönem yatırım yapmayı seven hemen herkesin gözdesi oluverdi.

Ethereum (*kısaca ETH*), Bitcoin'den farklı olarak merkezsiz yapılar kurgulamak ve merkezsiz uygulamalar geliştirmek isteyen yazılımcılar için programlanabilir Blockchain yapısı barındırıyor. Bu satırları okuduğunuzda aklınıza programların aynı anda birçok bilgisayarda çalıştırılması nedeniyle yüksek işlem gücü

sağlanması ve paralel işlem gücü elde ederek karmaşık işlemleri hayata geçirmenin kolaylığı gibi grid yapıların sunduğu avantajlar gelebilir ama ne yazık ki Ethereum ağı üzerinden yazılımımızın elde edeceği fayda bu değil. Bilakis, Ethereum ağı üzerinde dağıtık olarak program çalıştırmak, normal bir masaüstü bilgisayar üzerinde çalıştırmaktan **çok daha verimsiz** olur. Çünkü Ethereum ağı, **paralel programlama** veya **grid computing** gibi mantıklar üzerine kurgulanmadığı için yüksek işlem gücü ve performans elde etmek mümkün değildir. Burada üzerine yoğunlaşmamız gereken asıl fayda, çalıştırılan programın hiçbir zaman çökmeyeceğinin garanti edilmesi, programa müdahalenin mümkün olmaması, hata toleransının sağlanması, programa sağlanan verinin Blockchain üzerinde tutulması nedeniyle verinin değiştirilemez olmasının garantisidir.

Ethereum, temelde üç ana yapıdan oluşuyor diyebiliriz: Ana Ethereum Blockchain yapısı, **EVM** kısaltmasıyla kullandığımız **Ethereum Virtual Machine** ve üçüncü parti uygulamaların sahip olduğu yan ağlar ile bunlara bağlı token'ler. Token yapısı için bir akıllı sözleşmenin nihai sonucu da diyebiliriz. Ancak teoride bu kadar yenilikçi ve faydalı olan bir teknolojinin, pratikte yoz bir şekilde, sadece kripto para üretmek için kullanılması, Bitcoin madenciliğinde olduğu gibi Ethereum tarafında da **kaynakların gereksiz tüketimi**ni beraberinde getiriyor.

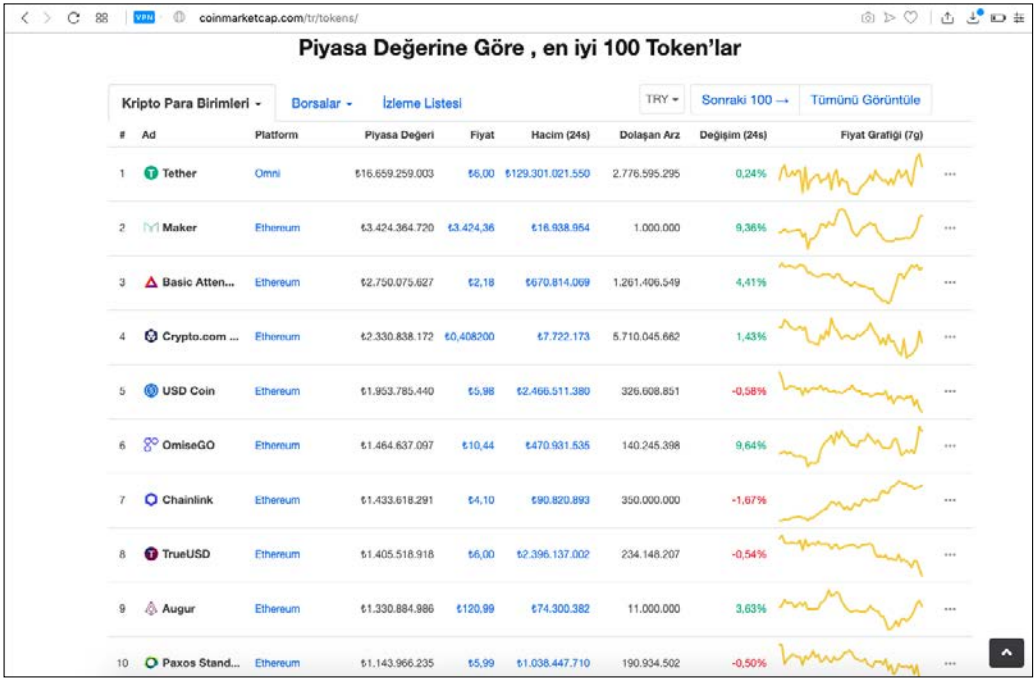
Ethereum ağında bir merkezsiz uygulama geliştirmek istediğinizde **Solidity** adı verilen özel bir programlama dilini kullanmak durumundasınız. Bu dil ile üretilen kod blokları, EVM üzerinde yorumlanarak çıktıları, Ethereum ağına bağlı çalışan merkezsiz uygulamalar olarak karşımıza geliyor. EVM ve Solidity sayesinde oluşturduğunuz bir akıllı sözleşmeyi tekrar çağırıp üzerinde değişiklik ya da geliştirme yapma imkânınız da var. **Workshop** bölümünde bu dil hakkında daha kapsamlı bilgiler sizleri bekliyor.

Ethereum'un Bitcoin'den ayrıldığı bir başka nokta da **toplam arz** konusudur. Bildiğiniz üzere Bitcoin ağında madencilik yöntemiyle üretilebilecek maksimum coin sayısı **21 milyon** adettir ve değiştirilemez. Ethereum toplam arz tarafında bir sınır uygulamaz. Yani **sonsuz sayıda** ETH vardır diyebiliriz. Ancak enflasyon ve piyasadaki ETH değeri noktalarındaki endişeleri gidermek adına her yıl madencilikle arz edilecek token sayısı sabitlenmiştir. Madencilik tarafında Ethereum ağı da Bitcoin ağında olduğu gibi **PoW** (*Proof of Work*) yöntemini kullanır. (*Bu yöntemin yakın zamanda PoS -Proof of Stake- ile*

değiştirilmesi gündemdedir.) Burada kullanılan hash algoritması ise **SHA-3** olarak adlandırılan, Ethereum ekibi tarafından geliştirilmiş bir algoritmadır. Bitcoin ise bu aşamada genel kabul görmüş hash algoritmalarından birisi olan **SHA-256** kullanır.

Şimdi Ethereum ağını oluşturan parçalara yakından bakalım.

TOKEN

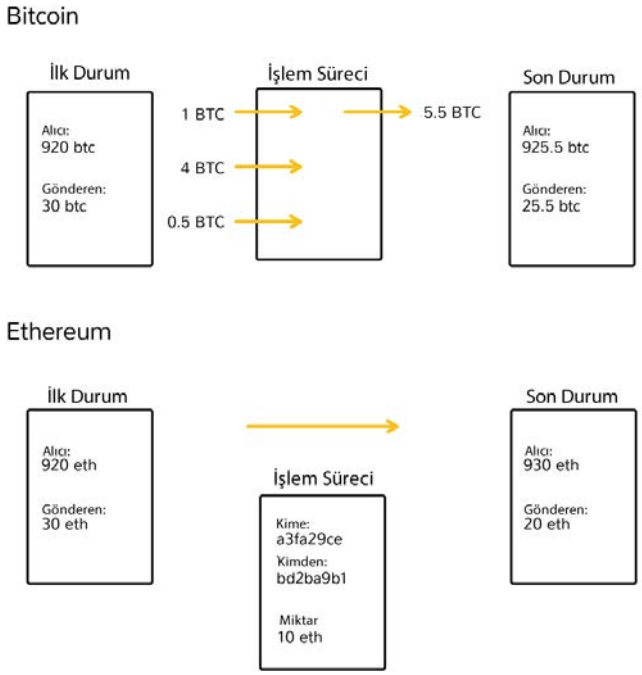


Şekil 1.28: 2019 yılının ilk çeyreğinde, en popüler ilk 10 token'nin piyasa değeri 30 Milyar TL'nin üzerindeydi.

*Yukarıda yaptığım girişte "**Coin**" yerine "**Token**" tabirini kullandığıma dikkat ettiniz mi?* İngilizce dilinde "**coin**"; madeni para, jeton ve Blockchain sonrasındaki bilişim dünyasında kripto para anlamına geliyor. Ancak jeton anlamına gelen bir İngilizce kelimemiz daha var: "**token**".

Ethereum ağı hayata geçip akıllı sözleşmeler aracılığıyla yeni kripto paralar üretilmeye başlanınca Blockchain ekosistemi bu iki tanımı birbirinden ayırdı.

Artık kendi özel Blockchain ağına sahip kripto paralara "**coin**", mevcut bir Blockchain ağı üzerinde yan ağ olarak (*side-chain*) çalışan kripto paralara ise "**token**" demeye başladık. Yani biri size token diyorsa bunun kendi blockchain ağı olmayan ama Ethereum ağı üzerinde bir akıllı sözleşmeyle üretilmiş, yan ağ olduğunu anlamanız gerekiyor. Mevzu bahis olan para bir coin ise onun kendi özel Blockchain ağına sahip olduğunu anlamalısınız. *Bu neden önemli?* Token dediğimiz yapılar, kurgulandıkları modelden farklı olarak hareket edemezler. Yan ağ yapıları, üzerinde koştukları ana ağdan bağımsız olamayacakları için ana ağın tüm zayıflıklarına ve kısıtlarına sahiptirler. Bu noktada farklı uygulama alanlarına sahip modeller arasında geçişler öngörüyorsanız, ya her modeliniz için yeni akıllı sözleşmeler oluşturacaksınız ya da ETH ağını kullanmak yerine kendi özel Blockchain ağınızı oluşturacaksınız.



Şekil 1.29: Bitcoin ve Ethereum ağlarının arasındaki farklardan birisi de ağdaki işlemleri kaydetme yöntemleridir. Ethereum banka mantığında işlem kaydı tutarken BTC ağında, cüzdanlardaki miktar yerine cüzdandan cüzdana yapılan gönderimler hesaplanır.

Konuya yatırımcı perspektifinden baktığımızda ise Ethereum ağı üzerinde token oluşturmanın kolaylığı ve ICO çılgınlığının moda hâline gelmesi yüzünden piyasada **kripto para enflasyonu** yaşandığını görüyoruz.

İlk dönemlerde 1000 adeti aşmayan kripto para türleri, Ethereum sonrasında 10 bin adete merdiven dayadı. Durum böyle olunca hacmi azalan kripto paraların sürekli borsalar tarafından liste dışı bırakıldıklarını görüyoruz. Tabii bu durum sektörün gelişimindeki en büyük engellerden birini oluşturuyor. Ancak handikaplardan bir başkası da günün birinde Ethereum ağında oluşturulmuş bir token'in değer bakımından Ethereum'u geride bırakabileceği inancıdır. Unutulmamalıdır ki yaşamsal fonksiyonları Ethereum ağına bağlı bir token hiç bir zaman Ethereum'dan fiyat kurlar (*USD, EUR, GBP ya da TL gibi ülke paralarına fiyat kurlar diyoruz.*) tabanında daha değerli hâle gelemez. Token oluştururken önceliği piyasadaki borsalarda alınacak hacim ve değerden ziyade, kurgulanan merkezsiz yapıdaki iş modeline vermelisiniz. Bu sayede ağın amacına uygun şekilde, merkezî olmayan uygulamalar geliştirebilirsiniz.

Ethereum üzerinde uygulama yazmak ya da en basit hâliyle hesap açarak **ETH yatırımı** istiyorsanız öncelikle bir hesap oluşturmanız gerekiyor.

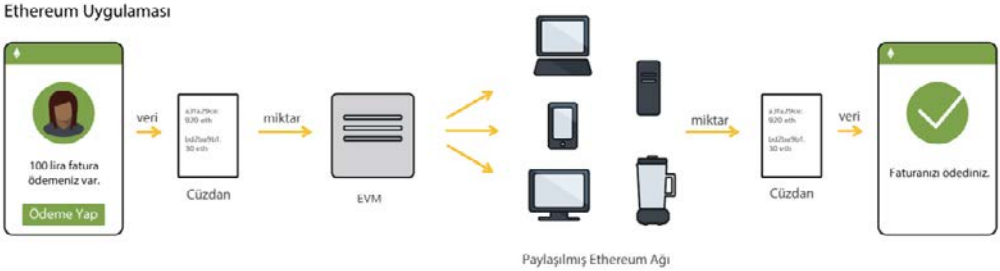
Her ağ yapısında olduğu gibi Ethereum ağında da işlemler gerçekleşmek için yetkilere ve hesaplara ihtiyaç duyar. Ethereum ağında iki tip hesaba sahibiz: **Haricî Olarak Sahip Olunan Hesaplar - Externally Owned Accounts** (*EOA*) ve **Sözleşme Hesapları - Contract Accounts**.

EOA, yani haricî olarak sahip olunan hesaplar, **özel anahtarlar** (*private key*) tarafından kontrol edilir. Bitcoin'de olduğu gibi genel kullanımdaki normal hesaplardır. Bir hesap için özel anahtara sahipseniz o hesaptaki ETH transferi dâhil tüm gönderim (*transaction*) işlemlerini gerçekleştirme yetkisine sahipsiniz demektir. Sahip olduğunuz ETH miktarını tutar ve transfer işlemleri yapmanıza imkân tanır. Eğer bir geliştiriciyseniz ve Ethereum ağı üzerinde bir merkezsiz uygulamanız varsa (*bu bir akıllı sözleşme de olabilir*) onu tetiklemek için de kullanabilirsiniz.

Sözleşme Hesapları, ağa bağlı kullanıcılar tarafından kullanılamazlar. Bu tip hesapların yönetimi ağ üzerinde kodlanmış merkezsiz uygulamalar ya da akıllı sözleşmeler tarafından yapılır. Uygulama üzerinde sahip olduğumuz ETH miktarını tutarlar. Bu hesap tipinde, uygulamanın çalışması için diğer hesaplardan gelen gönderiler (*transaction*) veya mesajlar gereklidir.

Tetikleyici gönderiyi kendi hesabınızdan yapabileceğiniz gibi başka bir otonom akıllı sözleşmeye de yaptırabilirsiniz.

ETHEREUM VIRTUAL MACHINE VE AKILLI SÖZLEŞMELER



Şekil 1.30: EVM, Ethereum ağının üzerinde oluşan hemen her işlemi yorumlayarak işleme alan yapıdır.

Ethereum Virtual Machine ya da kısaca **EVM**, Ethereum Blockchain'i üzerinde kodlanan programların çalıştırılmasını üstlenir. Okurlar arasında Java programlama diline hâkim olanlar varsa bu yapıyı **Java Virtual Machine** ile benzeştirebilirler. Aslen mantık aynıdır. Blockchain üzerinde koşan yazılımların ağ üzerinde dağıtık şekilde çalışabilmesi için EVM onları yorumlayıp ağdaki bilgisayarlara (*düğümlere*) gönderir. Burada tüm yükün EVM üzerinde olduğunu anlamışsınızdır sanırım. Hem Ethereum ana ağını hem akıllı sözleşmeler ve token'ler ile gelen yan ağları, hem de dağıtık yapıdaki merkezsiz uygulamaları denetleyip çalıştırmak ağır bir yüktür.

Ethereum ağında uygulama yazarken **Solidity** dilini kullanıyoruz demiştim. Bu dil ile yazılan uygulamalara **Smart Contract** ya da Türk diline geçmiş hâliyle **Akıllı Sözleşme** diyoruz. Akıllı sözleşme adı telaffuz edildiğinde akıllarda günlük evrak işlerinde sıklıkla kullandığımız anlaşmalar gelir. Aslında bu kısmen doğrudur. Yani bir iş sözleşmesi ya da sigorta poliçesi yerine geçecek akıllı sözleşmeler kodlayabilirsiniz. Ancak bunların dışında uçtan uca para transferi, anlık sohbet uygulamaları ya da yenilikçi bir sosyal ağ gibi projeler de kolayca Solidity ile akıllı sözleşme adıyla hayata geçirebilir. Bu tanım yerleşmiş bir tanım olduğundan kitapta değiştirmek ya da esnetmek istemiyorum. Ancak siz **akıllı sözleşmeler** denildiğinde hem karşılıklı bir akit, hem de bildiğimiz kurulabilir uygulamalar ya da web projeleri gibi her türlü yapıyı anlamalısınız.

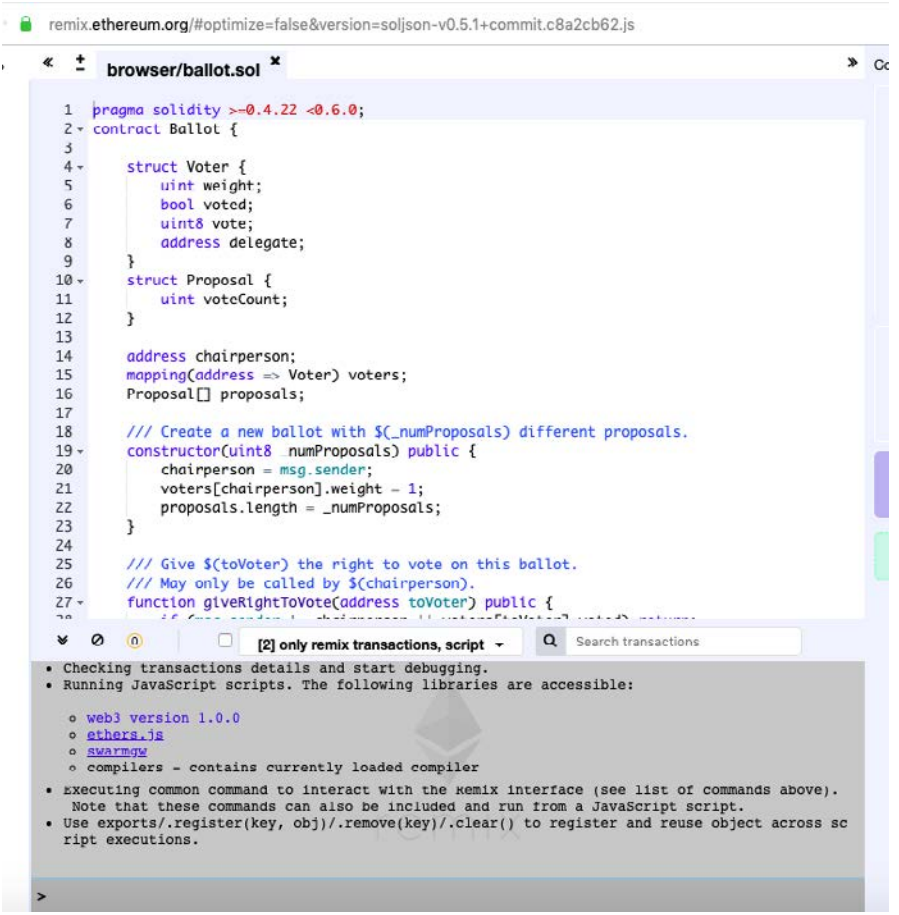
Bunu bir örnekle açıklayalım:

Hakan ve Burak aralarında bir iddiaya girerler. Hakan'ın iddiasına göre Beşiktaş – Galatasaray maçını Beşiktaş kazanacak ya da berabere bitecek. Burak'a göre bu durumun tam tersi gerçekleşecek. Kaybedenin 10 TL'yi kazanana vereceği konusunda aralarında anlaşırlar. Fakat para konusunda ikisi de birbirine hiç güvenmezler, Burak bunu bir akıllı sözleşme kodlayarak çözebileceklerini bildiğinden hemen bir kontrat yazıp devreye alır.

Bu örneğe göre akıllı sözleşmemizin akışı şu şekilde gerçekleştirir:

Öncelikle Hakan ve Burak'tan 10'ar TL alarak bloke eder. Ardından maç tarihi gelip de sonuç kesinleştiğinde, skoru çekerek kimin kazandığına karar verir ve 20TL'yi kazanan kişiye transfer eder. Bir akıllı sözleşme, ağdaki diğer kullanıcılardan farksız bir statüde olduğundan koşullar doğrultusunda bir hesaptan belli bir miktar paranın çekilmesi, bloke edilmesi ve sonra tekrardan bir hesaba yollanması çok sorunsuz ve basit olur. Ayrıca bir akıllı sözleşme başka bir sözleşmeye mesaj gönderebilir, onu tetikleyebilir. Bu şekilde birbirinin devamında farklı süreçleri işleten birden fazla akıllı sözleşmeden oluşan zincirler ile uygulama süreçlerinin daha da geliştirilmesi mümkündür.

SOLIDITY PROGRAMLAMA DİLİ



Şekil 1.31: Solidity dili için geliştirilmiş en başarılı IDE'lerden birisi, web tarayıcınızda eklenti olarak çalışabilen Remix'tir. https://remix.ethereum.org/ adresinden ulaşabilirsiniz.

Akıllı sözleşmeler geliştirme süreci Solidity programlama dili ile yapılır demiştik. Solidty, Ethereum çekirdek ekibi tarafından geliştirilmekte olan ağa özel bir geliştirme dilidir. Dolayısıyla yazılan kodlar Ethereum ağı dışında çalışamazlar.

Yapısı biraz **Python**, biraz **C/C++** ama daha çok da **Javascript** dillerinin yazım yapısına benzeyen bir programlama dilidir. Solidity ile bir akıllı sözleşme programladığınızda kodlarınızın Solidity derleyebilen bir derleyici ile derlenmesi gerekir. Ardından derlenen uygulamanızın Ethereum ağına

gönderisi (*transaction*) yapılır. Böylece uygulamanız dağıtık yapı içerisinde çalışmaya başlamış olur.

Örnek bir akıllı sözleşme kodu:

```
pragma solidity ^0.4.0;
contract SimpleStorage {
    uint storedData;
    function set(uint x) public {
        storedData = x;
    }
    function get() public constant returns (uint) {
        return storedData;
    }
}
```

Solidity kodlayabilmek için bir **derleyici**ye ihtiyacınız var. Ethereum ekibi bunun için browser tabanlı çalışan ve internet bağlantınız yokken de kod yazabileceğiniz bir test ortamı geliştirmiş durumda. Kullanımı oldukça pratik olan derleyiciyi **Chrome** motoru ile kullanabilirsiniz. *https://github.com/ethereum/browser-solidity/tree/gh-pages* adresindeki **ZIP** dosyasını indirip hemen ilk sözleşmenizi yazabilirsiniz.

Solidity ile ilgili daha detaylı bilgi ve uygulama örneklerini **workshop** bölümünde bulabilirsiniz.

İŞLEM ÜCRETLERİ VE GAS

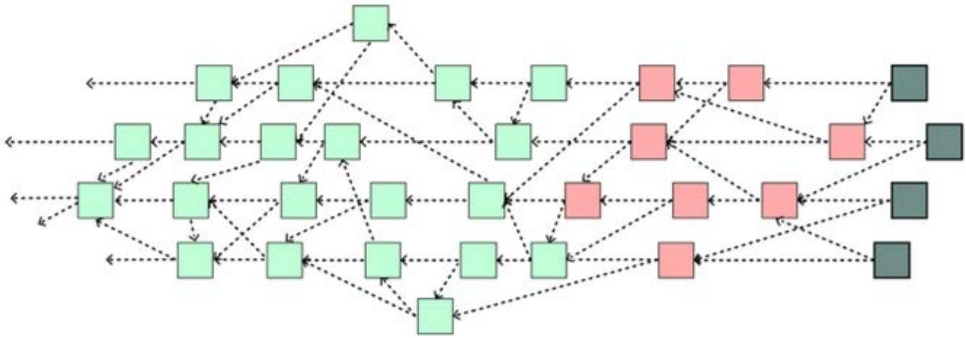
Ethereum ağında akıllı sözleşmelerin çalışması ve cüzdandan cüzdana para gönderme gibi işlemlerin gerçekleştirilebilmesi için bir ücretlendirme sistemi mevcuttur. Bu ücretlendirme sistemine **Gas** adı verilir ve Gas tahsilatlarında **Ether** para biriminin en küçük parçaları kullanılır. Gönderim işlemi veya akıllı sözleşme kodunun çalışabilmesi için gerekli Gas miktarı ve kullanabileceği maksimum Gas miktarı belirlenir. Aslında bu yapıyı Bitcoin ağındaki **gönderim komisyonu** (*transaction fee*) ya da **işlem ücreti**ne benzetebiliriz. Gas için sabit bir değer aralığı yoktur. Kodladığınız akıllı sözleşmenin çalışabilmesi için ağda kullanılan işlem gücü, veri erişim miktarı, kullandığı kaynak veya diğer belirleyici unsurlara göre gereken Gas miktarı *artar* ya da *azalır*.

Akıllı sözleşmelerin çalışması için gerekli olan Gas harcamaları, hesabınızda bulunan **Ether** (*ETH*) ile satın alınır. Programı çalıştırmak için gaz gereksinimlerini karşılayacak yeterli Ether yoksa işlem iptal olur ve tüm ara durum değişiklikleri işlem öncesi anlık durumuna geri döner. Bunun için mutlaka cüzdanınızda bir miktar ETH bulundurmalısınız.

Gas aynı zamanda, Ethereum'daki karmaşık hesaplamaları ağ üzerinde çalışmak için güvenli ve kontrollü hâle getiren hayati bir mekanizmadır, çünkü kötü amaçlı ya da kontrolden çıkmış programlar yalnızca çalıştırılmasını isteyen geliştiricilerin cüzdanlarındaki ETH finansmanı bitene kadar hayatta kalacaktır. Gas ödemeleri tahsil edilemez hâle geldiğinde, madenciler akıllı sözleşmenin işlemi üzerinde çalışmayı durdururlar. Ağda bir uygulamayı ücretsiz olarak çalıştırma olanağı olmayacağından sisteme yapılması muhtemel **DDOS** gibi olası saldırıların maliyeti çok yüksek olacaktır ve bir anlamda bu şekilde sistem güvenliği garanti altına alınmış olur.

Ethereum ağının gelir modeli işlem ücretleri üzerine kurgulanmıştır. Dolayısıyla uygulamalarınızın **sürekli** ve **kesintisiz** bir şekilde çalışmasını istiyorsanız Ether cüzdanınızı sürekli dolu tutmaya çalışmalısınız. Bunun ilk yöntemi bir borsa yapısı üzerinden ETH satın almaktır. Ancak ETH kuru sürekli dalgalandığından sizin için aşırı maliyetli bir hâle dönüşebilir. Zahmetli ama daha kârlı olan yol, **madencilik** yapmaktır. Bu sayede kendi kaynağınızı üretmeye başlayarak sürekli bir döngü yaratmış olursunuz.

IOTA AĞI: TANGLE



Şekil 1.32: IOTA Tangle blok, zincir ve madencilik barındırmayan bir merkezsiz ağ örneği olarak karşımıza çıkıyor.

IOTA, 2015 yılında **David Sønstebø**, **Sergey Ivancheglo, Dominik Schiener** ve **Dr. Serguei Popov** tarafından bir vakıfla birlikte kurulmuş özel bir kripto

para birimidir. Bu bölümde yer almasının sebebi, kendine özel bir ağ yapısına sahip oluşu. **Tangle** adıyla anılan bu yapı, teoride Blockchain gibi çalışsa da blok ve zincir yapısı bulundurmadığı için Blockchain olarak adlandıramıyoruz. Ama Blockchain perspektifinden baktığımızda sektörde **üçüncü bir ağ alternatifi** olarak kendini kabul ettirmiş durumda.

IOTA ekibi, geliştirdikleri Tangle adlı dağıtık ağ yapısının mevcut Bitcoin ve Ethereum ağlarından çok daha ileri bir noktada olduğunu savunuyor. Üstelik Tangle'nin "**gerçekten**" merkezsiz çalışan tek ağ yapısı olduğunu vurguluyorlar. Şimdi gelin *IOTA nedir, ne işe yarar ve Tangle, ataları sayabileceğimiz Bitcoin ve Ethereum ağlarından ne kadar farklı* yakından inceleyelim.

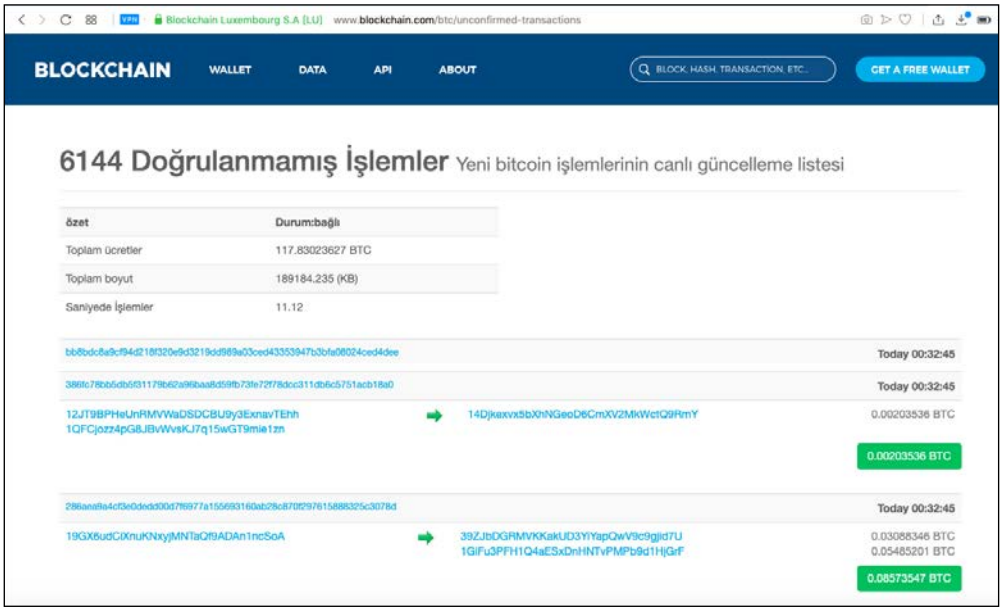
IOTA, adından da anlayabileceğiniz üzere, geleceğin üzerine inşa edilmeye başlandığı **IoT** (*Internet of Things*) yani **Nesnelerin İnterneti** teknolojilerine odaklanmış bir platform ve kripto para birimi. Amacı da her türden elektronik cihazı birbiriyle, IOTA ağı (*yani Tangle*) üzerinde iletişime sokabilen bir platform olmak.

Tangle ağı, kimliksiz ve merkezsiz bir dağıtım modeli üzerine kurgulanmıştır. Ekip, bunun temel amacını **veri transferlerini kullanması** şeklinde açıklıyor. Ağ üzerindeki düğümlerden toplanan veriler anonim bir biçimde kullanıma sokuluyor. Yani IOTA'nın hedefi, kullanıcı verilerini toplamak ve işleyerek ağa sponsor olan firmalara raporlar sunmak.

Tabii toplanan veriler ağ içerisinde IOTA kripto parasıyla dolaşıyor olacağından, IOTA'nın işlem gördüğü borsalardan IOTA aldığınızda aynı zamanda data da satın almış oluyorsunuz. loT ve datanın her şey olduğu bu günlerde IOTA'nın kurguladığı data paylaşım modeli oldukça efektif duruyor. Tabii teknik anlamda çalışıyor olması çok bir şey ifade etmiyor. Aynı zamanda ciddi bir kullanıcı sayısına ulaşılması şart.

Bitcoin ve Ethereum gibi madencilik esaslı ağlarda işlemler çoğunlukla **madenciler** tarafından onaylanır. Bu nedenle yüksek işlem ücreti öneren işlem sahiplerinin gönderimleri hemen ve öncelikli olarak onaylanır. IOTA'da blok mantığı ve madencilik olmadığından işlemlerin onaylanması tercih edilen bir ücretli iş değildir. IOTA cüzdanına sahip her kullanıcı kendisinden önceki iki işlemi onaylar ve sistem bu şekilde devam eder. Dolayısıyla IOTA ağında para gönderimlerinde işlem ücreti yoktur.

ÖLÇEKLENEBİLİR OLMAK



Şekil 1.33: Bitcoin ağında doğrulanmayı bekleyen işlem sayısı oldukça fazla. Bu durum ağın yavaşlamasına neden oluyor. Sebeplerin başında ise işlem ücretlerinin madencileri tatmin etmemesi geliyor.

Bitcoin ve Ethereum ağlarına katılan her yeni kullanıcı sistemde gerçekleşen yeni bir işlem demek oluyor. İşlem yoğunluğu arttıkça altyapı zorlanıyor ve kurgulanan sistem kullanılabilir olmaktan uzaklaşıyor. Tangle ağ yapısı bu sorunun üstesinden gelmek adına **yapısal değişiklikler** barındırıyor. Örneğin Blockchain'in aksine Tangle üzerinde sisteme katılan her yeni kullanıcı bir **düğüm** olarak (*node*) tanımlanıp sistem üzerindeki yükün yoğunluğuna ortak ediliyor. 2019 Nisan ayı itibariyle Blockchain üzerinde her saniye ortalama 50000 işlem onaylanmak için sırada bekliyor. Üstelik yaklaşık 6000 adet işlem onay için kuyrukta bekliyor.

Saniye başına gerçekleşen işlem sayısı bir finansal altyapının ölçeklenebilirliğini göstermesi açısından önemli bir indikatör ve aşağıdaki tabloda bazı finansal sistemlerde 1 saniyede gerçekleşebilen işlem sayısı yer alıyor.

Para Birimi - Finansal Altyapı	İşlem / Saniye
Bitcoin	4,5
Ethereum	13
Paypal	113
VISA (Kartlı İşlem Sağlayıcı)	4.000
IOTA	500

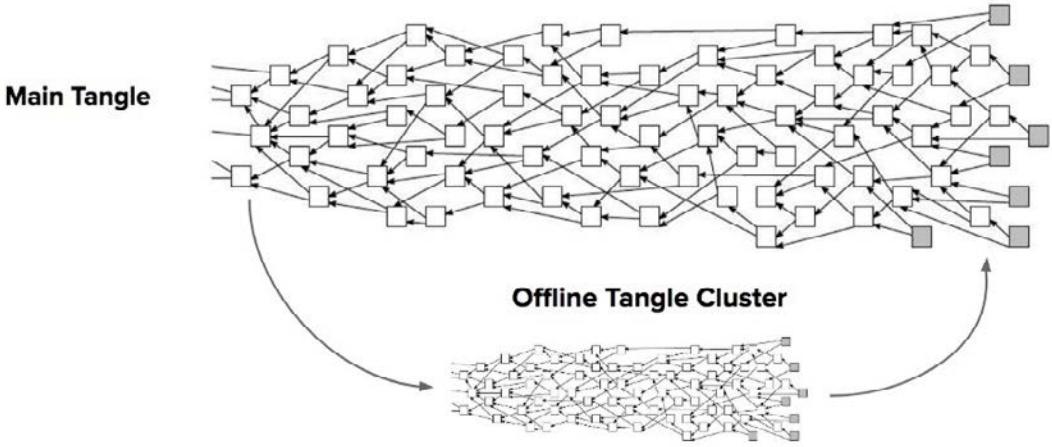
Şekil 1.34: Finansal altyapılar ve saniye başı düşen işlem adetleri

Mevcut kabul görmüş finansal yapılara meydan okumak için, ortaya konulan sistem performansının mevcuttaki muadillerine göre daha kullanılabilir olması beklenir. Öyleyse Bitcoin ve Ethereum gibi ağların daha çok yolu varken IOTA'nın VISA'yı yakalayabilmesi için sisteme katılan kullanıcı sayısının katlanarak artması gerekiyor.

KUANTUM BİLGİSAYARLI HACKER

Kripto para dünyasındaki en büyük korkulardan birisi, tüm güvenlik yapılarının geçilerek ağın kırılması ve kullanıcıların paralarını kaybetmesidir. Bu tarz bir senaryo imkânsız olmamakla birlikte oldukça **düşük bir ihtimal**dir ve yapılabilmesi için de donanım teknolojisinin gelişerek **kuantum bilgisayar** dediğimiz cihazların görücüye çıkması gereklidir. İşte IOTA tam da bu noktada bizlere kuantum işlem gücüne sahip bilgisayarların 10 yıl sonra günlük kullanımda olmasının beklendiğini hatırlatıyor ve Blockchain altyapısının çalışılabilir durumda kalması için gerekli **kümülatif işlem gücü**nün (*Hashcash*) neredeyse tamamını bir kuantum bilgisayar tarafından karşılanabilir seviyede olacağının altını çiziyor. Yani 10 yıl sonra bir hacker, kuantum bilgisayarının işlemci gücünü kötü niyetle kullanmak isterse Blockchain mantığında çalışan ağ yapılarını manipülasyona açık hâle getirmiş olacak. IOTA ağında ise bir kuantum bilgisayarın toplam Tangle işlem gücünün (*Hashcash*) %34'üne (*IOTA kurucuları bu oranın %51 ile eşdeğer olduğunu belirtiyor*) tek başına sahip olduğunu varsayarsak bile tüm ağı manipüle etme olasılığı yaklaşık olarak %29'dur. Tangle, kuantum işlemcilerin kötü niyetli kullanılma ihtimaline karşın yapısal olarak **Winternitz One-Time Signature** özelliği kullanır.

ÇEVRİMDIŞI TANGLE KÜMELERİ



Şekil 1.35: Tangle ağının en büyük artılarından biri çevrimdışı olan düğüm kümelerinin de işlem yapmaya devam edebilmesi diyebilirim.

Tangle, bir alt sistem ana ağdan koptuğunda bu alt sistemin kendi içinde ticaret yapmasına ve fırsat bulup ana ağa bağlandığında tüm kullanıcıların oluşturduğu **konsolide bilanço**ya, söz konusu çevrimdışı ticarette gerçekleşen işlemlerin eklenmesine olanak tanır. Böylece IOTA ağının üyeleri kısa ya da uzun süreli olarak internete bağlı olmasalar bile Tangle içerisinde ödemeler gerçekleştirilmiş olur. Bu özellik diğer ağ yapılarında bulunmuyor. Kimilerine göre büyük bir fırsat, kimileri içinse ağın bir açığı durumunda. Aslında bunu değerlendirirken büyük resme bakmamız gerekiyor. Tangle üzerinde madencilik ve blok mantığı olmadığından çevrimdışı işlemler manipülatif etki yaratmıyor.

SON SÖZ

Üç farklı Blockchain yapısını her yönüyle birkaç sayfaya sığdırmamız kolay değil. Ancak en çarpıcı taraflarını ve çalışma mantıklarını sizlere aktarmaya çalıştım. Bu bölümde Blockchain yapılarının aslında ne kadar gelişmiş kayıt tutma ağları olduğunu gördünüz. Bu yapıların günümüz muhasebe kayıt sistemlerinden en büyük farkı, içerisinde canlı ve sürekli büyüyen bir ekosistem barındırıyor olmasıdır. Bu yapı sayesinde üzerlerine farklı yapılar inşa etmek de kolaylaşıyor. Akıllı sözleşmeler bu tarz işler için biçilmiş kaftan. Ancak Tangle gibi çevrimdışı da kayıt tutabilen yapılar yakın gelecekte bizleri çok hızlı bir şekilde hayal dünyamızın sınırlarına taşıyacaklar.

Bu bölümden anlamanızı beklediğim iki sonuç var: İlki, geleceğin kuşku götürmez bir biçimde bu ağların egemenliğinde olacağıdır. İkincisi ise geleceği yakalamakta gecikmeden uygulama fikirlerinizi ya da mevcut iş modelinizi Blockchain teknolojisine geçirmeyi düşünmenizdir. Bunu yaparken ihtiyaç analizinizi çok iyi yapmanızı ve fikrinize en uygun olan ağ yapısını seçmenizi öneririm.

YENİ EKONOMİ VE BLOCKCHAIN

2



Şekil 1.36: Yeni ekonomi ile doğan e-ticaret, kredi kartları ve online ödeme sistemlerinin gelişmesinde büyük rol oynadı. Aynı zamanda global ekonomide kargo taşımacılığı başta olmak üzere, lojistik ve ithalat – ihracat alanlarında ciddi büyüme yaşanmasına neden oldu.

Kitaba ticaret, muhasebe ve paranın tarihî gelişimiyle başlamıştım. Aslında muhasebe ve para dediğimiz olgular, ticaretin evrimiyle karşımıza gelen ve **insan eliyle yaratılmış** yapı taşlarıdır. Yaşayan, öğrenen ve sürekli gelişen yapılardır. 1950'lilerden bu yana bilgisayarların hızla gelişmesi ve ucuzlayarak evlerimize kadar girmesi, kayıtların bilgisayarlarla dijital ortamlarda saklanmaya başlanmasına neden oldu. Ancak 80'li yıllardan sonra hayatımıza giren internet sayesinde "**işin şekli**" değişti. Artık iş, internette yapılıyor ve satışlar bir aracı olmadan kolayca gerçekleşiyordu. Hem de 7 gün ve 24 saat alışveriş yapılabiliyordu. Ticaret evriminin yeni halkasına gelmişti ve ekonomi ister istemez ikiye ayrıldı: **Yeni Ekonomi** ve **Eski Ekonomi**.

Blockchain öncesinde bildiğimiz "**yeni ekonomi**", online ürün satışı, içerik üretimi ve çok çeşitli alanlara dağılmış, online servislerden oluşmaktaydı. Müşteriyi tanıma, müşteri odaklı ve anlık pazarlama yapabilme, sosyal ağ merkezli çeşitli gelir modelleri ve "**Big Data**" özelinde kurgulanmış yapılardan ibaretti. Blockchain ile bu yapı çok farklı bir noktaya ulaştı. Artık yeni ekonomi denildiğinde, kişinin veya şirketin prosedürlerden bağımsız şekilde kendi kaynaklarını üretebilme gücünü de içerisinde barındırıyor. Bu güç, devletleri ve kartelleri korkutuyor, kripto para odaklı piyasaların gelişmesine zaman zaman köstek olabiliyor... Tabii şimdilik!

Son yıllarda yapılan araştırmalar, iş dünyasına atılacak gençlerin büyük bir çoğunluğunun **bilişim**, **teknoloji** ve **IoT** alanlarında girişimlere odaklandığını gösteriyor. Eski ekonomi içerisinde olmak isteyenlerin neredeyse tamamına yakını internet aracılığıyla geniş pazarlara açılabileceklerinin farkındalar. Üstelik son günlerin popüler terimlerinden biri hâline gelen "**e-ihracat**" kavramı sayesinde yeni ekonomi çapını büyütmeye devam ediyor. Eski ekonomi ise her geçen dakika küçülmeye devam ediyor (*evet, yanlış okumadınız, dakika dedim*) ve yeni ekonominin işlem hacmi içerisinde yok olup gideceği anı bekliyor. O an geldiğinde artık biz de ayrım yapmayı bırakıp sadece "**ekonomi**" demeye başlayacağız. *Sanki tarih tekerrür ediyor değil mi?*

Blockchain teknolojisi doğduğu anda kripto paralar ve sosyal ağların çarpan etkisiyle yeni ekonomiyi daha da büyütüp eski ekonomiyi tarih kitaplarına yollayacak süreci başlatmış oldu. Özellikle **2017** yılında 800 milyar USD piyasa değerini de aşıp birçok köklü sektörün işlem hacmini geride bırakmayı başardı. Ancak yasal boşluklar, regülasyon eksikleri ve devletlerin tavrını çekimser tutması gibi nedenlerden dolayı, e-ticaret dünyasına nüfuz etmek yerine izole bir yapı hâline geldi. Eğer kripto para ve Blockchain dünyasını **e-ticaret** (*yeni ekonomi*) içerisine tam olarak entegre etmeyi başarabilirsek e-ticaretin hacmi trilyonlarca USD değeri aşarak eski ekonominin fişini çekecek güce kavuşabilir.

Peki, bu entegrasyon nasıl olabilir? Böylesi bir devinim devlet eliyle mi olmalı yoksa bu devinimi bireysel olarak yapabilir miyiz? Bu sorunun cevabı **her ikisi de** olacak. Ama sosyal ağlar ve e-ticaret alanlarında devletlerin düzenleme yapmak dışında müdahale ettiğini görmüyoruz. Yeni ekonomideki döngü, bireylerin, fikirlerini internete taşıma kararı almasıyla başladı. Tıpkı bireylerin sosyal hayatlarını dijital sosyal ağlara taşıma kararı alması gibi…

Blockchain ve kripto para dünyasını e-ticaret içerisine almak için ilk adımın **bireyler**den gelmesi gerekiyor. Bireysel girişimler belli bir hacmi bulduğunda devletler regülasyon çıkartmaya ve müdahil olmaya mecbur kalacaktır. Tabii, Blockchain teknolojisi yeni ekonomi dediğimiz e-ticaret olgusunu da ciddi oranda değişikliğe sürükleyecek, hatta tanımını değiştirecek diyebilirim.

Buraya kadar her şey güzel, Blockchain teknolojisini öğrendik, anladık ve kullanmaya hevesliyiz. *Peki ama nereden başlayacağız?* Başlayacağımız nokta her zaman şu sorunun cevabını almak olmalı: *Modelimizin gerçekten Blockchain ağına ihtiyacı var mı?* Blockchain teknolojisini çok sevmiş, benimsemiş ve geleceğin, onun blokları içerisinde olduğunu düşünüyor olabilirsiniz. Ancak eğer iş modeliniz için Blockchain efektif bir çözüm olmaktan çok, ayak bağı olacaksa Blockchain macerasına atılmanız yersiz olacaktır. *En azından mevcut projeniz üzerinden*.

Blockchain Hangi İş Modellerine Çözüm Oluyor?

Şimdi, bu kısımda sizlere diğer Blockchain kaynaklarında olduğu gibi birkaç iş modelinden oluşan bir liste vermem gerekiyor. Siz de bu listeye bakarak işinizi Blockchain ile kurgulayıp kurgulayamayacağınızın kararını vereceksiniz. Ancak bu bölüme kadar hiç klişelere bağlı kalmadığımdan, bu kısımda da klişeden uzak duracağım ve liste paylaşmayacağım. Sizi bir kalıba koymaktansa iş modelinizi Blockchain kalıbına sığdırma işini size bırakacağım. O zaman başlayalım.

Yukarıdaki satırlarda şu soruyu sormuştum: *Modelimizin gerçekten Blockchain ağına ihtiyacı var mı?* Bu soruya cevap ararken kitabın önceki bölümlerinde Blockchain ağ yapısı için "**güven esasına dayalı, dağıtık ağ modeli**" dediğimizi hatırlamanız gerekiyor. Buradan yola çıkarak eğer iş modeliniz güven esasına dayalı bir yapıya sahipse ve bunun daha ileriye taşınması gerekiyorsa Blockchain göçü için kolları sıvamanız şart demektir. Ancak güven ikinci plandaysa ve veri işleme odağındaysanız, yine kitabın önceki bölümlerinde "**Blockchain bir veritabanı değildir, çünkü yazılan verileri silemezsiniz!**" dediğimi hatırlamanız gerekiyor. Yani iş modelinizin özünü iyi anlamalı, ardından "*Bu öz, Blockchain ağının muazzam güven yapısına ihtiyaç duyuyor mu?*" diye kendinize sormalısınız.

Şunu hiç aklınızdan çıkartmayın, iş modeliniz çalıntı ya da devşirme değilse, **özgün** ve **tamamen size ait**se büyük resme baktığınızda neye ihtiyacı olduğunu kolayca görebilirsiniz. Ancak önceden uygulanmış bir modeli ya da iş fikrini devşirdiyseniz süreçleri göz önüne alıp kararınızı yine Blockchain teknolojisinin güven ve zaman damgası özelinde vermeye çalışmalısınız.

Böylesi bir kararın, projeniz üzerinde çok köklü değişiklikler yaratacağını sakın unutmayın. İhtiyaç duyacağınız ekipman, altyapı ve geliştirici ihtiyaçlarınız elinizde mevcut olanlardan farklı olabilir. Daha da önemlisi Blockchain alanında deneyime sahip olmayan yazılımcılardan, programlama alanında ne kadar iyi olduklarından ziyade, **sizin ihtiyaçlarınızı ne ölçüde anladıkları** oranında fayda sağlayabilirsiniz. O yüzden siz de en azından mantıken Blockchain süreçlerine hâkim olmalısınız ki derdinizi anlatabilin.

Genel görüş ise öncelikle devletlerin alacağı kararlarla birlikte Blockchain teknolojisine geçiş yapılacağı şeklindedir. Bu pencereden bakıldığında noterlik hizmetlerinin **otonom** hâle getirilmesi, oy verme altyapılarının **şaibeden uzak** hâle gelmesi, uçtan uca para transferlerinde düşük maliyetler, para basma maliyetlerinden kurtulmanın yanı sıra **kalpazanlığın tamamen bitirilmesi** ve vergi tahsilatlarında **kayıpların sıfıra inmesi** gibi saymakla bitiremeyeceğimiz kadar başlık ve alan çıkıyor karşımıza. Hatta yazdıklarımı okuduğumda, Blockchain teknolojisinin bireylerden çok devletlere yaradığını görüyorum. *O zaman neden hâlâ sağlam adımlar atılmıyor* diye düşünebilirsiniz. Cevabı basit, devletlerin e-ticaret ve internet tabanlı ödeme süreçlerine geçtiği dönemleri hatırlayan ya da araştıran okurlarım varsa **özel sektörden çok daha sonra** olduğunu anımsarlar. Devletler genellikle ar-ge gerektiren, deneysel teknolojiler üzerine balıklama atlamazlar. Öncelikle **swot analizi**ni net bir şekilde yapabilecekleri donelerin oluşması için beklemeyi tercih ederler. Bunun anlamı şudur: Özel sektör yapsın, denesin, geliştirsin, eksiklerini giderip nihaî hâline getirsin, bireyler kabullenip kullanmaya başlamış olsun biz de devlet süreçleri içerisine alalım. Yani devletlerin kripto para ve Blockchain alanında nötr bir tavır sergileyip atılım göstermemesi ve hâlâ didikleyici bir şekilde sorgulaması oldukça normal bir süreçtir.

Kendi Blockchain Ağımız Olmalı Mı?

Peki, güven olayını anladık ve iş modelimizin buna ihtiyacı var. Yani Blockchain ağına geçiş yapacağız. Bu noktada iş modelimizin süreçleri çok önemli bir rol oynuyor. Burada süreçlerden biraz bahsetmemiz gerekli. İş modelimizi Blockchain ağına kısmî veya tamamen geçirme şansımız var. Eğer iş modelimizin ödeme ve tahsilat süreçlerini Blockchain ağına geçirmek istiyorsak yani yüksek komisyonlar ve kesintiler anlamına gelen "**middle man**" kavramından kurtulmak istiyorsak herhangi bir kripto paraya ve onun Blockchain ağına geçebiliriz. Bu noktada işimizin tahsilat ve ödeme tarafını Blockchain ağına geçirmiş oluyoruz.

Ancak eğer içerik üretiyorsak özgünlüğü damgalamak istiyorsak ya da kullanıcılarımızın servisimiz içerisinde paylaşacağı gelir bazlı modellerimiz varsa (*bu gibi süreçleri çeşitlendirebiliriz, ancak sizi sıkmak istemiyorum*) projemizi tamamen Blockchain ağına geçirmemiz gerekiyor, diyebilirim. Burada süreçleri yönetmemiz ve ilişki ağı kurmamız gerekeceğinden **akıllı sözleşmeler**i devreye almamız gerekiyor. Bir başka deyişle iş modelimizi alıp süreçlerine ayıracağız ve bu süreçleri, ilişkileri göz önünde tutarak akıllı sözleşmelere aktaracağız, ardından sistemi çalışır hâle getireceğiz. Bu noktada **merkezî olmayan bir yapı**ya da kavuşmuş olacağız. Tabii bu Ethereum ağı üzerinde (*EOS, NEO gibi Ethereum türevlerinden biri de olabilir*) çalışacağımız anlamına geliyor.

Ethereum ağı üzerinde bir yapı kurgulamak size zamandan ve başlangıç maliyetlerinden ciddi bir tasarruf sağlayacaktır. Hatta maliyetlerinizi daha da düşürmek için Ethereum dışındaki benzer ağ yapılarını da inceleyebilirsiniz. Önceki bölümden hatırlarsanız Ethereum ağına dâhil olmak için **ETH** bazında ve **Gas** adında komisyonlar ödememiz gerekiyordu. Yani sisteme "**merhaba**" diyebilmek için ETH cüzdanına ve biraz da ETH kripto parasına ihtiyacınız olacak. Ancak gelir modelinizin sonucunda oluşacak tahsilatlarınızı da ETH cinsinden toplayacağınız için süreklilik döngüsüne girmiş olacaksınız.

Ethereum ağı üzerinde kurguladığımız bu modelde sistemimizi tamamen akıllı sözleşmelere aktarıp Ethereum ana ağının üzerinde bir **yan ağ** (*sidechain*) olmuş olduk. Ancak bazı durumlarda işinizi, kullanıcılarınızı ve **know-how** birikiminizi "**başkasının**" ağı üzerinde bırakmak istemeyebilirsiniz.

Bu tarz bir yapı size her şeyinizi "**emanet**" etmişsiniz gibi bir his verebilir. Ayrıca üzerinde koştuğunuz yapı çökme ya da tıkanma yaşayacak olursa sizin sisteminiz de bundan nasibini almış olacaktır. Düşük bir ihtimal olmakla birlikte bir kötü durum senaryosu olarak planlarınız içerisine koymanız gerekli.

Bu noktada zamanınız ve bütçeniz varsa kendi özel Blockchain ağınızı geliştirerek iş modelinizi üzerine kurgulayabilirsiniz.

Kendi ağınıza sahip olmanın bazı avantajları vardır. Mesela, dilediğiniz gibi komisyonlar ve süreçler üzerinde oynayabilirsiniz. Modüler bir şekilde sonraki ihtiyaçlarınız doğrultusunda ağ yapısını değiştirebilirsiniz. Dilerseniz **halka açık**, dilerseniz **kapalı bir yapı** kurgulayabilirsiniz. Hatta ikisini de seçenek olarak sunabilirsiniz ama en güzeli de sizinki gibi birçok projeyi ağınıza dâhil ederek yan gelirler edinebilir ve sektördeki oyun kurucular kervanına katılabilirsiniz. Kendi ağınızı kurmanın dezavantajları da var tabii. Bunların başını çeken, maliyetler ve geliştirme sürecidir. Sonrasında da ağınıza düğüm olacak kullanıcılar bulmanız gerekecek. Yani bir "**topluluk**" sahibi olmalısınız. Bunun için sosyal ağları çok iyi kullanmalı, etkileyici ve ikna edici bir "**whitepaper**" hazırlamalısınız. Tabii devlet eliyle oluşacak olası regülasyonlara da hazırlıklı olmalısınız. Bu aşamada önemli bir nokta var; devletlerin regülasyonları ve yaptırımları (*negatif ya da pozitif yönlü olması fark etmez*) **halka açık ağlar**, kripto paraların işlem gördüğü **borsalar** ve **ICO süreçleri**ni ilgilendirir. Yani kapalı bir Blockchain ağı kurgulayıp sadece sizin kullanıcılarınızın yararlanabileceği bir yapı hayata geçirirseniz bu tarz süreçlerden çekinmenize gerek kalmayacaktır. Benzer şekilde iyi kurgulanmış, süreç yönetimi, bütçe planlaması ve yol haritası açısından ayakları yere basan şirketlerin ICO süreçlerinde de devletlerin yaptırım uyguladıklarını görmüyoruz.

KENDİ KAYNAĞINI ÜRETMEK

3

Günümüzde global ekonominin hem mikro hem de makro düzeyde girdiği çıkmazlar sonucunda ülkelerden başlayarak şirketlere ve bireylere kadar inmiş ekonomik kaynak sıkıntısını aşmak için birtakım yollar deneniyor. Bu yollara yenilikçi bir bakış açısı getirmeyi çok hızlı bir şekilde başaran Blockchain teknolojisi, kişilerin "**madencilik**" ya da "**musluk**" yapıları kullanarak kendi ihtiyaçlarını sıfırdan karşılayabilecekleri imkânlar sunarken; küçük, orta ve büyük ölçek ayrımı yapmaksızın her alandan şirkete "**ICO** (*initial coin offering*)" yaparak kaynak yaratma olanakları sunuyor. Bu güç, kimilerine göre mucize bir kurtuluşu getirirken kimilerine göre de devletlerin para basma gücünü ellerinden alıp otoritelerini zayıflatıp bankaların kredi verme yapılarını tıkayarak piyasaları daha büyük bir çöküntüye sürükleyebilir! *Peki, gerçekte durum ne?* İşin gerçeği, iki görüşün de **doğru** olması! Ancak durum kaotik ya da içinden çıkılmaz bir noktada değil. Bireyler bazında kabul görmüş ve şirketler düzeyinde kabul görmeye başlamış bir yapının belli bir doygunluğa ulaşınca devletler tarafında da kabul edileceğini düşünebiliriz. Bu süreç yavaş ilerleyen bir süreç olduğundan şimdilik kötü durum senaryoları ya da mutlak iyiye ulaşma ütopyalarını bir kenara bırakıp piyasanın yatayda ve dikeyde ilerleyişini izlememiz gerekiyor. Ancak kabul etmeliyiz ki 2009 – 2012 aralığında Blockchain ve kripto para dünyasına giren kesimin neredeyse yüzde 80'i 2017 yılını milyarder olarak bitirmeyi başardı. Evet, kripto paralar (*özellikle Bitcoin*) dünyamıza yeni milyarderler katmayı başardı. Üstelik bu milyarderler büyük girişimciler, prestijli üniversitelerden mezun dâhiler ya da çığır açan teknolojiler üreten kişiler değildi.

Büyük çoğunluğu 20 yaş altındaydı ve inanmışlardı! Evet, Bitcoin'in internetin parası olduğuna inanmışlardı! Bu inançları sayesinde bilgisayarlarının kaynaklarını, harçlıklarından ayırdıkları birikimleri Bitcoin'e yatırıp **musluk** (*faucet*) sitelerini bağımlılık derecesinde takip ederek binlerce Bitcoin'lik cüzdanlara sahip oldular. Bu insanları Bitcoin'e inandıran sebepleri bilme şansımız yok tabii, ama kültleşmiş bilim kurgu filmlerinin, bilgisayar oyunlarının ve fütüristik çizgi roman serilerinin etkisi olmadığını da söyleyemeyiz. Tüm bunlar bir yana, dünyada ardı ardına yaşanan global krizlerin insanları işsiz, evsiz ve çaresiz bıraktığı karanlık bir dönemde Bitcoin gibi bir yapının karşılık bulup kaynak sıkıntısına çare oluşu göz ardı edilmemelidir.

Şimdi gelin, *nedir bu kripto para dediğimiz olgu? Neyi amaçlar? Ne vaat eder?* Yakından bakalım.

Kripto Paralar



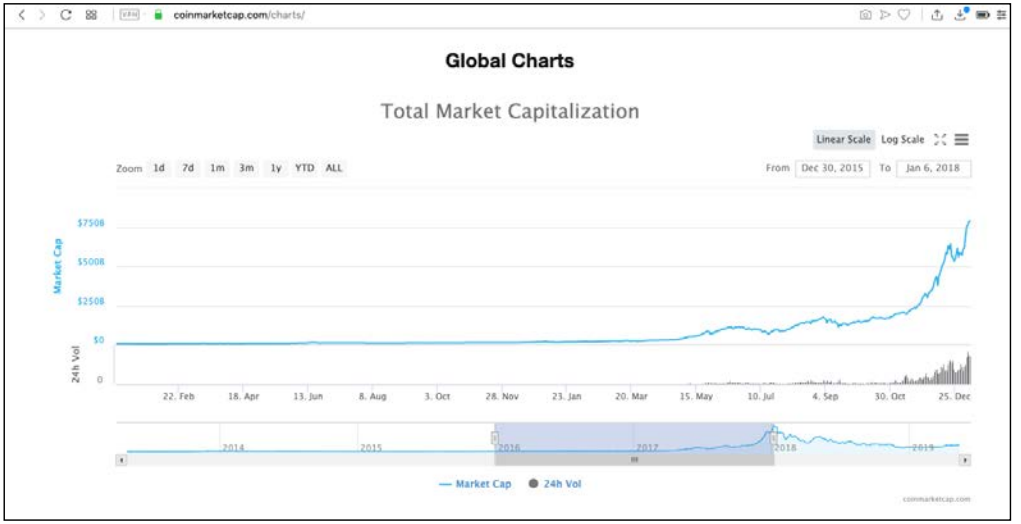
Şekil 1.37: Bitcoin'in hızlı değerlenme süreci, adına alternatif coin dediğimiz birçok yeni kripto para ile tanışmamıza sebep oldu.

2008 yılında, global krizin patlak vermesinden hemen önce, ABD ekonomisinin ve dünyanın en büyük finans kurumlarından birisi olan **Lehman Brothers**'in iflası dengeleri yerinden sarsmıştı. Üstelik bu, gölün durgun sularına atılan taşın çıkarttığı ilk dalgaydı.

Bu iflasın ardından önce yıkılmaz diye tabir edilen bankalar, ardından çok uluslu şirketler ve sonrasında da ülkeler ciddi bir ekonomik krizin avcuna düştü. İşinden olan binlerce kişi yeni bir iş bulamadığından kredilerini ve ipoteklerini ödeyemez hâle gelerek aileleriyle birlikte dipsiz bir kuyunun içerisine düşüverdiler. **Mortgage krizi** de denilen bu süreç insanların bankalara karşı tavır almasına neden oldu. Bu resimde her ne kadar bankalar insanları ve şirketleri batışa sürüklemiş gibi görünse de aslında onlar da batmamak için mudilerini ve kredi dağıttıkları şirketleri dibe çekmeye başlamışlardı. Bu zorlu sürecin ilk aşaması olan Lehman Brothers'in iflasını izleyen süreçte (*aslen sadece iki ay sonra*) ortaya çıkan ve daha önce de belirttiğim gibi kimliği hâlâ gizliliğini koruyan **Satoshi Nakamoto** takma adlı kişi veya grubun yayınladığı "**Bitcoin: Eşten Eşe Elektronik Nakit Ödeme Sistemi**" başlıklı teknik çalışma, her ne kadar bir dizi matematiksel ve teknolojik uygulamayı bizlere sunsa da aslında makalede verilen temel mesaj gayet insanî ve basitti: **aracıların olmadığı bir ekonomi yapısı kurmak!** Bu makalenin yayımlanmasının ardından Bitcoin her ne kadar 3 Ocak 2009 tarihinde hayatımıza girse de ilk kez 22 Mayıs 2010 tarihinde, **Laszlo** takma isimli bir kullanıcı tarafından 2 adet pizza satın almak için "**para**" amaçlı kullanılmıştır. Bu büyük anı onurlandırmak için dünyanın pek çok yerinde 22 Mayıs tarihi **Laszlo'nun Pizza Günü** olarak kutlanmaktadır.

Kripto paraların değeri, diğer her türlü mal, ürün ve parada olduğu gibi talep ve arzın dengelendiği noktada oluşur. Bir kripto paranın değeri, coğrafi olarak uygunluğu, yaygınlığı, kabul edilirliği, yatırımcının ona olan güveni, gerçek hayatta ödeme aracı olabilmesi ve pazarın o anki duyarlılığı ile doğrudan ilişkilidir.

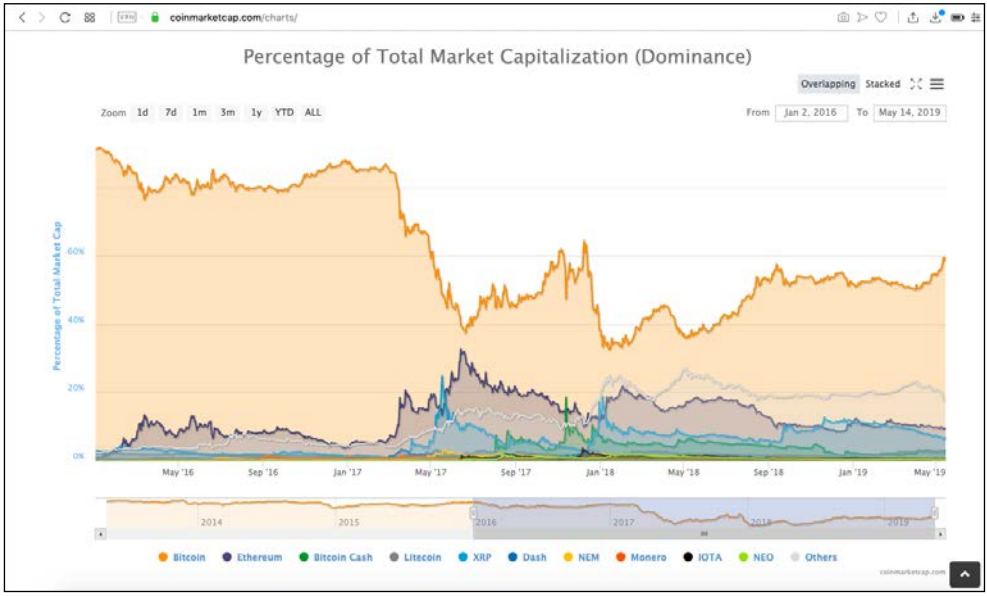
2011 yılından beri kripto para derken Bitcoin'in yanı sıra **altcoin** adını verdiğimiz diğer kripto paralardan da bahsetmeye başladık. Günümüze gelene kadar sayısı 3000'e (*Aslında kripto para kur sayısı 10 bin rakamını aşmıştır. Fakat coinmarketcap.com gibi platformlar ortak bir karar alarak borsalarda işlem görmeyen kripto paraları listelerinden çıkarttılar. Sonuç olarak borsalarda işlem gören kur sayısı 3000'e yakın diyebiliriz.*) yaklaşan altcoin'lerle birlikte kripto para piyasası oldukça büyümüş durumdadır.



Şekil 1.38: 2016 – 2017 yılları arasında Bitcoin ve diğer altcoin'lerin piyasa hacmi 820 Milyar USD seviyesini aşmıştır.

Yukarıdaki grafikte 2016 yılında hızlı bir şekilde artmaya başlayan piyasa hacminin, 2017 yılı sonunda 820 milyar ABD doları seviyesini aştığını görebilirsiniz. Ancak sonrasında 2018 yılı ilk çeyreğinde düşüşe geçerek 300 milyar ABD doları seviyesine kadar gerilemiştir.

2013 yılından itibaren piyasaya bakacak olursak, 2017 yılına kadar Bitcoin egemenliğinde bir hacim görüyoruz. Ancak 2017 sonu ve 2018 ilk çeyreği aralığında piyasadaki Bitcoin egemenliğinin yüzde 45 seviyesinin de altına doğru gerileyerek egemenliği altcoin'lere terk ettiğini söyleyebiliriz. Bu tespitimizi daha net anlayabilmek için aşağıdaki grafiğe bakmamız yeterli.



Şekil 1.39: BTC, ilk kripto para borsası açıldığı günden itibaren piyasayı domine etmeye devam ediyor.

Peki, Bitcoin neden geriliyor? Bir başka şekilde sormak gerekirse yatırımcı neden altcoin'lere yönelmeye başlıyor? Bu sorunun cevabını daha net verebilmek için kripto paraların artılarına ve eksilerine yakından bakmamız gerekiyor.

Kripto Paraların Artıları

Kripto para ağı merkezî değildir; herhangi bir aracı, yönetici, denetleyici yoktur, uçtan uca birbirine bağlı, **gönüllü katılım sağlayan cüzdan**lardan oluşur. Cüzdanlar bir bilgisayarda kullanılabileceği gibi cep telefonu ya da tablet gibi mobil cihazlarda da kullanılabilir. Bağlı tüm bilgisayarlar (*düğümler*), açık kaynak kodlu aynı programı (*kripto para yazılımının "core" uygulaması*) çalıştırırlar. Hepsi tüm işlemleri görür, hepsi tüm işlem geçmişini isterse tutabilir, istedikleri an diğer uçlardan işlem geçmişlerini alabilirler.

Dijital itibari paraların işlemlerinde, güvenilen bir aracıya ihtiyaç duyulurken kripto paralarda aracıya ve güvene ihtiyaç yoktur. Aracılık sisteminin maliyetleri yüksektir ve güvenlik açıklarına gebedir. Kripto paraların omurgasını oluşturan Blockchain teknolojisi güven katmanını üstlenir.

Kripto paralar borç değil, **değer taşıyıcı**dır. Banka hesaplarındaki paralar, bir tür borç senedidir. Bir hesap, bir bankanın müşterisine olan borcunu temsil eder. Kripto paralar bir borcu temsil etmez. Banka ve hükümetlerin, banka hesapları üzerindeki kontrol güçleri, kripto paralarda yoktur.

Devletler para arzıyla ve kısıtlamasıyla bankadaki paranın değerini etkileyecek (*enflasyon ve deflasyon*) mali kararlar alabilirler. Oysa kripto paraların arzı üzerinde banka ve devletlerin etkisi **yoktur**. Sisteme dışarıdan para arzı yapılamaz, dolayısıyla enflasyon oluşmaz. Para arzı, başarılı blok oluşturan madencilere verilen ödüller şeklindedir ya da madenciliği önceden yapılmış şekilde direkt olarak piyasaya arz edilebilirler.

İşlemler anonim olup **takma adlarla** yapılır. İşlemlerin gerçek kişilerle, kuruluşlarla, banka hesaplarıyla bağlantısı yoktur. İşlemler kripto para adresleri arasında gerçekleşir. Kripto para adresleri dijital rumuzlardır ve her tahsilat sonrası **otomatik olarak** değişirler. Böylece hesap adresleri üzerinden de takip yapılması mümkün olmaz. Ancak eski adresler silinemediklerinden (*Blockchain bölümüne bakabilirsiniz*), bu adreslere ödeme yollanması hâlinde tahsil ederek bakiyeyi cüzdanınıza eklerler.

İşlemler **şeffaf**, **hızlı** ve **küresel**dir. 2009 yılındaki ilk Bitcoin arzından bu yana, tüm işlemler, isteyen herkes tarafından görülebilir. Yapılan işlemler, neredeyse anında tüm dünyadaki kripto para ağına dağıtılır, makul bir süre içerisinde de onaylanır.

İtibari fiziksel parada işlemlerin hafızası yoktur. Kripto para işlem hafızası ise küresel hesap defteri olan Blockchain veritabanlarında tutulur. Kripto para kullanacak birisinin o kripto paranın sahibi olup olmadığına daha önceki kayıtlarına bakılarak karar verilir.

İşlemler geri alınamaz. Hiçbir otorite, devlet, kişi, bilgisayar programcısı, hatta sistemi tasarlayanlar dâhil, madencinin biri tarafından onaylanıp diğerlerince de kabul edilmiş ve Blockchain ağına yazılmış bir işlemi değiştiremez ya da geri alamaz.Kripto para kullanmak, cüzdan açmak, madencilik yapmak ya da kendi paranızı geliştirmek için bir otoriteden izin almanız gerekmez. Ayrıca hiç kimse ya da hiçbir kurum veya otorite, işlem yapılmasına engel olamaz.

Sistem güvenlidir. Güvenlik matematiksel olarak güvenilirliği ispatlanmış, **kriptografik dijital imzalama** metotları kullanılarak gerçekleştirilir.

Kötü niyetli kişilerin veriler üzerinde manipülasyon yapması, gizli/açık anahtar şifreleme yöntemi kullanılması sebebiyle mümkün değildir.

Kripto Paraların Eksileri

Kripto paralar Blockchain teknolojisi sayesinde merkezî bir yapı sunmazlar. Bu durum gerekli hâllerde işlemlere müdahale şansı tanımaz. Dolayısıyla yanlış meblağlar da ve/veya hatalı adreslere yapılan ödemeler geri alınamaz.

Kripto paralar tamamen **anonim** yapıda çalışmaktadır. Kullanılan adresler ödeme aldığı an cüzdan tarafından otomatik olarak değiştirilerek kullanıcının anonim kalmasını sağlar. Bu da kimin kime ödeme yaptığının anlaşılamamasına neden olur ki ticarette böylesi bir duruma yer yoktur.

Kripto paralar madencilik ile arz edildiklerinden çok fazla **donanım** ve **kaynak tüketimi**ne neden olurlar. Bu kaynak tüketiminin temel nedeni kullanılan algoritmaların çözülmesindeki zorluk yapılarıdır. Kripto paralarla yapılan işlemler riskleri bakımından değerlendirildiğinde ticarette gerçek bir para olarak kullanılamamaktadır.

Kripto paralarla yapılan ticaretin vergilendirilmesi güç olduğundan bazı devletlerin gözünde **kayıt dışı ekonomi** olarak değerlendirilir. Regülasyonların ve yasal temellerin eksikliği kripto para piyasasının büyümesinde engel teşkil etmektedir.

Kripto paraların artı ve eksi yönlerine baktığımızda resmin sadece tek bir perspektifini anlayabiliyoruz. Oysa fiyat paraların ve bankacılık sisteminin de artılarıyla eksilerine bakmamız gerekiyor. Ancak bu şekilde resmin tamamına bakabilir ve yeni ekonomiyle yeni ticaretin ihtiyaçları doğrultusunda **hibrit bir model** ortaya konabilir. Biraz hayal kuralım: *Mesela, bankalardaki IBAN mantığı gibi cüzdan adresleri olan, böylelikle kimin kime para yolladığının kolayca takip edilebildiği, hatalı gönderimlerin iptal edilebildiği, NFC ya da RF ID tabanlı teknolojiler sayesinde cep telefonlarımızdaki uygulamalarla ödeme alıp ödeme yapabildiğimiz ve bu sayede ticarette rahatça kullanılabilen; belki bir gün itibari paranın yerine geçebilecek, blockchain tabanlı bir paraya hayır diyebilir miyiz?*

Hayatımıza Giren İlk Kripto Para Bitcoin Mi?



Kripto paraların hayatımıza girmesinin ardından bir terimle daha tanıştık: **mikro ödemeler**. Mikro ödemeler sayesinde günümüz fiyat paralarıyla ödeyemeyeceğimiz kadar küçük değerleri kripto paralar ile ödeyebiliyoruz. Bu tarz ödemeleri, mesela bahşiş, teşvik ya da bağış yaparken kullanabiliriz. Bir örnek vermek gerekirse internette bir makaleyi çok beğendiğinizde o makalenin yazarına bir sonraki makalelerinde şevk vermesi açısından küçük teşvik ödemeleri yapmak isteyebilirsiniz. Ya da bir görsel ararken karşınıza çıkan çok hoş bir fotoğraf için çeken kişiye minnetinizi küçük bir teşekkür ödemesiyle yapabilirsiniz. Mikro ödemelerin kullanım alanlarına çok daha farklı örneklemeler yapılabilir.

Bu tarz ödemeleri fiyat kurlarla yapamıyoruz, çünkü günümüz fiyat kurları 100 eşit parçadan oluşurlar. Bu parçaların her birine mesela Türk Lirası için **kuruş** adını veriyoruz. Bugün bir blog yazarına yazısını beğendiğinizi göstermek için 3 kuruş para göndermek isterseniz, bu miktarı göndermek için miktarın katbekatı işlem ücreti ödemek zorunda kalırsınız. Bu durum tabii ki aracılar ve piyasa yapıcıların komisyon oranlarından hariçtir. Onları da devreye alırsak gönderim ücretiniz oldukça yükselecektir.

Kripto paralarda ise durum farklıdır. **1 kripto para** (*genellikle*) **100 milyon eşit parça**dan oluşur ve güzel tarafı şudur ki; her bir parçasını **tekil** olarak harcayabilirsiniz. Bu noktada kripto para hakkında bilgili olan okurların aklına **Bitcoin** ve **Litecoin** gibi kripto paraların gönderim ücretleri gelebilir.

Evet, bazı durumlarda Bitcoin'in 100 milyonda 1'ini harcamak size çok maliyetli olabilir. Ancak, bunun için bir protokol oluşturursanız, bu protokole dâhil olan madenciler gönderimlerinizi işlem ücretinden muaf olarak onaylayabilir. Hatta mikro ödeme gerektiren uygulamaları, işlem ücreti olmayan ağlardan birinde (*IOTA ağı gibi*) de yapabilirsiniz.

Burada önemli olan nokta, mikro ödeme mantığının doğuş anı. Tabii ki bunun kripto paralar ve Blockchain teknolojisiyle doğduğunu söyleyeceğimi bekliyorsunuz ama maalesef öyle değil. Ama şunu söyleyebilirim ki yaptığım araştırmaların bazıları bana kripto para öncesi mikro ödeme çalışmalarının Bitcoin'e ilham veren konulardan biri olduğunu gösterdi. *Meraklandınız mı? O hâlde bakalım ilk kripto paramız Bitcoin miymiş?*

90'lı yılların bilgisayar sektörü için unutulmaz zamanlar olduğu tartışılmaz. 90'larda günümüz internet dünyasının temelleri atılıyor, **UNIX** kullanıcı lisansları ucuzluyor, UNIX benzeri bir işletim sistemi olan **Linux** doğuyor, Microsoft her bilgisayara girmenin yolunu arıyor ve tabii ki **Marc Andreessen, Mosaic** adlı web tarayıcısını kullanımımıza sunuyor. Gerçekten de Mosaic, internetin her eve girmesinin önünü açan en büyük hamledir benim gözümde.

World Wide Web yaygınlaşmaya çalışırken bir kısım eleştirmenler geçici bir moda olduğunu yazıp çiziyorlardı ama internette görsel olarak bir şeyler paylaşmak ve Mosaic ile bu görsel içerikleri görüntüleyebilmek herkes için efsanevi bir gelişmeydi. Tabii bunun sonucunun reklam verenler için yeni mecralar olduğunun keşfedilmesi çok sürmedi. Bir anda basılı yayın kuruluşları bu alana balıklama atlayıverdiler. Hâlihazırdaki **reklam verenlerinin sayısını** ve **bütçesini** artırmanın en dâhiyane ve pratik yoluydu bu. Ancak bu gelir dalgası kısa sürdü. Çünkü, reklam verenler için yeni gözler gerekiyordu ve web içerikleri insanların üzerinde zaman harcamaları için yeterince zengin değildi. Kısa sürede reklam verenlerden daha fazla kazanç elde etmek için ziyaretçi sayılarının artması gerektiği anlaşıldı. Ziyaretçi sayısını artırmanın yolu da zengin ve bol içerik hazırlamaktan geçiyordu. Buraya kadar her şey çözülmüş görünüyordu, bir şey hariç: **İçeriklerin hazırlanması!**

İçeriklerin hazırlanması **emek** ve emek de **para** demekti. O dönem Tim Berners-Lee ve Andreessen tamamen **World Wide Web Consortium** (*W3C*) üzerine odaklanmış ve web içeriklerini kullanıcılara ürettirmenin yollarını arıyordu.

Akıllarına gelen çok parlak bir fikir vardı; **teşvik etmek!** Evet, içeriği hazırlayanı, içeriği hazırlaması için bir ödülle ikna etmek. Bunu yapmak için en iyi ödül seçeneğinin para olduğu kanısına vardılar. *Ama bunu nasıl yapacaklardı?* Bankalar 1 USD altı işlemeleri ne yazık ki kabul etmiyor üstelik çok yüksek komisyon oranlarıyla muhatap olunuyordu. Berners-Lee W3C üzerinde "**mikro ödemeler**" yapabilecekleri bir yapı inşa etmek için kolları sıvadı. İlk etapta hedefleri, içerik sağlayan internet kullanıcılarına teşvik amaçlı küçük ödemeler yapabilmekti.

Fikir basitti, web sayfasına küçük bir ödeme için ihtiyaç duyulan bilgilerin yerleştirilmesi ve bu bilgilerin de bankalar ya da yatırımcılar tarafından geliştirilen bir tür "**elektronik cüzdan**" sistemini harekete geçirmesini sağlamaktı. Ancak bu fikir uygulama aşamasında tıkandı. Elektronik cüzdanların işlenmesi ve para transferlerinin yapılması tarafında banka ayağına güvenmişlerdi. Fakat banka tarafındaki düzenlemelerin karmaşıklığı yüzünden hiç hayata geçemedi. Bunu çözmek için ihtiyaç duydukları kaynak da Illinois Üniversitesi tarafından karşılanmadı. Sonuç olarak rafa kaldırıldı.

Bu fikir gerek **çalışma mantığı** gerekse **çalışma amacı** tarafından Bitcoin ile örtüşüyor. Bitcoin'in sunduğu özelliklerin başında Berners-Lee'nin 90'larda W3C üzerinde yapmaya çalıştığı **mikro ödeme yapısı** geliyor. Yani birkaç eksik adımı saymazsak kripto paranın temellerini atmışlar diyebiliriz. En azından Satoshi'ye ilhamı vermiş olduklarını düşünüyorum. Hatta 2013 yılında Berners-Lee, **W3C Micropayments Markup Working Group**'un 90'larda yaptığı bu projeyi tekrar canlandırmaya çalıştı. Bir röportajında şöyle diyor: "*Mikro ödeme protokollerine yeniden yoğunlaşıyoruz. Bu, web'i çok farklı bir yer hâline getirmeyi sağlayabilir. İyi bir makale veya şarkı yazan birisine ödeme yapılabilmesi, bir şeyler yazan ya da müzik yapan çok daha fazla kişiyi destekleyecektir.*" Andreessen ise 2009 yılında Bitcoin ile tanıştığında, bu tarz bir yapının yakın gelecekte çok daha başarılı sistemler yapılmasına öncülük edeceğini düşündüğünü söyleyerek ekliyor: "*Eğer bir zaman makinem olsaydı ve 1993 yılına geri dönebilseydim kesinlikle yapacağım şeylerden biri Bitcoin ya da benzeri bir kripto para yazılımı yapmak olurdu.*"



Şekil 1.41: Bitcoin öncesi kripto para örneklerinden bir tanesi de WoW Token ya da Blizard'ın WoW oyununda geçen adıyla Gold'dur. Gold, oyun içi satın almaların dışında, oyuncuların reel hayatta birbirleri arasında yaptıkları alışverişlere de ödeme aracı olmuştur.

Evet, gördüğünüz gibi Bitcoin gibi yapılar aynı amaçlar doğrultusunda geçmişte denenmiş ancak farklı nedenlerden dolayı hayata geçememiş. Fakat günümüzde popülerleşen bir başka yapı olan "**stable coin**" (*değeri sabitlenmiş, madenciliği olmayan ve karşılık esasına göre çalışan bir kripto para türü*) mantığında yapılar site içi satın alma işlemlerinde kullanılmak üzere geliştirilmiş "**jeton**" mantığından hiç farklı değildir. Parayla "jeton" alıp bu jetonları site içi servislerde harcama mantığı 90'lı yıllarda oldukça popülerdi. Başka bir erken dönem Bitcoin uygulaması da dünyayı kasıp kavuran, **Blizard** imzalı **World of Warcraft** adlı oyunda kullanılan "**WoW Token**"lerdir. Oyun içerisinde **Gold** olarak adlandırılan bu jetonların amacı oyun içi satın almalar ve Blizard'ın sunucularında oyun oynama süresi satın almayı sağlamakken oyuncuların gerçek hayatta, hesaplarındaki Gold'ları reel alışverişlerde kullanmaları Bitcoin öncesi kripto para uygulamalarına güzel bir örnek teşkil eder. Bu tarz örnekler çoğaltılabilir tabii.

Kripto Paraları "Değer"lendirmek



Şekil 1.42: Kripto paraların değerini belirleyen en önemli etken üretim maliyetleridir. Fotoğrafta bir Bitcoin madeni görülüyor. Böylesi bir madenin maliyeti elbette ki arz edilecek Bitcoin'lerin satış fiyatına kaldıraç etkisi yapacaktır.

2016 ile 2018 yılları arasında kripto para piyasası öyle büyüdü ki hemen herkes Bitcoin ve diğer altcoinlere yatırım yapmak için yarışa girdiler. Hatta 2017 yılı için **milyarderler yılı** dersek yanlış olmaz. Çünkü, sadece birkaç ay içerisinde yüzde 7000'lere varan kâr marjlarıyla kripto para yatırımcılarının milyarderler kulübüne giriş yaptıklarını gördük. Üstelik bu milyarderlerin büyük bir kısmı 20'li yaşlarında gençlerdi. Bu durum tüm dünyada Bitcoin çılgınlığının fitilini ateşlemiş oldu. Bitcoin fiyatı 2017 Kasım ayında 20 bin USD değerine ulaştı ve bir anda aşağıya bir düşüş başladı. **Hiç bitmeyecekmiş gibi gelen bir düşüş!**

Düşüş neredeyse 6 ay boyunca devam etti. Fiyat, Bitcoin'in arz maliyet rakamına ulaşınca dondu ve boğa piyasası yerini ayı piyasasına bıraktı.

Birçok ekonomist kripto paraların **balon** olduğunu savunmaya başladı. Verilen örnekler yatırımcıların unutmaya çalıştığı "**.com balonu**" dönemine atıfta bulunuyordu. Özellikle, 15 bin USD ve üzerinde değerlerden Bitcoin almış olanlar şok ve derin bir depresyon hâline girdiler. *Hemen her kafada aynı soru işareti belirdi: Bitcoin fiyat değeri sıfırı görür mü?*

Tüm bunlar yaşanırken bir kesim vardı ki Bitcoin'in sert bir şekilde değer kaybetmesinden ve ayı piyasasına girmesinden hiç endişe duymuyordu. Bunun nedeni, paranın evrimine olan inançları ve kripto paraların değer sahibi olmasındaki etkenleri iyi biliyor olmalarıydı. Şimdi gelin sizlerle bu rahatlatıcı bilgiyi paylaşayım.

Kripto paraların değeri, diğer her türlü mal, ürün ve para kurunda olduğu gibi, talep ile arzın dengelendiği noktada oluşur. Bir kripto paranın değeri, coğrafi olarak uygunluğu, yaygınlığı, kabul edilirliği, yatırımcının ona olan güveni, gerçek hayatta ödeme aracı olabilmesi ve pazarın o anki duyarlılığı ile doğrudan ilişkilidir. Ancak yadsıyamayacağımız bir başka gerçeklik de madenciliği olan kripto paraların **arz maliyeti**dir.

Bitcoin, Litecoin, Ethereum ve *Dogecoin* gibi arzı madencilikle yapılan kripto paralarda madenciler donanım, elektrik ve internet harcamaları sonucunda blok çözerler ve bu blok içerisindeki ödül parayı arz etmiş olurlar. Bu temel harcama, ağdaki madenci sayısı artışıyla doğru orantılı şekilde büyür. Çünkü, madenci sayısı arttıkça ağın **hashrate değeri** yükselir. Bitcoin'in USD karşılığı değeri arttıkça, ağzı sulanan girişimciler hemen madenciliğe soyunduğundan ağdaki madenci sayısı artmaya devam eder ve ağın hashrate değeri daha da artar. Bu kısır döngü, madencilerin kazançlarının maliyetlerinden düşük hâle geldiği doyma noktasına ulaşıncaya kadar devam eder. Doyma noktasına ulaşmaktaki temel kurguyu Blockchain ağının kendisi yaratır.

Ağdaki madenci sayısı arttıkça Blockchain ağının kurgusundaki adilane dağıtım yapısı, hemen blok çözümlemek için yapılan hesaplamalarda ihtiyaç duyulan zorluk derecesini artırır. Yani madenciler daha fazla sistem kaynağı harcamak zorunda kalır. Bu da madencilerin temel harcamalarının artması demektir. Bu değişken temel harcamalar, madenciliği yapılan kripto paranın piyasa değerindeki **taban fiyatı** oluşturur. Kripto paranın piyasa fiyatı, bu taban fiyata ulaşırsa madenciler kâr edemez hâle gelirler. Eğer arz maliyetinin altında bir fiyat oluşursa o zaman madenciler zarar edeceğinden madenciliği sonlandırırlar. Yani ağdan ayrılırlar.

Bu senaryo, kripto paranın sonu gibi gelebilir; sonuçta madenciler olmazsa hem yeni arz olmayacak hem de işlemler onaylanmayacağından para trafiği duracaktır. Yani ağ çökecektir. Ancak Blockchain ağı öyle adilane kurulmuştur ki ağdaki madenciler azaldıkça zorluk derecesi otomatik olarak düşmeye başlar.

Zorluk derecesi düştükçe ağda kalan madenciler daha çabuk blok çözecek ve birim zamanda daha çok **ödül coin** kazanacaklardır. İşlemleri onaylayacak daha az madenci olacağından işlem onay ücretleri de madencileri doyuracak, hatta kârlılığa geçirecek duruma gelir. Buna ek olarak zorluk derecesinin düşmesi, madencinin arz için harcaması gereken temel maliyeti de düşürür. Daha az işlemci gücü dolayısıyla daha az elektrik harcanmış olur. Gelir artarken gider de düşer ve kârlılık maksimum değere çıkar. Bu noktada madencilik yeniden kârlı bir iş olur ve ağ kendi kendini kurtarır.

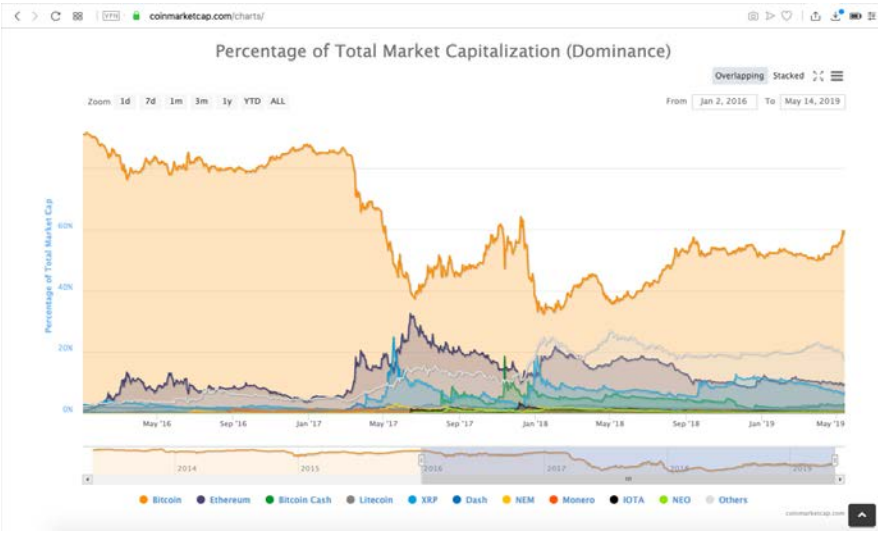
Kripto paraların arz maliyetleri aynı zamanda piyasaya sürülebilecek en düşük fiyat olarak karşımıza çıkar. Bir örnek vermek gerekirse bir madencinin bir adet bloğu çözme maliyeti 45 bin USD olsun. Bir blok çözüldüğünde içerisindeki ödül miktarı 12.5 BTC'dir (*şimdilik*). Madenci 1 adet BTC maliyetini hesaplarken 45.000 / 12.5 = 3600 USD işlemini yapar ve basit usulde 1 BTC'nin maliyetini bulmuş olur. Kötü durum senaryosunda madenciler belli bir miktardaki BTC'yi maliyet fiyatından satabilirler. Bu nedenle fiyatın taban değeri örneğimizdeki 3600 USD'nin altına inemez. Ancak bu hep böyle sürmez. Belli bir blok sayısına ulaşıldığında otomatik olarak blok başı ödül sayısı yarıya iner. Buna **halving** denir. Halving gerçekleştiğinde zorluk derecesi aynı kaldığından maliyet düşmez ancak ödül miktarı yarıya indiğinden 1 BTC'nin maliyeti otomatik olarak ikiye katlanır. Örneğimizdeki madencinin blok çözme maliyeti 45 bin USD olmaya devam ederken blok başı kazancı 6.25 BTC miktarına düşeceğinden, artık 1 BTC'yi 7200 USD harcayarak mal etmiş olacak. Bu da halving sonrası piyasadaki fiyatın ikiye katlanacağını gösterir.

Bu teknik bilgilere dayanarak her halving sürecinde BTC değerinin ikiye katlanacağını söyleyebiliriz. Bu bilgileri doğru analiz ettiğimizde BTC fiyatının öngörüsünü yapmak hiç zor olmaz.

Peki, madem durum bu kadar ön görülebilir, neden BTC fiyatının 20 bin USD'den kırılarak 3500 USD'lere kadar gerilemesi öngörülemedi? Aslında bunun cevabı basit. Eğer **servet hastalığı** denen psikolojik illetin pençesine düşmediyseniz, kâr marjını yeterli bulduğunuz an elinizdeki emtiayı satarak kârınızı toplamanız gerekir. 2017 yılı sonunda yaşanan BTC fiyat kırılımının temel nedeni buydu. Siz Bitcoin adını değeri 10 bin USD'nin üzerinde seyrettiği 2017 yılında duymuş olabilirsiniz, ama unutmayın ki Bitcoin ile 10 bin adeti 1 USD değere satılırken tanışan kişiler de var. Üstelik bu kişilerin sayısı az olmakla birlikte ellerindeki BTC hacmi, satış anında piyasayı negatif etkileyecek kadar yüksek düzeydedir.

Bu tip kullanıcılara "**Balina**" diyoruz. Balinalar piyasalardaki kripto paraların fiyatlarını manipüle edebilecek kadar yüksek miktarlarda kripto para adetlerine sahiptirler.

Buraya kadar anlattıklarım hâlihazırda bir değere oturmuş, piyasası, destekçi topluluğu ve madencileri olan kripto paralar için işleyebilir. Ancak bu tarz hesaplamalara dalıp ekonominin reel gerçekliklerini de görmezden gelmeyin. Basında ve internette birçok kişinin BTC'nin değeri üzerinde spekülatif görüşler bildirdiğini, ütopik hedefler koyduğunu görürsünüz. Bunlar gibi BTC üzerinde negatif görüş ve fiyatta negatif öngörülerde bulunan kişilere de rastlarsınız. Tüm bu görüşlerin özünde yere sağlam basan noktalar vardır, ancak demeç sahipleri bu noktaları eğip bükerek piyasa fiyatını yukarı ya da aşağı yönde etkilemeyi amaçlarlar. Şunu unutmayın ki ticarette temel kural malı ucuza alıp pahalıya satabilmektir. Yani bu tarz spekülatif kişilerin hedefleri BTC'yi ucuza alıp pahalıya satarak kâr marjlarını yükseltmektir.



Şekil 1.43: Grafikten görüldüğü gibi piyasa değerinin 820 milyar USD'yi aştığı 2017 yılının son çeyreğinde Bitcoin'in piyasadaki hacmi sadece yüzde 55.

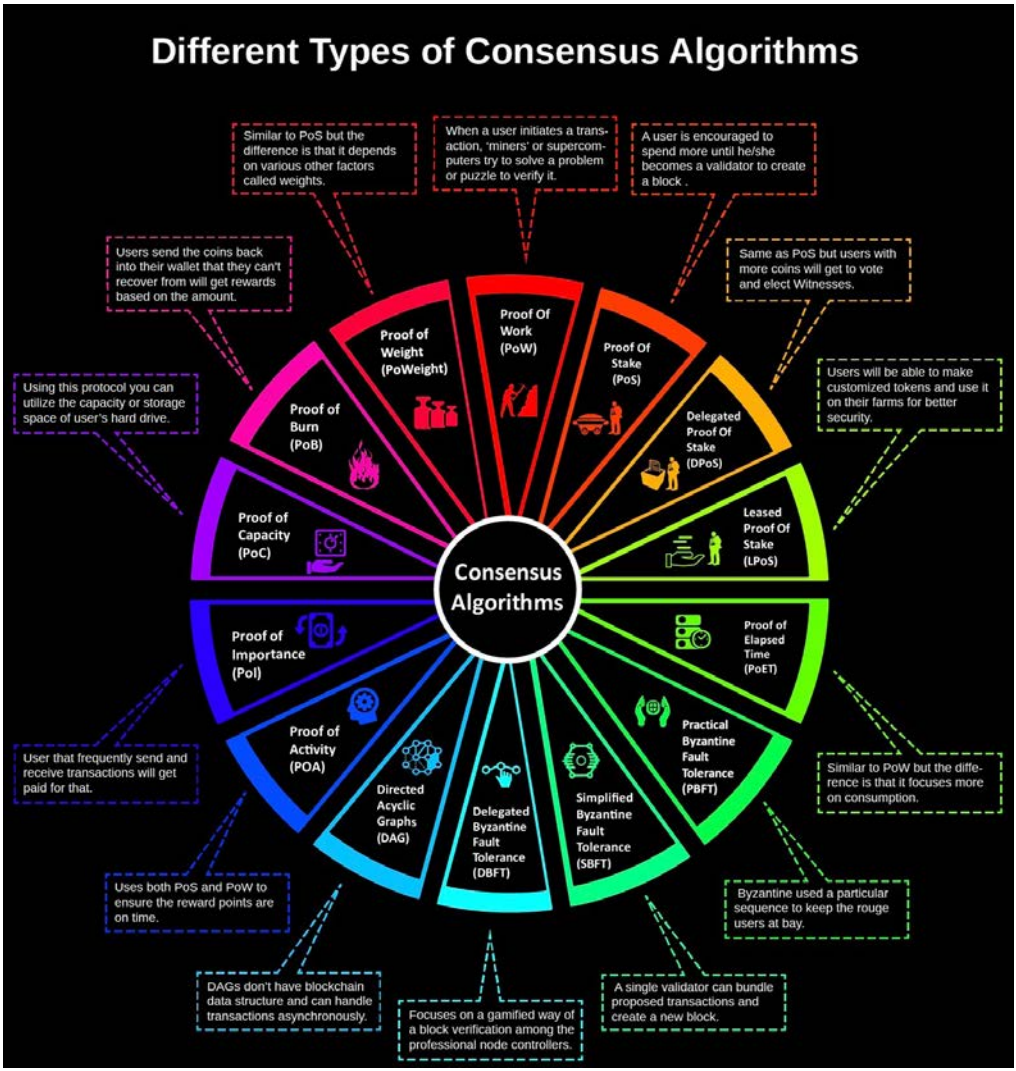
Bu tür spekülatif söylemleri değerlendirirken bazı temel ekonomik gerçeklerle birlikte ele almalısınız. Örneğin 1 BTC'nin 100.000 USD değere sahip olacağını savunan çok fazla kişi var. Bu söylemi değerlendirirken BTC'nin toplam arz miktarını hatırlamamız gerekiyor. Bu sayı 21 milyon adettir. 1 BTC 100 bin USD değere ulaşacak olursa 21 milyon adet BTC 2.1 trilyon USD değeri taşıyor,

demektir. Bu rakam ne kadar gerçekçi; bunu irdelemeniz gerekiyor. Dünyada fiyat para miktarı iki tipe ayrılıyor; **gerçekten basılı olan para miktarı** ve **bankacılık sisteminde hesap karşılıklarına yazılan ama basılı karşılığı olmayan para miktarı**. Bu iki değeri birbirine karıştırmamanız önemli. Yani BTC'nin değeri bir gün 2.1 trilyon USD değere gelecekse karşılığı olan 2.1 trilyon USD'nin kripto para borsalarında reel olarak toplanıp alıcı ve satıcılar arasında değiş-tokuş edilmesi gerekir. Bunun da mümkün olması için öncelikle dolaşımda o miktarda reel ve basılmış USD olması gerekir. *Peki, dünyada basılı USD miktarı ne kadar?* İnternette kısa bir araştırmayla hemen öğrenebilirsiniz ancak sizi çok yormayayım. 52 trilyon USD basılı olarak piyasada bulunuyor. Ancak bu miktarın tamamı dolaşımda değil. Yaklaşık 35 trilyon USD kadarı ülkelerin merkez bankalarında karşılık olarak tutuluyor. *Yani dolaşımda 17 trilyon USD bulunuyor. Şimdi şunu düşünerek risk almanız gerek, dolaşımdaki 17 trilyon USD'nin 2.1 trilyon kadarı BTC ardında toplanabilir mi?* Dolaşımdaki USD'nin yüzde 13'üne tekabül eden bir tutardan bahsediyoruz. Bu size gerçekleşebilecek bir değer gibi gelebilir. Ne de olsa 2017'de 820 milyar USD değere ulaşmıştı piyasa. Ancak bu yanlış bir kıyaslama olur. 2017 yılındaki 820 Milyar USD'lik karşılığın sadece yüzde 55'i Bitcoin'e aitti. Geri kalan yüzde 45'lik kısım altcoin'lerin topladığı para karşılığıydı. Şimdi gözünüzü kapatın ve tüm değerlemeleri bu donelere göre yapın. O zaman daha net sonuçlara ulaşacaksınız.

Bunlar işin tamamen teknik kısmı. Bir de değere etki eden fayda kısmı var. *Yani bu kripto para hangi amaca hizmet ediyor? Reel dünyada hangi sorunu ya da yükü hafifletiyor? Alışverişlerde geçerlilik oranı ne? Fiyat paraya dönüş hızı ne?* İşte bu gibi soruların yanıtları bize kripto paranın fayda eğrisini veriyor. Hiçbir işe yaramayan ya da topluluğa sahip olmayan, alışverişlerde geçerliliği olmayan bir kripto paranın arz maliyeti ne olursa olsun değer sahibi olmasını bekleyemezsiniz. Bitcoin özelinde düşünecek olursak onun ilk kripto para olduğunu ve bu yolun kapısını onun açtığını göz önüne almalısınız.

Evet, artık kripto para neye göre değerleniyor ya da spekülatif değer hedeflerinin sağlaması nasıl yapılıyor biliyorsunuz. Özetlemek gerekirse kripto paranın toplam arz miktarını, madencilik maliyetlerini, dolaşımdaki paranın ne kadarını karşılık olarak toplayabileceğini, nasıl bir faydaya sahip olduğunu ve fiyat paraya dönüş hızını değerlendirerek karar vermelisiniz. Tabii balina statülü yatırımcıları ve ellerindeki yüksek hacimlere varan kripto para miktarlarını göz ardı etmeyin. Bir anda piyasayı "**pump**" ve ardından "**dump**" yaparak resmi tersine çevirebilirler.

Kripto Paranın Anatomisi: Algoritmalar, Ödül Dağıtım Yöntemleri ve Madencilik Tipleri



Şekil 1.44: Kitabımda yer veremeyeceğim kadar çok mutabakat yöntemi türemiş durumda. Birçoğuna yukarıdaki şekilden ulaşabilirsiniz. Piyasada genel geçer bir hâl almış olanları ise detaylarıyla bu bölümde bulabilirsiniz.

Piyasada kripto paraların sayısı arttıkça yatırımcılar hangi kripto paraya yatırım yapacaklarına karar veremez oldular. Kripto para geliştiricileri ise kendi paralarını öne çıkartabilmek ve yatırımcıların ilgisini çekebilmek için, birçok teknik tanım üretmeye başladılar. Böylece titizlikle çalışıp fark yaratacak teknolojiler ürettiklerine dair genel kanı oluşturmaya çalıştılar. İşte bu dönemde karşımıza çıkan birçok mutabakat ve algoritma tipiyle tanıştık. Bazıları sektörde kendilerine yer buldu ve başka kripto paralar tarafından da kullanılmaya başladı. Bazıları ise unutuldu gitti. Şimdi gelin piyasada kabul görmüş ve terimleşmiş yapı taşlarına yakından bakalım.

Mutabakat Yapısı ve Süreci

Blockchain ağ yapısını anlattığım bölümde, verinin orijinaliyle birebir eş olan bir kopyasının ağ yapısı üzerindeki tüm makinelerde (*düğümlerde*) barındırıldığını söylemiştim. Bunun sağlanabilmesi için, ağ üzerindeki tüm düğümlerle **mutabakat** (*consensus*) yapılır. Mutabakat mantığı ağdaki verinin güvenliğinden, bloğun çözümü sırasında yapılan işin denetlenmesine ve kazanılan ödülün işleme konulmasına kadar birçok kritik göreve önderlik eder.

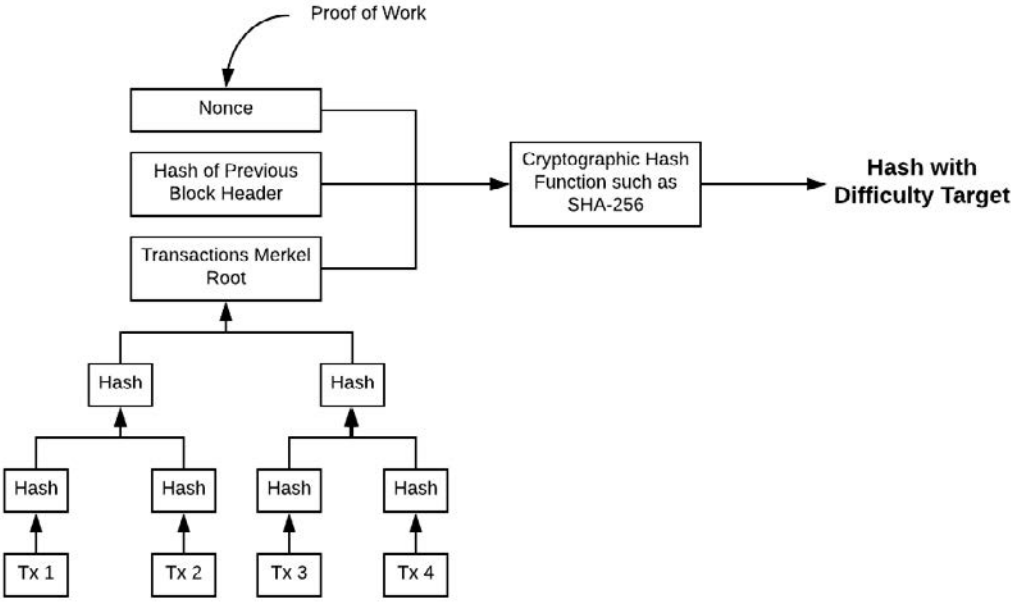
Blockchain ağlarında toplanan ve işlenen tüm verilerin, **blok** adı verdiğimiz ve zinciri oluşturan her bir halkanın içerisinde tutulduğunu söylemiştim. Blockchain ağlarında kırılmaz olduğunu iddia ettiğimiz "**güvenlik**" yapısının amacı, blokların içerdiği bilgilerin dış dünyadan saklanmasını değil, her bir bloğun içerdiği bilgilerin fark edilmeden değiştirilemeyeceğini sağlamaktır. Bunun sağlanması için de **hash sıkıştırma** algoritmalarından yararlanıyoruz demiştik. Sıkıştırılan özet veri, ağ içerisinde oluşturulduğu an, kriptografik özetleme (*Cryptographic Hashing*) ve zaman bilgisi kullanılarak damgalanır. Bu sayede verinin hiç değişmediğinden emin olunurken verinin özgün hâlinin kim tarafından ve ilk olarak ne zaman oluşturulduğu takip edilebilir.

Blok, temel olarak iki parçadan oluşur; blok içerisindeki veriler ve blok içerisindeki veri bütünlüğünü kontrol etmek amaçlı kullanılan **üst bilgi/başlık** (*Block Header*). Blok başlığı ise; bir önceki bloğa (*üst blok*) ait özet (*hash*) değeri, blok içerisindeki verilere ait Merkle kök değeri (*bunu kısaca tüm verilerden tek bir özetleme verisine erişme şekli olarak düşünebiliriz demiştik*) ve zaman bilgisi.

Blok başlığı içerisindeki bilgilerin, toplu bir şekilde, güvenli özetleme algoritmasından geçirilmesi ile o bloğa ait olan **özetleme bilgisi**ne (*block hash*) ulaşılır. Fark edebileceğiniz gibi her blok kendisinden önceki bloğa ait özetleme bilgisini içermektedir. Bu bilgiyi içeren bloğun özeti ise bir sonraki blok için kullanılacak özetleme bilgisini elde etmekte kullanılmaktadır.

Bu yapıda, kötü niyetli birisinin, Blockchain ağı üzerinde hedef aldığı bir blok içeriğini değiştirebilmesi için hem hedef bloğu hem de ondan sonra gelen tüm blokları değiştirmesi gerekmektedir. Blok üretiminin sürekliliği (*saldırgan değişim yaparken Blockchain'e yeni blokların katılıyor olması*) ve blok üretim yaklaşımlarının yapılarından dolayı, bu senaryo teorik olarak gerçekleştirilebilecek olsa da pratikte gerçekleştirilmesi normal koşullarda mümkün değildir. Dağıtık bir mimaride bulunan her bir düğüm üzerindeki blokların **eş bir yapıda** olabilmesi için, bu düğümlerin sisteme eklenmek istenen her yeni blok için bir mutabakat yaklaşımı sergilemesi gerekir. Blockchain platformları bu konuda farklı çözümler sunar. Mutabakat tipleri içerisinde en yaygın ve kabul görmüş olanları **Proof of Work, Proof of Stake** ve **PBFT** (*Practical Byzantine Fault Tolerance*) şeklinde karşımıza çıkmaktadır.

Emeğin İspatı: Proof Of Work



Şekil 1.45: Proof of Work, Bitcoin başta olmak üzere birçok popüler kripto paranın blok onay ve madencilik operasyonlarını yöneten mutabakat yapısı olarak karşımıza çıkıyor. Benim gözümde en mantıklı yapı

Bitcoin ve Bitcoin'den klonlanan çoğu altcoin'in Blockcahin ağ mutabakat yapısı, **Proof of Work**, yani **yapılan işin ispatı** yönteminden oluşur diyebilirim.

Bu yapıda sistemin bir blok yapısı hazırlanıp ilgili Blockcahin ağına eklenmesinin yönetimi için, çözülmesi zor ama çözümün doğruluğunun kolay kontrol edildiği bir problem üzerinden ilerlenir. Bu konuda en çok kullanılan problem türü, hazırlanan bloğa ait **özetleme** (*hash*) değerinin belirli bir yapıya (*tanımlanmış bir değer aralığı içerisinde olma, belirli bir karakter dizisi ile başlama gibi*) uymasıdır.

Özetleme (*hash*) fonksiyonları yapı itibari ile tek yönlü olduklarından ve çıktıları tahmin edilemediğinden, uygun bir değer üretilmesi için oldukça fazla sayıda deneme yapılması gerekmektedir. Örneğin, Bitcoin yapısında Nisan 2017 itibari ile saniyede ortalama **$3{,}5x10^{15}$** hash işlemi yapılmakta ve ortalama olarak 10 dakikada bir Proof of Work yapısına uygun blok üretilebilmektedir. Ancak paylaşılan bir özetleme (*hash*) değerinin kontrolü için sadece ilgili blok içerisinde özetleme (*hash*) değerinin bir defa hesaplanması yeterlidir. Şu andaki en popüler Blockchain platformu olan Bitcoin üzerinde bu mutabakat yaklaşımı kullanılmakta ve ilgili süreç, **madencilik** (*mining*) olarak adlandırılmaktadır.

Bu süreci, aşamalarına ayırmak istersek;

- Yeni blok içerisinde yer alması istenen işlemler/veriler seçilir.
- Bu işlemler/veriler kullanılarak Merkle ağacı yapısı ve Merkle kök değeri oluşturulur.
- Merkle kök değeri, bir önceki bloğun özetleme değeri, zaman bilgisi ve ardışık olarak artan bir sayaç olarak tanımlanabilecek "**nonce**" değeri kullanılarak blok başlığı oluşturulur.
- Blok başlığı özetlenerek (*hashing*) uygun bir değer (*belirli bir karakter kümesi ile mi başlıyor gibi*) oluşup oluşmadığı kontrol edilir.
- Eğer uygun bir blok özetleme değeri oluştu ise yeni blok başarılı bir şekilde oluşturulmuş demektir, bu bilgi ağ üzerindeki tüm makineler ile paylaşılır.
- Eğer uygun bir blok özetleme değeri oluşmadı ise nonce değeri arttırılarak uygun özetleme değeri yaratılmaya çalışılır. Nonce değeri limitine geldiğinde hâlâ geçerli bir blok oluşturulamadı ise (*yani geçerli bir özetleme değeri oluşturulamadı ise*) bu durumda blok başlığını oluşturan diğer değerlerde (*blok içerisinde yer alacak işlemler kümesinin sırası, içeriği gibi*) güncelleme yapılır ve akış tekrar baştan ele alınır.

Başka bir popüler Blockchain platformu olan Ethereum da bu mutabakat tipini kullanır. Proof of Work yaklaşımı, yapısındaki işlemin özel yapısı ve yüksek frekansta tekrarlanma ihtiyacından dolayı yüksek enerji tüketimi ve özel donanım gereksinimlerine sahiptir. Bu durum, çalıştığı ağ yapısının bu gereksinimlerin **daha elverişli** olduğu (*ucuz elektrik, düşük maliyetli donanım üretebilme yetkinliği*) ortamlara yönlendirilmesine ve dağıtık, merkezî olmayan ağ yapısı özelliğinin zamanla belirli ölçüde kaybolmasına neden olabilir. Hem kullanılan problemlerin farklılaştırılması hem de enerji tüketimi çıktılarının farklı şekillerde değerlendirilmesi gibi yaklaşımlarla yoğun enerji kullanımı sorunu çözülmeye çalışılıyor.

Sahipliğin İspatı: Proof Of Stake



Şekil 1.46: Proof of Stake, madenciliği olmayan kripto paralarda kullanılan bir mutabakat yöntemi. Cüzdanında en çok kripto para miktarını barındıran düğümlerin blok onayı verdiği, dolayısıyla parayla para kazanmanın kripto para dünyasındaki karşılığı olan bir yöntem.

Proof of Stake mutabakat yapısı kapsamındaki blok üretim ve geçerlilik onay mekanizması, bloğu üreten düğümün, ilgili Blockchain ağı üzerinde sahip olduğu pay miktarına göre çalışır. Yani bu yapıda blok onaylama ya da oluşturma yetkisi verirken düğümün cüzdanındaki kripto para miktarına bakıyoruz diyebiliriz. Bu tarz sistemlerde genellikle sistem içerisinde üretilebilecek tüm kripto para miktarı ilk başta **ağı kurgulayanlar** tarafından üretilir. Yani madencilik yapısı yoktur. Sistemdeki üyeler yatırımlarına göre paylarına düşen kripto paralarını alırlar, sonradan toplam arz miktarına yeni eklemeler yapılmaz. Sistem kapsamındaki pay değeri, sahip olunan kripto para miktarına göre hesaplanır.

Pay miktarına göre işlem yapma konusunda **Proof of Stake** yapısını benimsemiş kripto para ağlarında işlemlerin onaylanması açısından farklı yöntemler ve farklı davranış biçimleri geliştirildi. Bunlar arasında en çok tercih edilen **iki yöntem** şu şekilde çalışır:

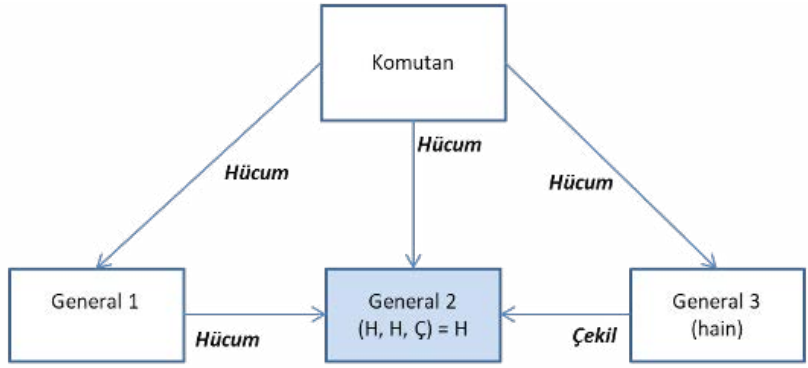
- Bir sonraki bloğu üretecek olan düğüm, sahip olduğu pay ile ilişkilendirilmiş bir **rastlantısal fonksiyon** ile belirlenir. Yani payı yüksek olan düğümün seçilme şansı daha yüksektir. İlgili düğüm belirli bir süre içerisinde uygun bir blok paylaşmaz ise kriterlere uygun bir sonraki düğüme geçilir.
- Bir düğüm belirlemesi yapılmaz, ancak pay bilgisi düğümün çözmesi gereken problemin (*Proof of Work yaklaşımına benzer bir şekilde*) zorluk derecesini değiştirir. Örneğin daha fazla pay sahibi olan düğüm için daha kolay bir **problem çözüm aralığı** sağlanır.

Bu yöntemin iki farklı senaryosunda da bir adaletsizlik sezebildiniz mi? Sanki kapitalist düzenin meşhur sözü olan "**para parayı çeker**" mantığının kripto parayla buluşmuş hâli gibidir. Bu adaletsizliği fark eden tek kişi ben değilim. Tek başına sahip olunan pay değerinin kullanılmasının yüksek pay sahibi düğümler için sürekli bir avantaj yaratmasından dolayı, akış içerisindeki hesaplamalarda kullanılmak üzere bir "**yaş**" (*age*) kavramı düşünülmüş. Bu kavram ile birlikte blok üretimi için kullanılan pay kapsamındaki kripto paraların yaş değerleri sıfırlanır, bu kripto paralar ancak belirli bir süre sonunda yaş değeri kazanmaya başlarlar ve yaş değeri işlemlerde **öncelik/geçerlilik** kazanmada avantaj sağlayıcı olur. İlk gelenin yoğun bir alım yaparak cüzdanındaki pay miktarını artırıp balina konumuna geçerek ağda **karar mercii** olması mümkündür. Bence bu mutabakat yapısının mantıksal hatası temelde budur. Bu tarz bir yapıyı kurgulamanın tek amacı, ICO sürecinde sisteme girmek isteyen yatırımcının alabileceği maksimum kripto para miktarını almasını (*ve hatta sınırlarını zorlayarak bu miktarı artırmasını*) teşvik etmek olabilir.

Bu mutabakat yaklaşımı kapsamında blok üretimi süreci, para basma olarak adlandırılmaz. Şu anda "**Proof of Work**" mantığı ile ilerleyen Ethereum ağının, yakın zaman içerisinde "**Proof of Stake**" yapısına geçirilmesi planlanmaktadır. Bu yapı, aslında daha kompleks bir uygulama şekli olsa da bu geçiş ile birlikte işlemlerin doğrulanma ve blok oluşturma süreci daha hızlı ve kolay bir hâle getirilerek Bitcoin ağında olduğu gibi pahalı ve yüksek güç tüketimine sahip **özel madencilik donanımları**na olan ihtiyacı (*ve dolaylı olarak oluşan donanım tabanlı merkezciliği*) ortadan kaldırması öngörülüyor.

Ayrıca katılımcıların platform ile daha **derin sahiplik ilişkisi** geliştirmesi sağlanarak platformun çıkarına en uygun şekilde çalışmaları teşvik edilmiş olacak. Bu dönüşüm ile birlikte Ethereum platformu, daha büyük uygulamaları barındırmak için **daha ölçeklenebilir** bir hâle gelmeyi hedefliyor. Tabii madenciliğin Ethereum token üzerindeki katma değeri ne duruma gelecek, bilemiyoruz.

Practical Byzantine Fault Tolerance - PBFT



Şekil 1.47: Meşhur Bizans Generalleri Probleminden ilham alan PBFT mutabakat yöntemi, en popüler üçüncü yöntem olarak sektörde yer bulmuş durumda. HyperLedger platformu bu yöntemi benimsemiştir.

Bu yapı adını, Bizanslı generallerin kullandığı bir yöntemden almaktadır. Bizans İmparatorluğu'nda, imparatordan gelen emirlerin gerçek olup olmadığını anlamak için, generallerin kullandığı oldukça basit ve etkili bir yöntem vardı. İmparator, ordusuna bir emir vereceği zaman bunu generallere ulaştırmak için birden fazla ulak yollamakta ve generaller de emri aldıklarında kendi aralarında ulaklar ile bu emirleri paylaşmaktaydılar. Bu süreç içinde eğer imparatordan gelen emir ulakların çoğunluğu tarafından doğrulanmış ise bu emrin doğru olduğu kabul edilir, azınlıkta kalan emirler göz ardı edilirdi.

Bu çözüm, Blockchain dünyasına şu şekilde yansıdı: Ağ yapısına dâhil her **doğrulayıcı** (*validator*) rolüne sahip düğüm için özel bir **açık-gizli anahtar ikilisi** oluşturulur. Her düğüm diğer düğümlerin açık anahtar bilgisine sahiptir.

Düğümler kendisine gelen bir **işlem** (*transaction*) bilgisini, kendi üzerinde tutulan veri yapısını kullanarak kontrol eder, onayladığı bir işlemi imzalayarak ağ ile paylaşır.

Eğer bir işlem belirli bir sayıda (*mesela 2n düğümden oluşan bir ağ için bu sayı n+1 olabilir*) düğüm tarafından onaylanmış ise mutabakat sağlanmış kabul edilir ve bu işlem ağ tarafından geçerli işlem olarak tanımlanır.

Proof of Work, Proof of Stake gibi yaklaşımlardan farklı olarak **PBFT kaynak sahipliği** (*donanım, pay gibi*) akış içerisinde soyutlanmakta, bu sayede en küçük katılımcı bile dâhil olduğu ağın yapısında söz sahibi olmaktadır.

Bu yaklaşım kapsamında ağa dâhil olan tüm doğrulayıcı makinelerin birbirinden haberdar olması ve ağa dâhil olacak yeni bir doğrulayıcının merkezî bir sistem tarafından onaylanması gerekmektedir. Bu durum Blockchain yaklaşımının temellerinden olan "**herkese açık, merkezî olmayan ağ yapısı**" kavramı ile çelişir. Bundan dolayı bu mutabakat yaklaşımını, **açık** (*public*) yapılar yerine, daha çok şirketlerin tercih ettiği **özel** (*private*) Blockchain ağ yapıları içerisinde değerlendirmek daha doğru olur.

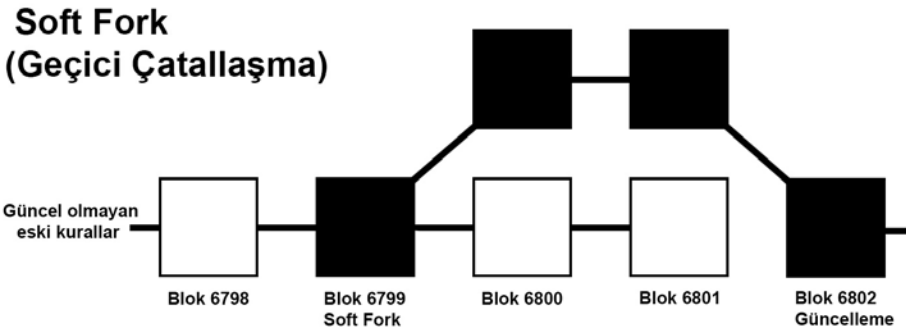
Linux Foundation önderliğinde kurulan ve bünyesine *IBM, SAP Ariba* gibi devleri katmayı başaran **HyperLedger** platformu kapsamında, varsayılan mutabakat yapısı olarak **PBFT** kullanılır.

Bu üç yaklaşımın dışında, Proof of Work ve Proof of Stake'nin birlikte uygulanması ile gerçekleşen "**Proof of Activity**"; ayrıca blok üretim sürecine dâhil olmak için disk alanı tahsis edilmesini gerektiren "**Proof of Capacity**" gibi farklı mutabakat çözümleri de bulunur, ancak bu yapılar henüz deneysel anlamda ve çok küçük ölçekli denemeler kapsamında kullanıldığından **alternatif yöntem** olarak değerlendiriliyor.

Fork (Çatallanma)

Blockchain ağlarını anlattığım bölümde bu yapılara dâhil olmak için "**core**" uygulamalara ya da cüzdan uygulamalarına ihtiyacımız olduğunu söylemiştim. Bu yazılımlar aynı zamanda ağda işlem yapabilmemiz için ihtiyaç duyduğumuz açık ve kapalı ağ anahtarlarını bizim için üretirler. Bu uygulamalar sayesinde ağa özel mutabakat yöntemi ve sürecine dâhil oluruz.

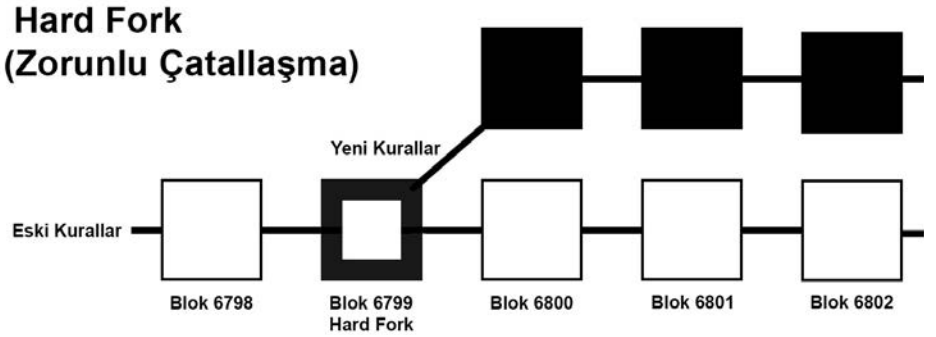
Bu yazılımlar genellikle açık kaynak kodlu olarak geliştirilir (*Bitcoin MIT lisansı ile dağıtılan açık kaynak kodlu bir uygulamadır*) ve Github gibi yapılar üzerinde herkesin erişimine açıktır. Bu sayede gerek kripto paraların çekirdek geliştirici kadrosu gerekse de topluluk içerisindeki gönüllü geliştiriciler sürekli olarak kodları günceller ve geliştirir. Bu güncellemeler genellikle **ağın yeteneklerini ve imkânlarını geliştirme**ye ve **performansını artırma**ya yönelik yapılır. Ancak ağın genel yapısını, yani *blok oluşturma, toplam arz miktarı* ve *mutabakat yapısı* gibi karakteristik özellikleri etkileyecek değişiklikler yapılmaz.



Şekil 1.48: Soft Fork adını verdiğimiz geçici çatallanmaya tabii olmuş bir zincir örneği

Önceki bölümlerde bir Blockchain ağındaki her bir noktanın, ağın yerel bir kopyasına sahip olduğunu söylemiştim. Bazı istisnai durumlarda yazılımda gerçekleştirilen güncellemeler sonrasında, güncelleme yapanlar ağa yeni blok eklemeye devam ederken güncelleme yapmayanlar yeni blok ekleyemez hâle gelirler. Ancak mutabakat sürecine dâhil olabilirler (*eklenen blokları doğrulayabilirler*) ve akabinde bu blokları kendi zincirlerine kopyalayabilirler. Bu Blockchain ağında bir "**Geçici Çatallaşma**" (*Soft Fork*) durumu ortaya çıkartır. Geçici çatallanmaya maruz kalan kullanıcılar core uygulamalarını güncel sürüme yükselttiklerinde ise artık çatallanma yapısından çıkarak **ana blok zincire** dönerler.

Fakat bazı yazılım güncelleme durumlarında geliştiriciler, yaptıkları kod değişiklikleriyle güncelleme yapmayan düğümlerin artık mutabakat sürecine dâhil olmasını engeller. Yani ağda oluşan yeni blokları doğrulayamaz, okuyamaz ve kopyalayamazlar. Yazılım güncellemesi yapanlar Blockchain ağında yeni bloklar eklemeye devam ederken yazılım güncellemesini yapmayanlar yeni yapıya katılamamakla beraber, eski mutabakat yapısında yeni bloklar ekleyebilirler. Böyle bir senaryoda eski yapıdaki düğümlerin bir kısmı yeni yapıya geçmeyerek eski zinciri devam ettirebilir. Bu durumda ister istemez birbirinden ayrılan iki tane yeni Blockchain ağı oluşmuş olur. Bu tarzda gerçekleşen çatallanma tipine "**Mecburi Çatallaşma**" (*Hard Fork*) adını veriyoruz.



Şekil 1.49: Hard Fork yapısına örnek bir zincir yapısı. Bu tarz çatallanmalarda eski zincire sadık düğümler varsa birbirinden bağımsız iki Blockchain yapısı oluşmuş olur.

Blockchain ağlarında yaşanabilen çatallanma probleminin en büyük örneği "**DAO Olayı**" olarak adlandırılır ve 2016 yılında Ethereum ağında gerçekleşmiştir. Aslen bir token olan ve gelişmiş bir "**akıllı sözleşme**" yapısı kurgulamayı hedefleyen DAO platformu (*Decentralized Autonomous Organization – Merkezî Olmayan Özerk Kurum*), herhangi bir beşeri desteğe ihtiyaç duymadan bir kurumun kurallarını ve karar verme mekanizmalarını bir akıllı sözleşme kapsamında tanımlayarak çözüm sunmayı hedeflemekteydi. DAO projesi **Nisan 2016** tarihinde Ethereum üzerinde çalışmak üzere tanımlanmış bir şekilde ICO gerçekleştirmiş ve 11.000 civarında kişinin katılımı ile birlikte **170 milyon dolar** değerine yakın **Ether** (*ETH*) toplanmıştır. Bu toplanan kaynak, girişim sermayesi fonu olarak kullanılmak üzere DAO'nun yönetimine bırakılmıştı. Ancak DAO'nun üzerine inşa edildiği akıllı sözleşme yapısındaki bir kod açığını fark eden hacker'lar, toplanan fonun yaklaşık 1/3'ünü kendi hesabına aktarmayı başarmıştır.

Buradaki önemli nokta bu işlemi gerçekleştiren hacker'ın Ethereum ve DAO kurallarına aykırı davranmayarak herhangi bir ihlal gerçekleştirmemiş olmasıydı. Yani sisteme bir siber saldırı metodu kullanarak girmemiş ve para, hesaplar hack'lenmek suretiyle alınmamıştı. Bu aşamayı, cebimizden düşürdüğümüz cüzdanı gördüğü hâlde bize vermek yerine kendine alan bir kişi örneğine benzetebiliriz. Aslen etik olmamakla birlikte fiilî bir şekilde suç sayılabilecek bir hırsızlık da diyemiyoruz.

Bu durum sonrasında, sadece bu olaya özgü işlemlerin iptali için bir **Mecburi Çatallanma** işlemi gerçekleştirilmiştir. Ancak Ethereum yapısı üzerindeki bazı kullanıcılar (*benim gibi*), "**kurallara uygun davranan her işlem geçerli bir işlemdir**" düşüncesi ile bu değişikliği kabul etmedi. Bunun üzerine eski kod yapısı ile devam edenler **Ethereum Classic** (*ETC*) adını almış ve yeni kod yapısı ile devam edenler **Ethereum** (*ETH*) şeklinde yoluna devam etmiştir. Bu sebeple şu anda aynı başlangıç noktasından doğan ancak farklılaşarak yoluna devam eden **iki farklı** Ethereum Blockchain ağı bulunmaktadır.

Madencilik Yapıları ve Kaynak Tüketimi

Kripto para denince ilk akla gelen soru **madencilik** oluyor. İnsanlar mevcut donanımları ile bir hesaplama yapıp bu hesaplamanın sonucunu da para olarak kullanabilecekleri fikrini çok seviyorlar. Bunun temel nedeni kuşkusuz **kaynak kıtlığı** ve **gelir eşitsizliği** diyebiliriz. Ancak madenciliğin evlerdeki donanımlarla yapılabildiği dönemler 2012'lerde bitti diyebilirim. Bugünlerde madencilik ile para kazanmayı hedefliyorsanız ciddi bir başlangıç sermayesine ihtiyacınız olacak. Merak edip internette kendi "**rig**" yapılarını oluşturmanın yollarını araştırmış olan okurlarım bilirler ki; en azından 20 adet yüksek yapılı grafik kartına ihtiyaç duyuyoruz. Bu kartları özel donanımlar sayesinde bir ana karta bağlayarak çalışmaya başlıyoruz. Ancak bu noktada çok sancılı bir ayar ve çalışma süreci var. Bu süreç zarfında donanımlara zarar verme ihtimaliniz yüksek. Ayrıca her şey yolunda gitse bile çalışma esnasında kartların bazıları (*belki hepsi*) arızalanabilir. Bu yüzden mutlaka **yedek kartlar** bulundurmanız gerekiyor. Tabii elektrik faturanızı da düşünmeniz gerekiyor. Çünkü kartların çektiği elektriğin yanı sıra sistemin bütününü soğutmak için harcayacağınız ek elektrik maliyetlerini de düşünmeniz gerekiyor. Bunun yanı sıra **geniş bant internet bağlantısı** ve **bol bol sabit disk alanı**na sahip olmanız iyi olacak.



Şekil 1.50: 8 adet grafik kartından yapılmış "ev tipi" bir rig görüyorsunuz. Bundan 5 yıl önce böylesi bir rig size günde 1 BTC kazandırabilirdi.

Ancak işin acı tarafını bir kenara bırakırsak bir de tatlı tarafı var. O da **işlem onay kısmı**. Yani madencilik yaparken hem blok çözüp içerisindeki ödül kripto para miktarını almak için yarışıyorsunuz hem de ağdaki kullanıcıların işlemlerini onaylayarak işlem ücretlerini topluyorsunuz. Zaten ağ yapısında toplam arz dolaşıma katılıp da madencilik işi bittiğinde artık sadece işlem onay ücretlerini kazanmaya başlıyorsunuz. Yani sürekli bir gelir kapısı oluşturmuş oluyorsunuz. Tabii sektörde sürekli yeni kripto para ağları baş göstereceği için **zorluk derecesi düşük** kripto paraların madenciliğini yapmayı deneyebilirsiniz.

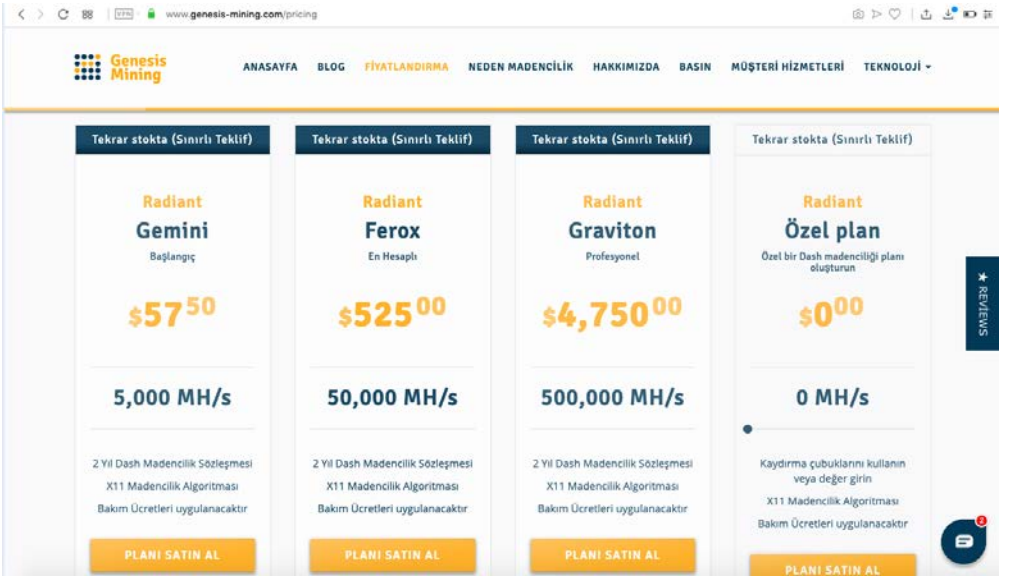
Ancak 2015 yılından itibaren popülerleşen bir başka madencilik türü var: **Bulut Madencilik**.

Bulut madencilik adından da anlayabileceğiniz üzere, bulut sunucularda dağıtık şekilde kurulmuş bir **maden havuzu**ndan bahsediyoruz.

Bu tarz yapılar genellikle çok büyük hangarlarda (*Tabii gerçekten bu işi yapıyorlarsa! Zira Bulut Madencilik yaptığını iddia ederek para kazanmayı hayal eden girişimcilerin paralarını çalan yapılar da var*) binlerce grafik kartından oluşan rig yığınlarıyla **kripto para madenciliği** yaparlar. Sizler de bu tip şirketlerden bütçenize uygun "**madenci**" paketleri satın alabiliyorsunuz. Yani rig yapılarını kiralamış oluyorsunuz. Bu tarz yapılar evde rig kurmaktan çok daha az maliyetli olabiliyor. Yalnız hemen hemen tüm bulut madencilik firmaları, satın aldığınız hash gücünün ürettiği kripto para miktarından temel altyapı giderlerini düşerler. Bu temel altyapı giderleri aslında genel elektrik, internet, IT elemanlarının maliyetleri ve yedek donanımlar için oluşturulmuş operasyonel giderler genel havuzu için bir **katkı payı**dır diyebiliriz. Tabii ki eşit olarak paylaştırılmıyor bu giderler. Her müşteri sahip olduğu hash gücü ile doğru orantılı bir şekilde kesintilere tabii tutuluyor.

Bulut madencilik yapıları genel olarak gün sonları kazdığınız kripto para miktarlarını belirlediğiniz **cüzdan adreslerine** yollarlar. Günde 0.01 BTC kazabileceğiniz bir hash gücü satın aldığınızı düşünürsek hemen her gün 86 USD kazandığınızı söyleyebiliriz (*BTC kuru şu sıralar 8600 USD*). Türkiye şartlarında oldukça iyi bir ek gelir diyebilirim. 0.01 özel bir rakam değeridir. Şu sıralar BTC ağında gönderim ücretleri yaklaşık olarak 0.0005 BTC olduğundan genellikle minimum gönderim miktarları 0.01 BTC oluyor. Ancak bulut maden yapılarında başka kripto paraları da kazmanız mümkün. Genellikle *Litecoin, Dogecoin, Ethereum, Monero* ve *Dash* gibi kripto paralar en popüler madencilik paralarıdır.

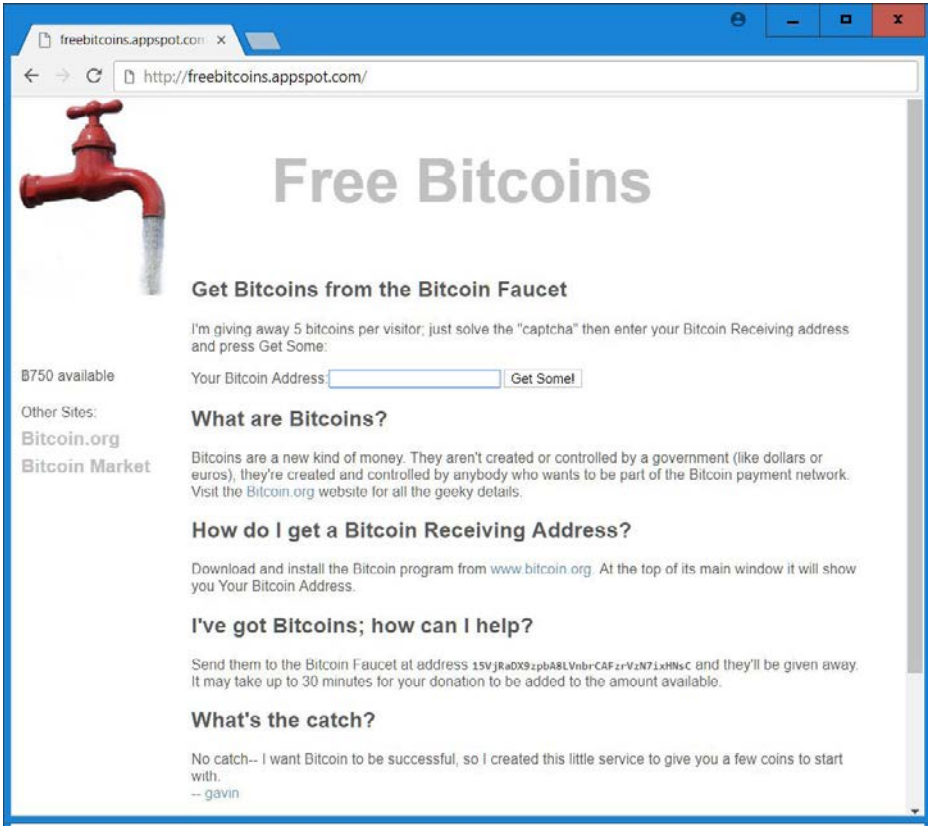
Bulut madencilik ilk doğduğu dönemlerde ömür boyu BTC madencilik kontratları satılıyordu. Ancak son yıllarda **süreli** (*genellikle 1 ile 3 yıl arasında*) kontratlar satılmaktadır. Bu arada unutmadan ekleyeyim: Madenciliği yaptığınız kripto paranın zorluk derecesi yükseldikçe ve piyasa değeri düştükçe kazdığınız miktar azalacak ve artık altyapı giderleri için kesinti miktarları negatife düşecektir. Bu durum gerçekleştiğinde kontratınız kapanır. Yani sonlanır. O yüzden piyasayı sürekli takip etmeniz gerekir.



Şekil 1.51: Bulut madencilik denilince akla ilk gelen şirket Genesis Mining'dir. Yukarıda 2019 yılı ikinci çeyreğine ait bulut madencilik paketleri görülüyor.

Kripto paralar ilk popüler olmaya başladıklarında "**musluk**" (*faucet*) adı verilen web siteleri türemeye başladı. Tek sayfadan ibaret bu basit web siteleri, belli periyotlarda ziyaret ettiğiniz zaman size ödül olarak kripto para veriyorlardı. Bu periyotlar **5 dakika** ile **1 gün** arasında değişiklik gösteriyordu. Yani her 5 dakikada bir siteyi ziyaret etmeniz durumunda belli miktarlarda kripto para kazanıyordunuz.

Peki, bu siteler neden kripto para dağıtıyordu, diye düşünebilirsiniz. Popüler oldukları yıllar aşağı yukarı 2010 ve sonrasına denk geliyor. O yıllarda BTC'nin piyasa değeri 1 USD'den az ve online reklam gelirleri hâlâ iyi bir kazanç kapısı. Kripto para tutkunlarına reklam izlemeleri karşılığında BTC dağıtmak oldukça kazançlı bir işti diyebilirim. O yıllarda ben de bu tarz siteleri takip ediyordum. Bir musluk sitesinden günde (*24 saatte*) 5 BTC kazanıyordum. O dönemler kaç tane musluk sitesi takip ettiğimi söylemeyeceğim. Ama 2011 yılında BTC fiyatları 30 USD'yi aştığında musluk sitelerinin dağıttığı BTC miktarları bir anda azalmaya başladı. Takdir edersiniz ki günde her ziyaretçinize 150 USD verecek reklam gelirini elde etmek bir hayli zordur.



Şekil 1.52: 2010 yılında ziyaretçilerinin her birine günde 5 BTC dağıtan musluk sitesi

Bu tarz siteler hâlâ faal. Yani dilerseniz musluk sitelerini takibe alarak sıfır maliyetle kripto para toplamaya başlayabilirsiniz. Şu aralar **Bitcoin, Bitcoin Cash, Litecoin, Dogecoin** ve **Dash** en çok musluk sitesine sahip kripto paralar diyebilirim. Ancak 8-9 yıl önce dağıtılan miktarları elde etmeyi hayal etmeyin. Fakat birden fazla musluk sitesini takibe alıp azimle ziyaretlerinizi sürdürürseniz 1 ay gibi bir sürede 10 ile 20 USD aralığında kripto para toplamanız mümkün. Bu rakamlar sizi zengin etmez tabii ki ama dilediğiniz kripto para borsasına giriş yaparak küçük yatırımcılar kervanına katılabilirsiniz. Kim bilir, belki de kısa sürede paranızı birkaç katına çıkartabileceğiniz yeni nesil bir kripto para bulabilirsiniz. Musluk sitelerini ve "**Airdrop**" adını verdiğimiz ücretsiz ICO metotlarını takip ederek kısa sürede borsalarda **al-sat** yaparak para kazanacak yatırım tutarlarına sahip olabilirsiniz.

Madencilik denildiğinde hatırı sayılır bir kesimin tüyleri diken diken oluyor ve dünyada "**daha faydalı**" işler için kullanılabilecek miktarlarda elektrik enerjisinin kripto para üretmek için boşa harcanmasını (*yani madencileri*) kınıyor. Bu bakış açısına katılmamakla birlikte **yeşil enerji ile madencilik** yapılabileceğine inancım tam. Bu arada madenciler elektriği bedava kullanmadıkları için aslında boşa harcanmış bir elektrik sarfiyatından bahsetmiyoruz. Sonuçta bankacılık ve finans sektörü de çok büyük miktarlarda elektrik tüketimine sahip (*yapılan araştırmalara göre madencilikten daha fazla bir tüketimden bahsediyoruz*). Üstelik madenciler gibi yüksek katsayılarla çarpılmış faturalar ödemek yerine, kurumsal anlaşmalar sonucunda ev kullanıcılarından çok daha az birim ücretlere sahip elektrik faturaları ödüyorlar. Bu noktada madencilik faaliyetlerini irdelerken birçok hesaba katılmayan etken olduğunu da düşünmemiz gerek. Ayrıca madencilik dediğimiz olgu (*Ethereum ve Dogecoin gibi toplam arzın sınırsız olduğu kripto paralar hariç*) toplam arzın tamamen dolaşıma katılmasıyla sonlanan bir durum. Ancak önceki bölümlerde anlattığım mutabakat yöntemlerinden **Proof of Stake,** elektrik maliyetlerini ortadan kaldırmak için düşünülmüş çözümlerden biridir.

Coin vs Token: Hangisi Gerçek Para?

Kripto para dünyasına adım attığınızda birçok farklı terim ile muhatap oluyorsunuz. Bunlardan bazıları anlam olarak birbirine çok yakın olmasına rağmen, teknik olarak çok farklı olabiliyor. Mesela **Coin** ve **Token** terimleri bunların başını çekiyor. Genellikle kripto para ekosistemine adım atan "**çömez**"ler ikisini birbirinden neye göre ayırdığımızı anlamakta zorlanıyorlar.

Mantıken ikisi de İngilizce anlamı bakımından aynı amaca hizmet ediyor. Coin ve Token bizim bildiğimiz anlamda "**jeton**" olarak kullanılıyor. Fakat fiyat paralarda (*reel para birimlerinde*) **madeni birimler** için de coin tabiri kullanılıyor. Aslen tam Türkçe karşılığı olarak (*belki de eski Türkçe demeliyim*) **Sikke** anlamına geliyor. *Peki, kripto dünyasında ayrım nasıl yapılıyor?* En basit ifadeyle, kendi özel Blockchain ağına sahip kripto paralara **coin** diyoruz. Kendi düğüm yapıları, karakteristik cüzdan adresleri ve özel mutabakat yapısına dayalı madenciliği ile tam bir kripto paradan bahsediyoruz. *Bitcoin, Litecoin, Dogecoin, Dash, Ethereum* ve *IOTA* gibi kripto paralar coin dediğimiz yapıya örnektir.

Token tabiri ile tanışmamız Ethereum ağının doğuşuna tekabül ediyor. Ethereum ağı üzerinde akıllı kontratlarla oluşturulan kripto paralara token diyoruz. Token dediğimiz yapılar kendi Blockchain ağı olmayan ama Ethereum ağı üzerinde **yan ağ** olarak çalışan ve Ethereum sanal makinesi tarafından yönetilen kripto paralardır. Yapı olarak Ethereum'dan farksızdırlar. Ancak sizin verdiğiniz isim ve kısaltmayla anılırlar. Madencilikleri olmaz. **Pre-mine** dediğimiz, tüm toplam arz miktarının dolaşıma sunulduğu kripto paralardır. Borsalara oluşturuldukları akıllı sözleşme protokolüyle eklenirler. Tabii değerleri sizin geliştirdiğiniz iş modeli ve hayata uygulanabilirliğiyle ölçülür.

Token'ler genellikle **ICO** düzenleyen şirketlerin vazgeçilmez enstrümanı konumuna oturmuştur. Bir nevi talep toplama ve kaynak yaratma aracıdır diyebiliriz. Bu yüzden kripto para borsaları **token listeleme**yi çok sever. Kripto para borsalarının listelerine bakacak olursanız token sayısının çok yüksek olduğunu görürsünüz. ICO aracı olarak kullanılan token'ler talep toplama sırasında geliştiricilerin belirlediği bir değer üzerinden satıldığı için borsalara risksiz görünür. Ancak token tarafında değerlemeler genellikle afaki yapılır. Yani geliştirici ekip bir **White Paper** hazırlar. Bu biraz teknik, biraz ekonomik biraz da iş modeli içeren bir dokümandır. Yatırımcı, bir White Paper incelerken geliştiricilerin mevcut hangi sorunu Blockchain ve kripto para ile çözdüğünü görmek ister. Ayrıca modelin gelir düzeneğini, yol haritasını ve ekibin niteliğini de inceler. Eğer aklına yatan bir modelle karşı karşıya olduğuna inanırsa ICO sürecine katılarak token satın alır. Token toplamanın ICO ve borsalar dışında bir **alternatifi yoktur**. Token yapısı madencilik içermediğinden dolayı sadece satın alarak sahip olacağınız yapılardır. Bazen bir token, borsalarda işlem görmeye başladığında ICO sürecinde satıldığı değerden çok daha yukarıya ya da aşağıya inebilir. Bu süreçte kâr elde etmek ya da zarara uğramak mümkündür. Borsalarda değerin değişmesinin temel iki nedeni vardır: ICO sonrası geliştirici ekibin vaatlerini yerine getirmesi/getirememesi ve borsadaki oyuncuların token'e gösterdikleri ilginin artması/azalması.

Evet, sürekli duyduğunuz coin ve token nedir, ne değildir, artık biliyorsunuz. Bu aşamadan sonra projenizde coin ya da token kullanmak sizin vereceğiniz bir karar olmakla birlikte, sadece birkaç saat ayırarak Ethereum ağı üzerinde bir akıllı sözleşme oluşturup kendi yan ağınız üzerinde çalışacak bir token hazırlayabilirsiniz. Bunun alt yapı maliyetlerini coin geliştirmeye nazaran düşürdüğünü hatırlatmak isterim. Ancak projenizin yapısı kendinize özel bir Blockchain ağı gerektiriyorsa mecburen coin geliştirmek için kolları sıvayacaksınız.

Stable Coin: Karşılık Para

Kripto para dünyasında coin ve token dışında kullanılan bir terim daha var: "**stable coin**", yani "**sabit para**". Bu para türü teknik açıdan bakıldığında bir coin ya da token olabilir. Burada stable coin dediğimiz yapıyı farklı kılan şey değerinin **sabit**, **değişmez** ve karşılığı olan fiyat paranın ayrılarak bir **emanet hesapta saklanıyor olması** diyebiliriz. *TrueUSD, Tether, CircleCoin* gibi paralar bu yapıya örnektir. Bu tarz paraların geliştirilme amacı genellikle bir sistem içerisinde kullanılacak değer deposu yaratmak ya da dalgalı değeri sahip coin ve token'ler ile yapamayacağımız uçtan uca para transferi gibi iş modellerini daha efektif hâle getirmek diyebiliriz. Popüler olduğu bir başka alan da **kripto para borsaları**dır. Yatırımcılar kripto paralar üzerinde elde ettikleri kârlılığı korumak için değeri dalgalanmayan bir yapıya dönmek zorundadır. USD işlem yapabilen borsalarda bu işi **USD** üstlenir. Ancak her borsa (*lisans gerekliliği ve regülasyon eksikliği gibi durumlar yüzünden*) USD ile gerçek anlamda işlem yapamaz. Bu yüzden, onun yerini alabilecek, değeri USD ile aynı ve değişmez olan yapılara ihtiyaç duyarlar. **Stable coin** bu iş için biçilmiş kaftandır.

Bunu bir örnekle açıklayayım: Borsada bir token ya da coin üzerinden kâr elde ettiniz diyelim. Bu noktada kârı alabilmek için elinizdeki coin ya da token'i satmak durumundasınız. Satış yaptığınızda ise elinize Bitcoin ya da Ethereum cinsinden bir para geçmiş olur. Ancak Bitcoin ve Ethereum da **dalgalı kur**a sahiptir ve değerleri düşebilir. Bu noktada Bitcoin ve Ethereum üzerinde kalarak belki de altcoin üzerinden kazandığınız kârı kaybetme riskiniz olur. İşte borsalar yatırımcılarını bu sıkıntılı durumdan kurtarmak için stable coin listelemeyi tercih eder. Altcoin'i ya da Bitcoin ve Ethereum gibi ana coin'leri sattığınızda stable coin'e geçerek kârınızın fiyat para karşısında erimesini önlemiş olursunuz.

Stable coin, **madenciliği olmayan** bir yapıdır. Pre-mine olarak arz edilir ve karşılık esasına göre çalıştığından sadece karşılığı yatırılanlar dolaşıma çıkar. Karşılığı olmayan stable coin'ler dolaşıma sürülmez. Bazen karşılığı olmadığı hâlde dolaşımda olduğu tespit edilen stable coin'lere rastlanabilir. O zaman bunlara karşılık yaratabilmek için henüz dolaşıma sürülmemiş stable coin'ler imha edilir. Bu işleme "**coin burn**" ya da **para yakma** denir. Para yakma işlemi her türlü coin ve token için uygulanabilir.

ICO VE IEO



Şekil 1.53: ICO yapısı kripto para ve Blockchain ile ilgili olmayan şirketler için bile büyük bir finansal kaynak yaratma yöntemi oldu.

Önceki bölümde token hakkında yazdıklarım arasında ICO kısaltmasından bahsettiğimi fark etmişsinizdir. **ICO** dediğimiz kavram, 2014 yılından bugüne duyduğumuz bir **kaynak toplama metodu**dur. **Initial Coin Offering** tabirinin kısaltması olarak kullanılır. Bu yapı kurgulanırken reel dünyadaki şirketlerin halka arz yöntemiyle borsalar üzerinden hisse satıp kaynak toplama süreçlerinden ilham alınmıştır. Yani "**Initial Public Offering**" (*Halka Arz*) mantığının Blockchain ve kripto para dünyasına yansımasıdır diyebiliriz.

ICO, temel olarak herhangi bir token ya da coin için bir Blockchain platformu üzerinde satış veya teklif olarak yatırımcılara kaynak sağlama amacıyla sunulur. ICO süreçlerinin tamamında arz edilen token'ler, Bitcoin veya Ether gibi en popüler kripto paralar karşılığında **değiş-tokuş** edilir. Bu işlem Blockchain ağı üzerinde akıllı sözleşme altyapısıyla kaydedilir. Yani güvence altına alınır.

Gerçekleşen ICO uygulamaları ile alakalı pazar verilerini yayınlayan **Coindesk** sitesinde yer alan bilgilere göre 2014 yılının başından 2018 yılının ilk çeyreği sonuna kadar gerçekleşen ICO'ların pazar değeri **12,11** milyar doları geçmiş durumda. Bu rakamın 7,4 milyar dolarının 2018 yılının ilk çeyreğinde, 5,4 milyar dolarının 2017 yılında, 300 milyon dolarının 2016 yılında ve geri kalan kısmının ise 2015 ve 2014'te gerçekleştiğini görüyoruz.

Elbette bu rakamlar ABD doları bazında işlemleri değil, bu değer karşılığında gerçekleşen ve ağırlıklı olarak **Bitcoin ve Ether ile yapılan transferler**i ifade ediyor. Bu noktadan hareketle kaynak ihtiyacı olan bir şirketin kolayca ve hiçbir prosedüre bağlı kalmadan, göreceli olarak çok kısa bir sürede ihtiyaç duyduğu finansal kaynağı, bir token ya da coin üzerinden "**kârlılık**" vaat ederek yaratması sürecini ICO olarak tanımlıyoruz diyebiliriz.

2019 yılına geldiğimizde ise ICO yapısının "**özel arz**" yapılarına da sıcak baktığını görüyoruz. Yani halka arz yerine belli kriterleri sağlayan ve amaçlar doğrultusunda bir araya gelen şirketlere yönelik ICO'ları deneyimledik. Bunların en önemlisi (*bana göre*) bir kripto para borsası olan **Bitfinex**'in düzenlediği "**Private Token Sale**" (*Özel Token Satışı*) ICO'suydu. Belli bir kesimi hedefleyerek düzenledikleri ICO sonucu 1 milyar USD değerinde (*stable coin kullanılarak yapılmıştır*) kaynağı 10 gün gibi bir sürede toplamayı başardılar. Bu ICO bize şirketlerin kaynak yaratma konusunda kullandıkları IPO sürecinin de evrimin eliyle geliştiğini gösteriyor. Tabii 2019 yılına kadar gerçekleşmiş ICO'lar toplamında yaratılan kaynağın neredeyse yüzde 10'luk bir kısmı, 2019 yılında yapılan tek bir ICO ile gerçekleşmiş oldu. Bu da bize ICO mantığının çok hızlı bir şekilde kabul görerek büyüdüğünü gösteriyor.

IEO yapısı ise 2019 yılında popülerliğe kavuşan bir kavram. "**Initial Exchange Offering**" yani **Borsa Üzerinden Arz Yapma** kelimelerinin baş harflerinden oluşan bir kısaltma. Şu sıralar internette hemen her yerde IEO reklamlarını görebilirsiniz. IEO süreci, halka arz konularına sıcak bakan bir borsa bularak başlıyor. Genellikle *Bittrex, Binance* gibi büyük borsalar başta olma üzere çoğu borsanın bu süreci desteklediğini, üzerlerinde gerçekleşen IEO örneklerine dayanarak söyleyebilirim.

Bu mantığın ICO'dan tek farkı, satış yapacağınız yatırımcıların hâlihazırda işlem yaptıkları bir borsa üzerinde olmaları. Günümüz IPO yapısı zaten bildiğiniz gibi borsalar eliyle yapılabiliyor. Bir şirketin kendi başına halka arz amacıyla hisse çıkartıp kendi üzerinden satması mümkün değil. Öncelikle şirketin yapısının, pazar payının ve finansal yeterliliklerinin incelenerek bir onay sürecini tamamlaması bekleniyor. Ardından borsa üzerinden ve aracı kurumlar eliyle hisseler satışa çıkıyor. Bunu izleyen süreçte şirket periyodik olarak halkı kendisiyle ilgili gelişmelerden haberdar etmekle yükümlü oluyor.

Tüm bu süreç bu kadar prosedüre bulaşmadan ICO mantığında ama kaynak arayan şirketin kendi eliyle değil de bir **borsanın aracılığıyla** yatırımcıya sunulmuş oluyor. Tabii konuyla alakalı bilgisi olan okurlarım bilirler ki; borsalarda bir coin veya token listeletmek pek kolay bir süreç değildir. Borsaların bazı kriterleri vardır. Gelir kapıları işlem üzerinden aldıkları komisyon oranları olduğu için işlem hacmi büyük olmaya aday kripto paraları listelemek isterler. Bu noktada kripto paranın White Paper'inin özenli ve iş modelinin ayakları yere basan tarzda olması beklerler. Ekibinin bu işlerin altından kalkacak kişilerden seçilmiş olmasını beklerler. Tabii dünyanın her yerindeki girişimci ve yazılımcıları tanıyamayacaklarından projenin işinin ehli danışmanlara sahip olmasını görmek isterler. Son olarak da bir topluluk tarafından desteklenip desteklenmediklerini bilmek isterler. Bu önemlidir; **çünkü topluluk varsa diğer kalemler göz ardı edilebilir**. (*Örneğin, Bitcoin ekibi hakkında hiçbir şey bilmiyoruz ama White Paper olarak gayet yeterli ve büyük bir destekçi topluluğu var.*) Bunları sağlıyorsanız bir miktar bağış ya da gelir paylaşım modeli sunarak kripto paranızı listeletebilirsiniz. İşte bu kriterler IEO sürecinde de devreye giriyor. Borsalar, içi boş kripto paraların hâlihazırdaki kullanıcılarını zarara uğratması riskini göze alamıyorlar. Üstelik ICO sürecinde ICO düzenleyen şirket yetkilileri dolandırıcı çıkarsa tüm sorumluluk onların üzerinde kalıyor. Ancak aynı durum IEO sürecinde gerçekleşirse sorumluluk aynı zamanda borsayı da bağlıyor. Bu alandaki tüm yasalar henüz griyken borsa sahipleri risk almak istemiyorlar.

Nasıl ICO Yapılır?

Buraya kadar gayet güzel bir şekilde ICO mantığını anladık. Üstelik bizim de kendi projelerimiz için kaynağa ihtiyacımız var. O halde ICO yapalım ve kaynak ihtiyacımızı çözelim. *ICO yapalım da nasıl yapalım?* ICO sürecine girmek için bu tarz işlemleri yönetmeyi ve geniş kitlelere ulaştırmayı vaat eden **platform** ve **danışmanlar** her yerde bulunabiliyor. Öncelikle bu yolu deneyebilirsiniz. Tabii bu yapıların ve danışmanların bir **gelir modeli** var. Öncelikle bu konuda mutabakat yapmanız gerekiyor. Bu aşamayı geçtiyseniz ya bir token ile ya da kendi Blockchain ağınız üzerinde ürettiğiniz bir coin ile bu sürece adım atabilirsiniz. Eğer bir coin ile yola çıkacaksanız doğru hamle bir miktar coin'i pre-mine ederek ICO sürecinde satmak ve geri kalanını madencilere bırakmak olacaktır. Tabii madencilik için gerekli core yazılımını ICO sürecinden sonra dağıtmanız mantıklı olur. Yoksa madencilik yapmayı size kaynak yaratmak için satın alma işlemine tercih edebilirler.

Bu aşama size riskli geliyorsa ICO süreci için bir token üretip ICO sonrası kendi Blockchain ağınıza geçince bu token miktarlarını yeni coin'iniz ile değiştirme yoluna gidebilirsiniz. Kendine ait veya herhangi bir Blockchain ağı üzerinde oluşturulan coin ve token'ler için **farklı satış yapıları ve süreleri** bulunuyor. Bunlardan ilki, **sabit bir fiyat belirlemek**. İşin niteliğine göre üretilen token'lere bir bedel belirleyerek satışı başlatabilirsiniz. Örneğin 10 Kodlab coin (*böyle bir coin yok, örnek olması için uydurdum*) için 0.01 BTC bedel öngörebiliriz. 1 BTC'nin 8000 USD değere sahip olduğu şu günlerde 10 Kodlab coin 80 USD değere sahip olmuş oluyor. Ancak bu tarz bir değerleme sizi ICO sonunda farklı bir noktaya getirebilir. Mesela BTC'nin değeri 4000 USD seviyesine gerilemiş olabilir. Bu senaryoda ICO sonlandığında 10 Kodlab coin 40 USD yapıyor olacak. Yani bütçeniz için düzenlediğiniz ICO başarılı bitmiş olmasına rağmen hedef rakamın sadece yarısını toplamış olacaksınız. Bir diğer yaklaşım ise Kodlab coin'lerimizi açık artırmaya sunarak **fiyatı alıcıların belirlemesine izin vermek** olabilir. Ancak bu seçenek hem popüler bir seçenek değil hem de bütçenizi toparlamanız için koyduğunuz hedefin çok çok altında bir değerle ICO'yu sonlandırabilirsiniz. Unutmayın, ICO sürecinde topladığınız para, siz projenizi vaat ettiğiniz şekilde hayata geçirdiğinizde "**yatırım**", hayata geçiremediğinizde "**borç**" hâlini alıyor. Eğer proje topladığınız parayla hayata geçmiyorsa iade etmeniz sizi dolandırıcılıkla yargılanmaktan kurtaracaktır.

Ben size, bu sürece coin ya da token başı sabit bir değer belirleyerek başlamanızı öneririm. Arz edeceğiniz token'ler için bir değer belirlendikten sonra, sıra süreye karar vermeye geliyor. Bunun için de kabul görmüş iki seçeneğimiz bulunuyor:

- **Sınırlı Token Satışı (*Hard Cap*):** Satılacak token adedi için bir üst limit belirleniyor. Satış, bu adetteki kadar token satılınca son buluyor.
- **Süreli Token Satışı (*Soft Cap*):** Bu senaryoda ise belirli bir süre sınırı koyarak bu süre içinde belli bir adet sınırı olmadan token'ler satışa sunuluyor.

Tabii siz bunlardan birini seçmek yerine ikisini birden ICO'nuzda kullanabilirsiniz. Örneğin, "*10 gün süreyle 1 milyon Kodlab coin satışa sunuldu*" dediğimizi varsayarsak hangi koşulumuz önce biterse ICO'muz da bu süreçte sonuçlanmış oluyor. ICO yapacak olan girişimler sıklıkla ICO sürecini daha cazip hâle getirmek için bazı **ödül mekanizmaları** sunarlar.

Örneğin, 30 günlük satışın ilk 10 gününde ICO'dan token alanlara yüzde 10 oranında fazla coin vermek, ikinci 10 günlük dönemde yüzde 5 oranında fazla coin vermek gibi...

Buraya kadar gördüğünüz üzere, yeni nesil bir finansman kaynağı olarak karşımıza çıkan ICO ve IEO yapıları günümüzün global bir sorunu olan kaynak yaratma ve kaynağa ulaşma gibi sıkıntıları aşmakta çok **kilit bir rol** oynuyor. Ancak bazı fırsatçıların elinde ICO mantığı çok ciddi bir zarar makinesine dönüşebiliyor. Kaynaklar sömürülürken sektöre inancı olan yatırımcıların güvenini tamamen kaybetmesine sebep olabiliyor. Şimdi gelin biraz da ICO dünyasının risklerinden bahsedelim.

ICO'ların Olası Riskleri

2019 itibariyle söyleyebilirim ki Estonya, Singapur ve Malta gibi bazı ülkeler hariç, dünya üzerinde ICO süreçlerini düzenleyen ya da ICO yapanı da ICO'da yatırım yapanı da koruyabilecek düzeyde kanunlar çıkartmış olan bir yer yok. Şimdilik… Ülke yönetimlerinin ICO, Blockchain ve kripto para odağında bu kadar kayıtsız kalması, sektörde ciddi bir boşluk yaratıyor ve bu boşluk ICO'yu dolandırıcılık gibi art niyet anlamında kullanmak isteyenlerin elinde dipsiz bir çukura dönüşüyor.

ICO'ların hemen hepsinde token değerlemeleri hiçbir mantıksal temele dayanmıyor. Tek amaç, işi hayata geçirmek için ihtiyaç duyulan bütçeyi toplamak için token satmak oluyor. Yani token için bir değer belirlenirken belli başlı bir mantığa ya da hesaba dayanıyormuş gibi gösteriliyor. Ortada henüz ürün yokken ve ekip kendisini kanıtlamamışken gelecekte yapılacağına dair söz verilen bir iş için **milyonlarca USD** değer biçilebiliyor. Bu temelsiz yaklaşımlar yatırımcıların bilgisizliği ile buluşunca var olmayan bir işin bir gecede değeri milyarlarca dolara ulaşabiliyor. Elbette temelsiz değerlemeler, ICO'ların aşırı ilgi görmesi, dolandırıcıların da ilgisini çekiyor.

Yüksek fiyat dalgalanmaları, pazar manipülasyonları gibi yaklaşımlar nedeniyle bir gecede yüksek değerlemelere ulaşan veya ICO öncesinde büyük beklenti ile değeri yükseltilen projeler, daha sonra yatırımcılar için bütünüyle bir zarara dönüşebiliyor.

Öngörülemeyen teknik riskler her zaman mevcuttur. Hassas ve kritik şifreleme yaklaşımları bazı projeler için yazılan akıllı sözleşmelerde açıklara sebep olabilir. Önceki bölümlerde bahsettiğim DAO platformunun başına gelen talihsiz olay, bir kodlama açığından meydana gelmişti hatırlarsanız. Hatta Ethereum ağı bu açığı kapatmak için çatallanma yaşamak durumunda kalmıştı.

Bir diğer önemli problem ise ICO yapılan ağların teknik altyapılarının direnç değerlerinin henüz net bir şekilde **teste tabii tutulamaması** diyebiliriz. Örneğin Ethereum platformu üzerinde gerçekleştirilen ICO'lardan bir tanesi (*CryptoKitties*) o kadar ilgi topladı ki tüm Ethereum ağı çöktü. Yaşanan talebi **EVM** (*Ethereum Virtual Machine*) yorumlayamadı ve ağdaki tıkanma 1 haftalık yoğun çalışma sonucunda çözülebildi.

Bunlar bir yana dursun, bazen de art niyet olmamasına rağmen paranızı kaybedebilirsiniz. *Nasıl mı?* Basit! ICO ve kripto paralar milyar dolar değerinde döngüler yakalamaya başlayınca birçok kestirmeden zengin olma heveslisi, bu alana el attı. Ekonomi ve iktisat bilgileri yeterli olmakla birlikte Blockchain ve kripto paralar hakkında bilgi sahibi değillerdi. Kurdukları yazılım ekipleri ise işlerinde iyi olmasına rağmen çok yeni bir teknoloji olan Blockchain alanında deneyimsiz ve bilgisizlerdi. İş fikirlerinin Blockchain alanına aktarılıp aktarılamayacağını sorgulayamadılar.

Sonuçta teknik dokümanlarını ve yol haritalarını hayata geçiremediler ve yatırımcılardan ICO süreçlerinde topladıkları paralarla birlikte batıp gittiler. Bir ICO projesine yatırım yapmadan önce projenin gerçekliğini ve hayata geçirilebilirliğini **çok iyi analiz etmeniz gerektiğ**ini hatırlatmak isterim.

Aklınızda yanlış bir algı oluşmasın, ICO dediğimiz yapı para tuzağı ya da yeni nesil bir dolandırıcılık yöntemi değildir. Sadece ICO süreçlerine yatırımcı olarak dâhil olduğunuzda yasaların yatırdığınız parayı güvence altına almadığını bilmeniz önemli.

Yasalar, ICO süreçlerine Bitcoin ya da Ether gibi kripto para ile katılabildiği sürece hiç müdahil olmuyorlar. Ancak ICO süreci fiyat para birimleri (*TL, USD ya da EUR gibi*) ile yapıldığında durum farklı oluyor. O zaman ilgili merciler yasal süreçler başlatıyor. Bu yüzden de ICO ve IEO gibi yapılarda yatırım yapmak için kripto para sahibi olmanız şart.

BLOCKCHAIN VE GÜVENLİK

4



Şekil 1.54: Hacker dediğimiz tayfa gözünü Blockchain ağlarına dikti. Bakalım başarılı olabilecekler mi?

Blockchain hakkında basında yazılıp çizilen birçok farklı konu var. Ancak bir tanesi var ki geliştiricisinden yatırımcısına kripto para dünyasındaki hemen herkesi korkutmaya yetiyor. Evet, güvenlik açıklarından bahsediyorum.

Birçok siber güvenlik uzmanı Blockchain ağının belli başlı güvenlik açıklarına sahip olduğunu ve şartların olgunlaşması durumunda art niyetli kişilerin bu açıkları kullanarak ağda manipülasyon yapabileceğini savunuyor. Şimdi, burada **önemli bir ayrıntı** var. Blockchain ağındaki olası güvenlik açıkları derken aslında Blockchain ağının mimari yapısından, yani **kendisinden** bahsediyoruz. Onu oluşturan düğüm mantığını ve mutabakat yöntemlerini kastediyoruz. *Kafanız karıştı değil mi?* O zaman detaylandıralım.

Yüzde 51 Saldırısı

Bu tehdit, direkt olarak Bitcoin ve mutabakat metodu olarak Proof of Work kullanan diğer kripto paraları hedef alıyor. **Açık** yerine **tehdit** dediğime dikkat edin. Bu kelimeyi özel olarak seçtim. Çünkü yüzde 51 denilen "**kurgusal**" saldırı, aslında bir açığı kullanmıyor. Aksine, fiilî olarak ağdaki düğümlerin yüzde 51'ini kontrol altına almaktan bahsediyoruz. Evet, evet ağdaki düğümlerin yüzde 51'ini yönetmek… Bir başka deyişle ağın madencilik yapısında kullanılan hash gücünün yüzde 51'ini **tek elde toplamak!** Üstelik bu mümkün. *Peki, yüzde 51'i tek elde topladığımızda ne olabilir?* Ağda yeni blok onaylama işini tamamen elimize alabiliriz. Böylece istediğimiz blokları onaylayabiliriz. Mesela ağa bizim yerleştirebileceğimiz sahte bloklarla başlarız işe. Çünkü bu hash gücü ile madenciliğin de söz sahibi biz oluyoruz. Bununla kalmayıp ağı çatallayabiliriz mesela… Hem de istersek geçici olarak istersek de kalıcı olarak çatallarız. *Kulağa felaket gibi geliyor değil mi?* Ama bu fikri hayata geçirmeye çalıştığınızda bazı küçük pürüzlerle karşılaşıyorsunuz. İlk pürüz şu: *Hangi ağ bütünlüğünün yüzde 51'i?*

Bu ilginç bir konu ve daha ilginci ise bu sorunu ortaya bir açık olarak atan siber güvenlik uzmanlarının hiçbirisinin ağın boyutlarını ve **ne hızda ve ne sürede genişlediğini** göz önüne almıyor olması. Ağdaki düğüm sayısı anbean tutarlılık göstermez. Düğümler herhangi bir nedenden dolayı ağdan kopup sonra tekrar bağlanabilirler. Bu nedenle ağdaki düğüm sayısını ya da hash gücünü gerçekçi olarak sadece anlık alabiliriz. O an elimizdeki hash gücü ağın gücünün yüzde 51'ini buluyorsa saldırıyı başlatabiliriz. Tabii o sırada ağa yeni katılanlar da olacağını farz etmeliyiz. Genellikle her dakika yeni katılanlar olur ve bu ağın toplam düğüm sayısıyla hash gücünü **sürekli değişken** kılar.

Yani elimizdeki hash gücünün ağın yüzde 51'ini barındırdığı o anı yakalamak oldukça zor bir durumdur. Yakaladıktan sonra (*Bitcoin ağını baz alıyorum*) değişiklik yapabilmek için **10 dakika** süremiz başlıyor. Çünkü ağımız 10 dakikada bir blok üretiyor. Bu noktada ağa, o 10 dakikalık süre içerisinde kimsenin dâhil olmaması da ayrıca önemli. Çünkü kontrolünü elimizde tuttuğumuz hash gücü azalmamalı. İşin güzel tarafı ağda geriye kalan yüzde 49'luk kesim. Bu kesim bir düğümün ağın yüzde 51'ini elinde tuttuğunu fark ettiği an **zorunlu bir çatallanma** başlayacaktır. Bunu fark etme süreleri, en iyi ihtimalle 3 blok oluşturma süresi olan **30 dakika** olabilir. Çünkü ağda işlemler bloklar içerisine önceliklere göre yerleştirilir. Önceliği düşük olan işlemler, sırada en fazla (*istisnalar hariç*) 3 blok bekleyebilir. Daha uzun beklediğinde (*ki ağın yüzde 49'luk kısmına ait işlemlerin tamamının ya da tamamına yakınının beklediğini varsaymamız gerekiyor*) kontrolün **çoğuldan tekile** indiği hemen anlaşılır. Ancak saldırgan kendisini gizlemek için diğer blokları da her 3 blokta bir onaylamaya kalkarsa o zaman zaten kendi oluşturduğu bloklar zincirde kalmakla birlikte diğer bloklar da zincire eklenecek ve bu defa mutabakat yapısı çakışan blokları ayırarak ağı onarmaya çalışacaktır. Çakışan blokların sayısı belli bir rakamı bulduğunda yüzde 49'luk kesim yine bir şeylerin ters gittiğini anlayacaktır ve sonuç yine çatallanma ile ağın dürüst tarafını yola devam edecek şekilde kurtaracaktır.

Bir başka pürüz de ağın yüzde 51'ini elinde tutarak aslında ağı yönetmekte olan bir saldırganın, cebren aldığı bu yetkiyi **ne için** kullanacağını ön görememizdir. Yani bir düşünün, Bitcoin ağını yönetiyorsunuz. Blok onaylama işini tekelinize almışsınız ve tüm onaylama ücretleri sizin cüzdanınıza akıyor. Üstelik ağdaki para arz gücünü de elinizde tuttuğunuzdan blok başı ödüller de sizin cüzdanınıza akıyor. İşin özü, *elinize geçirdiğiniz altın yumurtlayan tavuğu, ağı manipüle ederek kesip atar mıydınız? Yoksa zenginliğinize zenginlik katmaya devam mı ederdiniz?* Eğer ilk seçeneği tercih ederseniz ağ manipüle olur, bu durum ortaya çıkar, halkın ağa olan güveni sarsılır ve Bitcoin'in piyasa değeri sıfıra düşer. Siz de elinizdeki her şeyi kaybedersiniz. Ağın geri kalanındakilerle birlikte tabii… Ama ikinci seçeneği seçerseniz zenginliğiniz katlanarak yükselir (*yüzde 51'i elinizde tuttuğunuz sürece*) ve ağ çalışmaya devam eder. Siz de neticede ağa ve geriye kalan yüzde 49'a hizmet etmeye başlamış olursunuz.

Yüzde 51 senaryosundaki bu çıkarımlar bana aittir; ama Satoshi, Bitcoin'in White Paper'ini yazarken güvenlik başlığı altına **benzer çıkarımlar** örneklemiştir. Üstelik ağın hash gücü düşük olduğu dönemlerde bile Bitcoin yüzde 51 saldırısına maruz kalmamıştır. *Bu biraz düşündürücü değil mi sizce?* Yüzde 51 saldırısına Proof of Work yöntemine göre daha zayıf olan bir mutabakat yöntemi var: **Proof of Stake**.

Bu mutabakat yöntemi, hatırlarsanız, madencilik yapısı olmadığı için kripto paranın tamamı **en baştan** (*ilk bloktan*) arz edilmiş oluyordu. Üstelik genellikle bu arz yöntemi **ICO eliyle** yapılıyordu. Bu tip yapılarda blok oluşturmak ve blok onaylamak için arz edilmiş kripto paranın cüzdanlardaki miktarı baz alınıyordu. Yani cüzdanında kripto parası çok olan, ağda söz sahibi oluyordu. Şimdi bunu biraz genişletelim ve bir cüzdanda arz edilmiş para miktarını yüzde 51'inin olduğunu varsayalım. *Sonuç ne olurdu?* Evet, felaket. *Peki, bu mümkün mü?* Aslında mümkün. ICO süreçlerinin en eksi tarafı piyasada **balina** statüsünde oyuncular yaratmasıdır. Ağın yüzde 51'ini daha ICO aşamasında kaybetmek mümkün. İkinci aşama olan borsalarda yapılan al-sat işlemleri de bunun için uygun bir ortam sağlar. ICO aşamasında balina konumuna gelen bir yatırımcı borsadaki fiyatlarda dilediği gibi "**pump**" ve "**dump**" yapabileceği için cüzdanındaki para miktarını kolayca artırabilir. Buna bir önlem olarak geliştiriciler yüzde 51'lik payı güven altına almak isteyebilirler. Ancak o zaman ağın düğümlerini oluşturan yatırımcıların güvenini kaybederler. Çünkü yüzde 51 saldırısını kendileri yapmış olurlar. İyi niyetli ya da kötü niyetli olmaları fark etmez.

Coin Burn dediğimiz **arz miktarında imha** yoluyla azaltmaya gitme konusu da yüzde 51'in önünü açan durumların başında geliyor. Geliştiriciler bu yola giderken çok iyi hesap yapmalılar. Dolgun cüzdanlara sahip balina yatırımcılardan birisine yüzde 51 yetkilerini verebilirler.

Gördüğünüz gibi yüzde 51 saldırısı denilen yöntem tamamen gerçek olmakla birlikte, gerçekleştirmesi zor; gerçekleştirdikten sonra o güçle ne yapılacağına karar vermesi daha da zor bir durumdur. Kimsenin bindiği dalı kesmek istemeyeceğini de unutmayalım.

Kuantum İşlemci Riski

Yüzde 51 gibi bir başka sıkıntılı kötü durum senaryosu da önümüzdeki on yıl içerisinde hayatımıza girmesi beklenen **kuantum bilgisayarlar** ve onların yaratacağı **yüksek işlem gücü**. Bu tarz bir işlem gücü kullanarak ağdaki hash gücünün tamamından bile daha yüksek bir güç yaratılabileceği savunuluyor. Böylesi bir durum matematiksel olarak mümkün. Bu işlemci gücü sayesinde ağın tamamını manipüle etmek olasıdır. Ancak bu aşamada yine ağların blok üretim sürelerinin içerisinde kalıyoruz. Yani bizim cihaz hızımızın pek bir önemi olmuyor. Zira blok başı Bitcoin için **10**, Litecoin için **2,5 dakika** beklemek zorundayız. Ayrıca böylesi bir hash gücüyle blokları büyük bir hızla çözebileceğiz diye düşünebilirsiniz. Ancak bu yarattığımız hash gücü o kadar muazzam olacak ki, mutabakat yapımız ve ağımızın karakteristik eşitlikçi mantığı zorluk derecemizi erişemeyeceğimiz bir noktaya çıkartacak. Bu durum kuantum işlemciyi zorlamaya başlayacağından, diğer madencileri de hash güçlerini yükseltmeye zorlayacak. Sonuç olarak kısa süre içerisinde ağdaki kuantum işlemci gücüne sahip düğümler artacak. Yani bu saldırı türü de mümkün olmasına rağmen **sürdürülebilir değil**.

Son Söz

Gördüğünüz gibi Blockchain ağ yapısı sözde açıklara sahip olmakla birlikte hâlâ güvenli ve yıkılmaz bir noktada duruyor. Kitabımın önceki bölümlerinde DAO platformunun başına gelenleri örnek olarak paylaşmıştım. Bu örneğe hemen her Blockchain kaynağında rastlarsınız. Bunun temel nedeni, **benzeri ikinci bir vakanın olmaması**dır. İlgili bölümde de anlattığım gibi DAO platformu aslen hack'lenmemiştir. Ethereum ağı üzerinde yazılan bir akıllı sözleşmenin protokol açığını kullanılarak para, başka hesaba aktarılmıştır. Çatallanma ile sonuçlanan çözüm süreci sonrası benzeri bir sıkıntı yaşanmadı.

Ancak yine de kripto paralar milyon hatta milyar dolarlık hırsızlıklar ve hack hikayeleriyle süslü bir geçmişe sahip. Bu hikayeler de ne yazık ki gerçek ve kripto paraların değerlerini bir hayli negatif etkileyebiliyor. Ancak bu tarz haberlere konu olan saldırılarda Blockchain ile ilgili bir durum ya da risk yoktur. Genellikle borsaların bulunduğu sunucuların hack'lenmesi sonucunda yaşanan hırsızlıklardan bahsediyoruz. Standart sunucu hack senaryoları bunlar. Ancak sonuçta para kayıpları ciddi boyutlarda seyrediyor.

Ancak sonradan yapılan soruşturmaların sonuçlarına göre bu tarz hırsızlıkların çok küçük bir kısmı hack suçlarına dayanıyor. Geri kalanı tamamen kurgu ve düzmece hırsızlıklar. Üstelik suçluların büyük bir kısmı ekip içerisinden çıkıyor. Yani erişim haklarını kötüye kullanan çalışanlar ya da şirket yetkililerinden bahsediyoruz. Bunu da unutmamak gerekiyor.

Blockchain ve kripto paralarla ilgili söyleyeceklerimiz şimdilik bu kadar. Artık Blockchain ile ilgili kod yazma zamanı geldi. Hadi kendi Blockchain ağımızı oluşturalım.

WORKSHOP: KOD YAZMA ZAMANI!

5

Blockchain nedir, *ne değildir* anladığımıza göre artık mevcut işimizi ya da yeni iş fikirlerimizi kurgularken ona da yer verebiliriz. *Peki, nereden başlamalıyız? Hangi donanımlara ve geliştirme dillerine sahip olmalıyız? Blockchain oluşturmak zor mu? Hangi Blockchain yapısı fikrime uygun?* Kafanızdaki soru işaretlerini daha fazla bekletmeden uygulamalı workshop bölümümüze geçelim.

Nasıl Bir Blockchain?

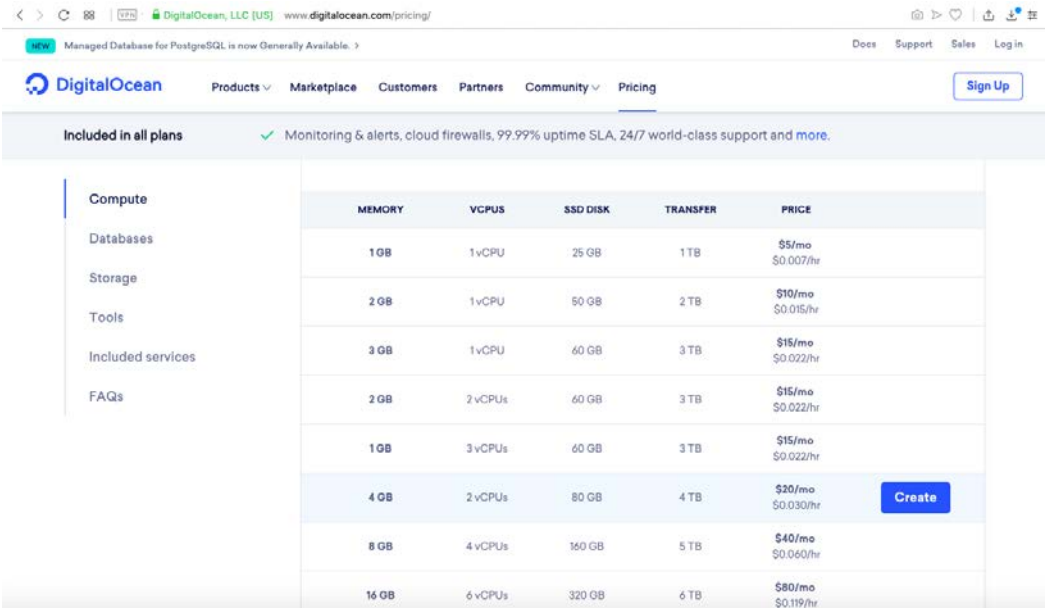
Sevgili okurlar, bu sorunun cevabı ne yazık ki bende yok. Yani iş fikrinizi veya projenizi bilmeden nasıl bir Blockchain ağına ihtiyaç duyacağınızı söyleyemem. Ancak, sahip olduğunuz iş fikri kendi Blockchain ağınızın olmasını gerektiriyorsa bir tane yapmak durumundasınız. Tabii bu; **emek**, **zaman** ve **para** demek oluyor. Eğer mevcut işinizi taşımaktan bahsediyorsak daha çok emek, daha çok zaman ve daha çok para demek oluyor. Fakat projenizin kendi özel ağına ihtiyacı yoksa **Ethereum** üzerinde akıllı sözleşmeler ile kolayca ve çok küçük maliyetler karşılığı yan ağınıza ve token'lerinize kavuşabilirsiniz.

Kendi ağınızı kurmak istiyorsanız benim tavsiyem Proof of Work mutabakat yöntemi ve Bitcoin kodlarını ya da Litecoin kodlarını kullanmanız yönünde. Amacımız, klon yaratmak olacak tabii; ama siz kafanızdaki modele uygun olarak kodlarda değişiklikler yapacaksınız.

Eğer Ethereum üzerinde akıllı sözleşmeler ile projenizi hayata geçireceksiniz **Solidity dili**, **bir bilgisayar** ve **biraz Ether parası**na ihtiyacınız olacak. Elinizde hiç sermaye yoksa Solidity dilini öğrenene kadar musluk sitelerini

kullanmanızı öneririm. Böylece elinizde GAS ödemeleri için biraz para olacaktır. Musluk sitelerini "**faucet**" anahtar sözcüğüyle arama yaparak bulabilirsiniz. Bir yerden başlamak için **Coinpot ailesi** musluk sitelerini kullanabilirsiniz. Bunlar *Moonbitcoin*, *Moonlitecoin*, *Moondogecoin*, *Moonbitcoincash* ve *Moondash* olarak sıralanır.

İHTİYAÇ LİSTESİ

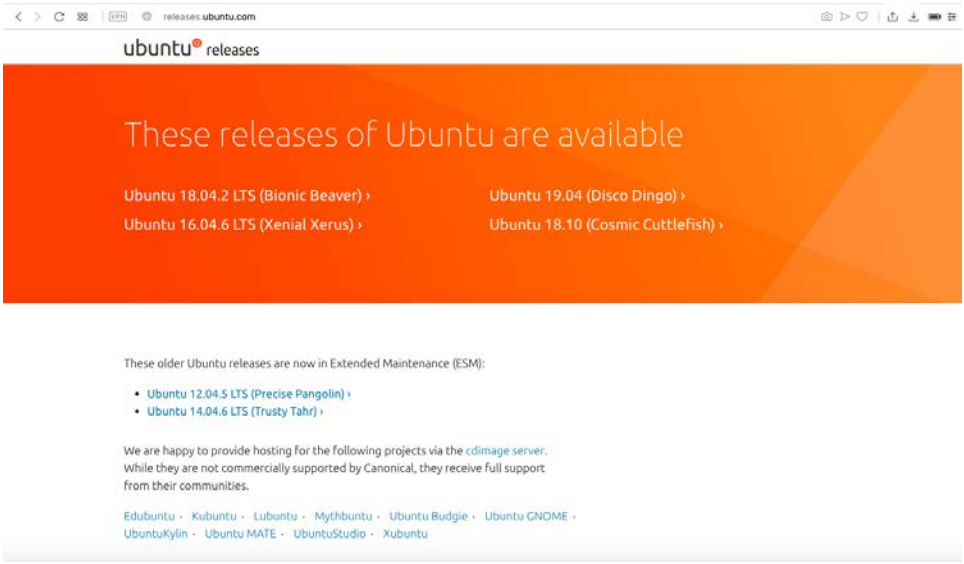


Şekil 1.55: DigitalOcean, uygun maliyetli bulut tabanlı sunucu çözümleri sunuyor. Root erişimi verdiği için Blockchain üzerinde çalışan geliştiricilerin gözdesi durumunda.

Bu aşamada bazı ihtiyaçlarımız olacak. Bunların başında bir web sunucusu gerekiyor. Sunucu olarak **DigitalOcean**, **Godaddy** ya da **Amazon** en çok tercih edilen servis sağlayıcılarıdır diyebilirim. Bunlar içerisinde en uygun maliyetli olanı tercih edebilirsiniz. Eğer kafanızdaki yapıya uygun bir modeli hâlihazırda hayata geçirmek istiyorsanız daha yüksek yapılı bir sunucu tercih edebilirsiniz. Ama sadece kendinizi geliştirmek ve Blockchain hakkında programlama deneyimi kazanmak niyetindeyseniz sunucu kiralamak yerine, evinizde atıl bulunan bilgisayarları kullanabilir ya da yüksek yapılı bir bilgisayar üzerinde **sanal makineler** oluşturmayı deneyebilirsiniz. Bu aşamada tercihi size bırakıyorum. Nihayetinde elimizde olmasını istediğimiz sistem şu özelliklere

sahip olmalı: Minimum 2 GHz CPU, 4 GB RAM ve en az 20 GB sabit disk alanı. Bu donanım üzerinde **GNU/Linux** işletim sistemi olması işinizi çok ama çok kolaylaştıracaktır. GNU/Linux tarafında deneyiminiz yoksa Ubuntu 16 da işinizi görür. Ubuntu 12 ya da 14 de kullanabilirsiniz (*birçok geliştirici 12 sürümünü tercih ediyor*). Bu sürümleri Ubuntu sitesi üzerinden eski sürümler başlığı altında bulabilirsiniz. Eski sürümleri tercih ediyor olmamızın nedeni, ihtiyaç duyacağımız bazı kütüphane ve uygulamaların yeni sürümlerin repolarına henüz eklenmemiş olmasıdır.

Bu sunucu makinemizi hazırladıktan sonra geliştirme yapacağımız bilgisayar üzerinden sunucumuza **SSH** bağlantısı sağlayacağız. O yüzden sunucu kurulumu biter bitmez ilk iş olarak **SSH Server kurulumu** yapmanız gerekecek. Biz Blockchain ağını bir sunucu üzerinde başlatmış olacağız. Mutabakat yapımızın oluşturacağı zincirin baş halkası bu sunucu olacak. O yüzden sunucumuzun IP adresini yazılımımızın içerisine kaydedeceğiz. Böylece gerek madencilerimiz gerekse cüzdan sahiplerimiz güncel blok listelerini bu IP üzerinden mutabakata sokacaklar. Projeniz büyüdükçe ağı güvende tutmak için daha çok sunucuya ve daha çok IP adresine ihtiyacınız olacak. Buna **kendinizi** ve **bütçenizi** hazırlayın.



Şekil 1.56: Geliştiricilerin en yakın dostu Ubuntu işletim sistemidir. Kripto, para ve Blockchain odaklı çok fazla kütüphaneye sahiptir. 12.04 sürümü Blockchain için biçilmiş kaftandır. Derleme sırasında ihtiyaç duyacağımız bazı paketler yeni sürümlerde bulunmuyor.

Geliştirme yapacağımız bilgisayarımızda hangi işletim sisteminin yüklü olduğu bizim için önemli değil. Bir internet bağlantısı ve metin editörü olması yeterli. Bir **C IDE'si** olması çok daha iyi olur tabii. Ancak ağı oluşturacağınız sunucu ilk etapta elinizin altındaki mevcut bir makine olacaksa ikinci bir geliştirme bilgisayarına ihtiyacınız yok demektir. Direkt olarak sunucu üzerinde çalışabilirsiniz. Ancak bu durumda sunucu kurulumu yaparken mutlaka grafiksel bir masaüstü ortamı da kurun. Ubuntu (*diğer dağıtımların sunucu sürümleri de böyledir*) sunucu sürümü **grafiksel arabirim** içermez. GNU/Linux alanında bilgi birikiminiz yeterliyse konsol üzerinde uygulamalarınızı gerçekleştirebilirsiniz tabii.

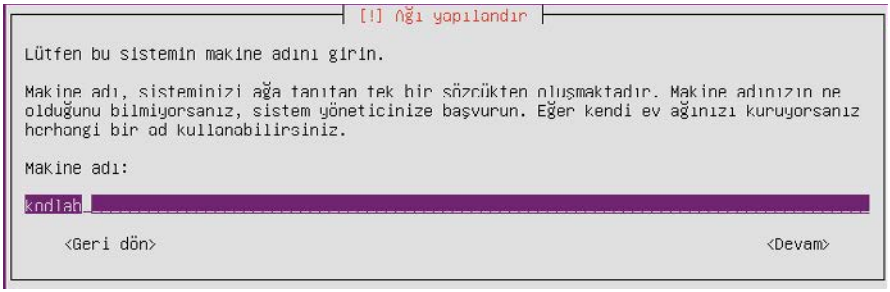
Kitabın *mutabakat yöntemleri*ni anlattığım bölümünden de hatırlayacağınız gibi benim şahsi tercihim **Proof of Work**. Bu noktada Bitcoin ya da Litecoin gibi bir kripto para yapısının Blockchain ağını baz almayı tercih edeceğim. Siz Proof of Stake ya da başka bir mutabakat yöntemini kullanmak istiyorsanız bu yöntemi kullanan bir yapıyı baz alabilirsiniz.

Hash algoritması olarak **SHA türevleri**ni kullanabilirsiniz ya da daha az sistem kaynağı tükettiği için **SCRYPT algoritması**nı tercih edebilirsiniz. Ben herkesin geliştirme yapmak için yüksek yapılı bir sunucu kiralamadığını varsayarak SCRYPT algoritmasını tercih ediyorum. Böylece eski makineleri bu uygulamada kullanacak olan okurlar donanım engeline takılmayacaklar. Eğer ağınızı halka açmak isterseniz madencilik yapınız Bitcoin ağında olduğu gibi **ASIC** cihazlara ihtiyaç duymayacak.

Geldiğim nokta bana Litecoin'i baz almam gerektiğini gösteriyor.

Mevcut bir kripto paranın kodlarını kullanmamızın nedeni algoritma ve mutabakat açısından mesai harcamamamızdır. Bu iki öge dışındaki birçok yeri kendi projemize göre değiştireceğiz zaten. Bu tarz yapılar *GPL* ya da *MIT* gibi açık kaynak kod lisansları kullandıklarından, yaptığımız şey tamamen kurallara uygun. Zaten bir sonraki bölümümüz olan akıllı sözleşme geliştirme uygulamalarımızı da **Ethereum** ağı üzerinde yapıyor olacağız.

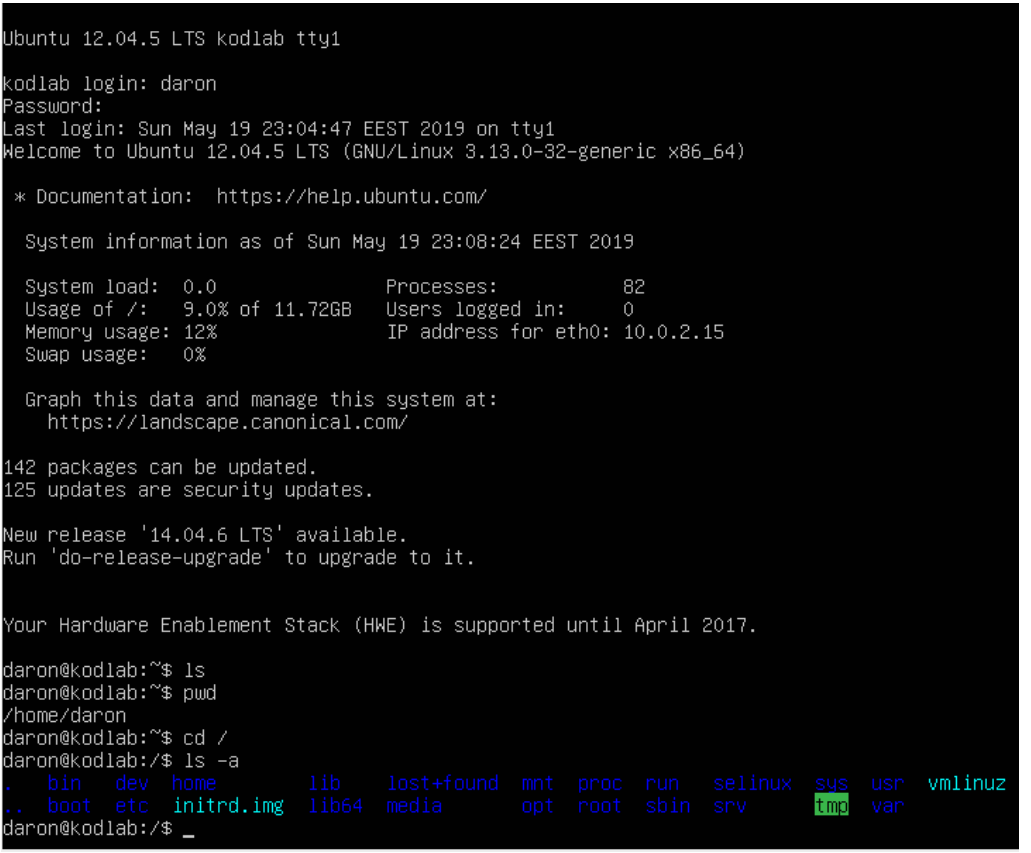
Sunucumuz **Ubuntu Server** olacak demiştim. SSH ile bağlanacağımızdan bahsetmiştim. Okurlarımın ne kadarı Linux konsol komutları hakkında bilgili bilemiyorum. O yüzden en çok kullanacağımız komutların üzerinden kısaca geçmekte fayda görüyorum. Başlayalım.



Şekil 1.57: Ubuntu sunucu sürümü, grafiksel arabirim içermemesine rağmen kurulumu oldukça kolaydır. Özellikle 12.04 sürümü çok az sistem kaynağı tüketir.

Linux Konsol Komutları

GNU/Linux açık kaynak kodlu özgür ve ücretsiz bir işletim sistemidir. Geliştirme yapanlardan tutun da web sunucusu ihtiyaçlarına kadar hemen her alanda kullanılabilen yüksek yapılı bir işletim sistemidir. Üstelik işin niteliğine göre özelleştirilmiş dağıtımlara sahiptir. Kripto para ve Blockchain gibi yapılar geliştirilirken genellikle kütüphane ve yazılım desteği yüzünden Linux işletim sistemi tercih edilir. Biz de geleneği bozmuyor ve Ubuntu üzerinde geliştirmeye başlıyoruz. Yapacağımız konsol işlemlerinde genellikle bu bölümde sıraladığım komutları kullanacağız. Konsol kullanımı konusunda kendinizi yeterli hissediyorsanız bu bölümü atlayabilirsiniz.



Şekil 1.58: Linux üzerinde konsol kullanarak çalışmak ilk başlarda garip gelse de alıştıkça çok seveceğiniz bir eğlence hâline gelecek ve Blockchain geliştirirken uzun saatlerinizi bu tarz bir ekranın başında geçireceksiniz.

PWD

Terminal penceresini çalıştırdığımız zaman, giriş yapmış olduğunuz kullanıcının "**home**" dizininde açılır. Eğer hangi dizin içerisinde olduğumuzu bilmek istiyorsak "**`pwd`**" komutunu kullanırız. Bu komutun çıktısı bize root kullanıcısından başlayarak tam hangi dizinde olduğumuzu gösterir.

LS

`ls` komutu, bulunduğunuz dizinde hangi dosyaların olduğunu bilmek için kullanılır. "**`ls -a`**" ya da "**`ls -all`**" komutunu kullanarak tüm gizli dosyaları görebilirsiniz.

CD

Bir dizine gitmek için kullanılan komuttur. Örneğin, **Home** klasöründeyseniz ve **Downloads** klasörüne gitmek istiyorsanız, "`cd Downloads`" yazabilirsiniz. Unutmayın ki bu komut büyük/küçük harfe duyarlıdır ve klasörün adını tam olarak olduğu gibi yazmanız gerekir. Bulunduğunuz klasörden geri dönmek veya bir üst klasöre çıkmak istiyorsanız "`cd . .`" komutu kullanılır. "`cd /`" komutu ile ana dizinimize dönebiliriz.

MKDIR

Bir klasör veya dizin oluşturmanız gerektiğinde kullanılır. Örneğin "**blockchain**" adlı bir dizin oluşturmak istiyorsanız "`mkdir blockchain`" komutunu yazabilirsiniz.

RMDIR

Bir dizini silmek için kullanılan komuttur. Ancak, `rmdir` yalnızca **boş bir dizini** silmek için kullanılabilir. İçi dolu bir klasörü silmek istiyorsanız komutu "`rm -r klasörünadı`" şeklinde kullanmanız gerekir.

TOUCH

Bir dosya yaratmak için kullanılır. Bu dosya herhangi bir şey olabilir. Örneğin bir **zip** dosyası ya da **txt** dosyasını aynı komutla yaratabilirsiniz.

MAN &-HELP

Bir komutun nasıl kullanılacağı ve hakkında daha fazla bilgi edinmek için `man` komutu kullanılır. Örneğin, "`man touch`" komutunu kullandığınızda "`touch`" komutunun kullanım bilgisi sayfalarını gösterir. Bir komutu yazıp arkasına "`-help`" eklersek yine aynı sonuca ulaşırız.

CP

`Cp` komut satırı üzerinden kopyalama yapmak için kullanılır. İki argüman alır; ilki kopyalanacak **dosyanın konumu**, ikincisi **kopyalanacak** yerdir. Örneğin, "`cp blockchain.doc /home/Daron/blockchain`".

MV

Dosyaları komut satırı yoluyla taşımak için kullanılır. Aynı zamanda bir dosyayı yeniden adlandırmak için `mv` komutunu kullanabiliriz. Örneğin az önce kopyaladığımız **blockchain.doc** dosyasının ismini "**BLOCKCHAIN.doc**" olarak değiştirmek istiyoruz diyelim; "`mv blockchain.doc BLOCKCHAIN.doc`" komutu ile bu işlemi gerçekleştirebiliriz.

NANO VE VI

Linux komut satırı üzerinde hâlihazırda yüklü olan metin düzenleyicilerdir. Nano, programlama dillerin çoğunu tanıyan ve dile özel komutları renklendirme özelliğine sahip güçlü bir metin editörüdür. Vi, nano'dan **daha basittir**. Bu komutla yeni bir dosya oluşturabilir veya bu düzenleyiciyi kullanarak dosyayı değiştirebilirsiniz. Ubuntu üzerinde ön tanımlı olarak nano metin editörü gelir. "`nano deneme.c`" komutunu kullandığınızda **deneme** adlı bir **C** dosyası oluşturmuş olursunuz.

SUDO

Ubuntu gibi Linux dağıtımlarında yaygın olarak kullanılan bir komuttur. **Sudo**, "**SuperUserDo**" kelimesinden gelir. Bu komutu, yapılacak bir işlemde root yetkilerini kullanmak istersek veya idari bir yapıyla erişim gerekiyorsa kullanabiliriz. Örneğin "`sudo su`" komutunu kullanarak sistemde admin yetkisi ile işlem yapmaya başlayabiliriz.

APT-GET

Ubuntu gibi Debian tabanlı hemen her Linux dağıtımında, komut satırından uygulama indirip kurmak için bu komut kullanılır. Genellikle `install` parametresiyle çağırılır.

WGET

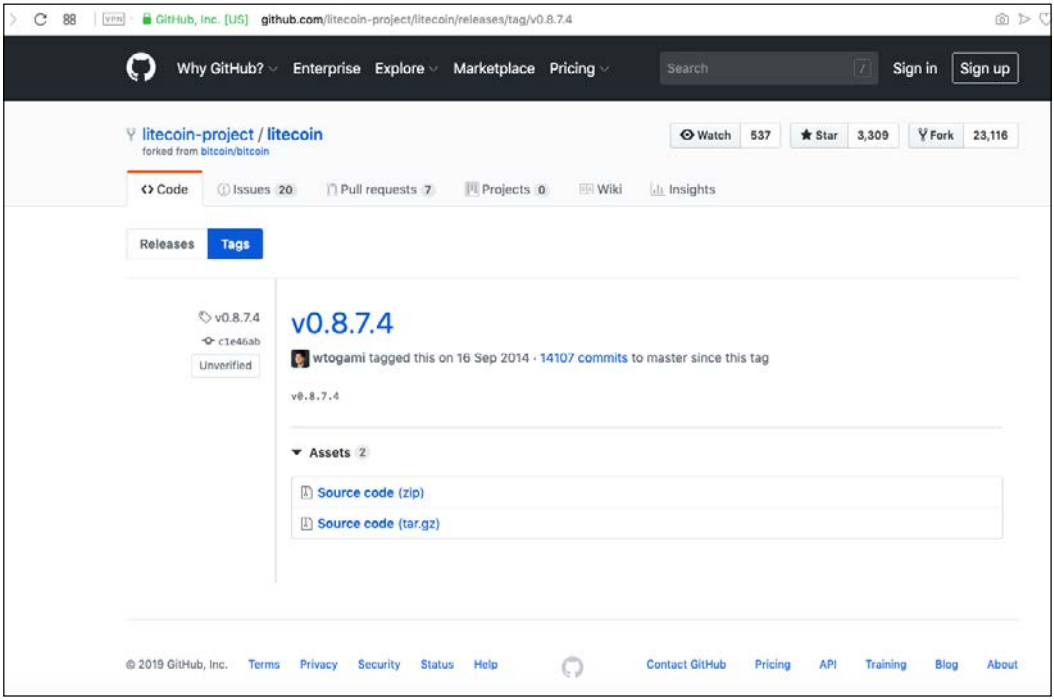
Komut satırında çalışırken internette herhangi bir yerde bulunan ve yolunu bildiğimiz bir dosyayı, çalıştığımız dizine indirmek için kullanılır.

IFCONFIG

Komut satırı üzerinden Network kartlarının bilgilerini ekrana yazdırmamız için kullanılır.

Kodlab Blockchain

Ne yapacağımızı özetleyeyim: Ben burada yayınevine atıfta bulunarak Kodlab adlı bir kripto paranın üzerinde koştuğu, Proof of Work mutabakat yapısı kullanan ve SCRYPT hash algoritmasıyla blok üreten bir Blockchain oluşturacağım.



Şekil 1.59: Sade ve yeni nesil Litecoin teknolojilerinden uzak olduğu için örnek uygulamalarımızı v0.8.7.4 sürümünü temel alarak gerçekleştiriyoruz.

İşe *Github.com* üzerinden Litecoin'in proje sayfasına girmekle başlıyorum. Buradan **0.8.7.4** sürümünü indiriyorum. Bu sürüm, üzerinde **Lightning-Network** gibi karmaşık yapıların eklenmediği yalın ve tek düze bir Litecoin sürümüdür. Dolayısıyla kafanızı karıştırmaya gerek yok. Sizin için pratik olması açısından konsol üzerinde şu komutu kullanmanızı öneririm:

```
sudo wget https://github.com/litecoin-project/litecoin/archive/
v0.8.7.4.zip
```

Bu komutta kullandığımız "**sudo**", bizi sistem yöneticisi yetkileriyle donatıyor. Hemen arkasından gelen "**wget**" komutu ise, kendisinden sonra gelen dosyayı üzerinde bulunduğu uzak sunucudan indirmemizi sağlıyor.

Dosyamız, komutu hangi dizin içerisinde çalıştırdıysak o dizin içerisine indirilir. Dosyayı **tar.gz** sıkıştırma formatında indirdiyseniz;

```
sudo tar -xzvf litecoin-0.8.7.4.tar.gz
```

konsol komutunu kullanarak içerisindeki kaynak kodları dışarıya aktarmış olursunuz. Eğer **zip** dosyası olarak indirdiyseniz öncelikle zip uygulamasını sisteminize kurmanız gerekiyor. Çünkü zip uygulaması Ubuntu üzerinde ön tanımlı olarak gelmiyor. Bunun için uygulama kurarken kullandığımız **apt-get** yapısını kullanacağız.

```
sudo apt-get install zip
```

Uygulamamız kurulduktan sonra zip dosyalarını açmak için,

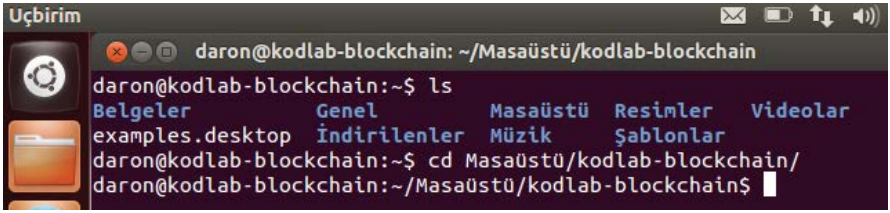
```
sudo unzip litecoin-0.8.7.4.zip
```

komutunu kullanıyoruz. Artık kaynak kodlarımızı klasör olarak çıkartmış olduk. Şimdi kaynak kodlarımız üzerinde birtakım değişiklikler yapmaya başlayabiliriz. Ancak önce dizinin adını değiştirip "**kodlab-blockchain**" yapalım.



Şekil 1.60: İndirdiğimiz kaynak kod dosyasını açıyoruz ve bir kopyasını oluşturarak adını "kodlab-blockchain" yapıyoruz. Siz başka bir isim de kullanabilirsiniz.

Şimdi terminal üzerinden yeni oluşturduğumuz **kodlab-blockchain** dizinine giriyoruz. Bu işlem için terminalde **cd / "dosyaları indirdiğiniz dizin" / kodlab-blockchain** komutunu kullanıyoruz. Tırnak içerisine aldığım kısma, **kodlab-blockchain** hangi dizin altında bulunuyorsa o dizini yazıyoruz.



Şekil 1.61: Benim dizinim Masaüstünde bulunduğu için "cd /Masaüstü/kodlab-blockchain" komutunu kullanıyorum.

Şimdi artık kaynak kodlarımızı özelleştirmeye başlayacağız. Eğer **kodlab-blockchain** dizini içerisindeyken "`ls`" komutu kullanırsanız dizinin içerisinde birçok dosya ve klasör olduğunu görürsünüz. Her bir dosya içerisinden Litecoin'e ait olan karakteristik tanımlamaları değiştirmemiz gerekiyor. Bu işlemi terminal kullanmadan yapma şansımız ne yazık ki yok. Terminal kullanmadan yapmaya kalkarsak her bir dosyaya tek tek ve satır satır uygulama yapmamız gerekir. Bu da çok uzun soluklu ve hataya sebebiyet verecek bir durum olurdu. O yüzden terminalin gücünü kullanarak tüm klasör ve dosyalara tek seferde ulaşarak ilgili değişiklikleri yapacağız. Başlayalım. Önce dizin içerisindeki Litecoin geçen yerleri bulup bunları **Kodlabcoin** ile değiştireceğiz. Bunun için şu komut dizgisini kullanıyoruz:

```
find ./ -type f -readable -writable -exec sed -i "s/Litecoin/
Kodlabcoin/g" {} \;
```

Terminal işlemi gerçekleştirmek için kısa bir süre bekleyebilir. İşlem tamamlanınca yanıp sönen komut satırı imlecimiz geri dönmüş olacak ve diğer komut dizgisini girebileceğiz. Devam ediyoruz.

```
find ./ -type f -readable -writable -exec sed -i "s/LiteCoin/
KodlabCoin/g" {} \;
```

Bu komut ile dizin içerisindeki tüm dosya ve klasörlerde `LiteCoin` anahtar sözcüğünü bulup `KodlabCoin` ile değiştirmiş oluyoruz.

Devam ediyoruz.

```
find ./ -type f -readable -writable -exec sed -i "s/litecoin/
kodlabcoin/g" {} \;
```

Bu komut, tahmin edebileceğiniz gibi, Litecoin geçen yerleri tespit edip **`litecoin`** ile **`kodlabcoin`**'i değiştiriyor. Devam ediyoruz.

```
find ./ -type f -readable -writable -exec sed -i "s/litecoind/
kodlabcoind/g" {} \;
```

Komut satırı üzerinden kripto paralarla çalışırken ilerleyen adımlarda derleyerek oluşturacağımız çekirdek uygulamayı kullanırız. Bu uygulamaya web katmanından da erişmeniz mümkün olduğu için muazzam bir kullanım esnekliği ve kolaylığı sunar. Kripto paraların çekirdek uygulamaları genellikle "**kriptoparaadıd**" şeklinde adlandırılır. Adın sonuna gelen "**d**" bize dosyanın aslında çalıştırılabilir bir **daemon** dosya olduğunu gösterir (*binary formunda daemon*). Bu adımda kaynak dosyamızın kodları içerisindeki "**`litecoind`**" sözcüklerini bulup "**`kodlabcoind`**" ile değiştirmiş olduk. Şimdi son değişikliğimizi yapmaya geldi sıra.

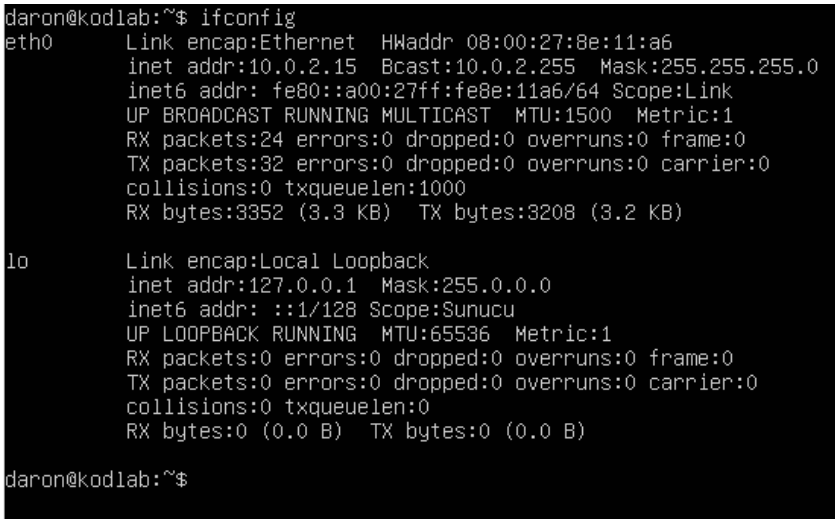
```
find ./ -type f -readable -writable -exec sed -i "s/LTC/KOD/g" {} \;
```

Bu komut dizgisi ile kripto paramızın kısaltılmış kur tanımı olan KOD ile Litecoin'in kur tanımlayıcısı olan LTC'yi değiştiriyoruz. Bu adımdaki kısaltmayı ben uyduruyorum. Gerçek bir uygulama yapıyor olsaydık mevcut kripto paralar tarafından kullanılmayan bir kur tanımlama kısaltması bulmamız gerekecekti. İnternet üzerinden küçük bir aramayla kafanızdaki kısaltmanın kullanımda olup olmadığını bulabilirsiniz.

Evet, yapmamız gereken tüm değişiklik ve özelleştirmeleri tamamladık. Artık **kodlab-blockchain** dizini altında bize ait ama karakteristik olarak Litecoin ile birebir aynı olan bir Blockchain ağ kaynak kodu var. Artık projemizin özelinde ve iş modelimize bağlı olarak kodumuzda düzenlemeler yapacağız. Ben bu örnek uygulamamda uçtan uca para transferi amacıyla kullanılacak bir Blockchain kurgulayacağım.

Bu tarz bir yapı için iş modelime belli başlı kriterler belirliyorum. Mesela hızlı olması oldukça önemli. Bildiğiniz gibi Bitcoin **10 dakikada bir blok** üretiyor. Bu nedenle onay sürecini göz önüne aldığımızda bir gönderimin onaylanarak karşı tarafa geçmesi için 30 dakikaya ihtiyacımız oluyor (*mutabakat yapımız minimum 3 onay istediğinden dolayı*). Litecoin'de ise blok üretme süreci 2,5 dakika olduğundan 7,5 dakika içerisinde gönderimimiz karşı tarafın cüzdanında harcanabilir hâle geliyor. Ancak ben daha hızlı bir ağ düşünüyorum. Mesela

blok süresi 30 saniye olan bir ağ. Adres başlangıcı özel olmalı. **L** ile başlayan adresler genellikle Litecoin tarafından kullanıldığı için bunu değiştirmem gerekli. Ben **K** ile başlayan adresler oluşturmayı hedefliyorum. Bunun gibi biraz model biraz makyaj üzerine odaklı kriterlerinizi belirlemeniz gerekli. Tabii en önemlisi kurguladığımız zinciri bağımsız bir Blockchain hâline getirebilmek için bir IP adresine ihtiyacımız olacak. Bu IP adresi bizim zincirimizin ilk halkası olacak. Mutabakat yapımız için referans olan halka **ilk halka**dır. O yüzden önemli. Eğer kullanıma açacağınız bir kripto para hazırlığı içerisindeyseniz şimdiden bir IP adresi almanız gerekli. Ama amacınız deneme yapmaksa yerel **IP adresi** işinizi görecektir.



Şekil 1.62: Linux terminali üzerinde çalışırken IP adresimizi öğrenmek için "ifconfig" komutunu kullanıyoruz.

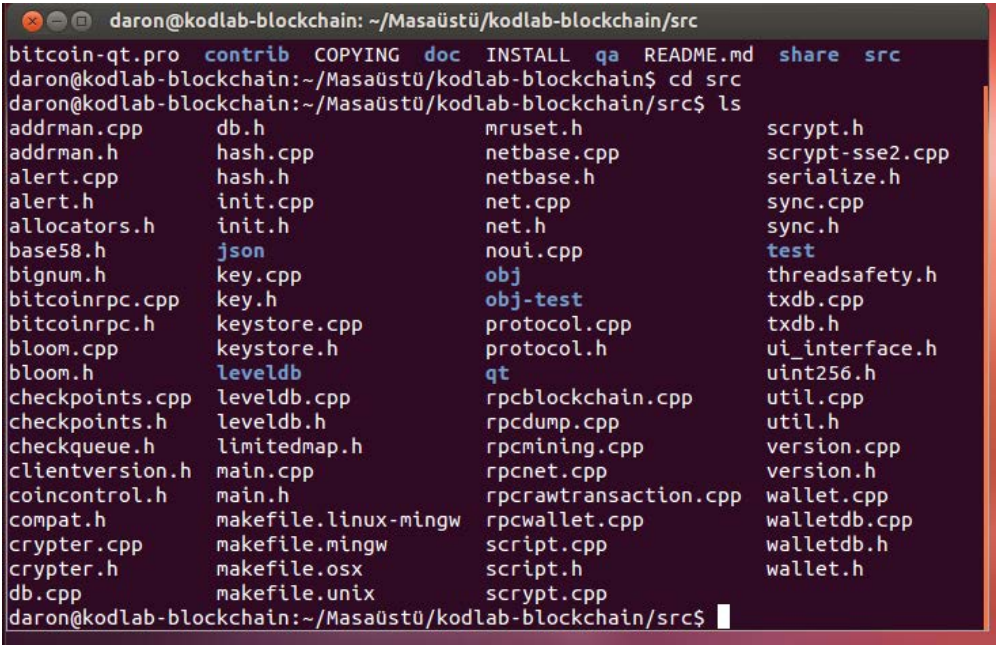
Üzerinde çalıştığınız yerel makinenin IP adresini öğrenmek için terminalde "**`ifconfig`**" komutunu çalıştırıyoruz.

```
sudo ifconfig
```

Bu komutun çıktısı **Şekil 1.62**'deki gibi olacaktır. Komut çıktısı içerisinde "**`eth0`**" başlığına denk gelen paragrafta "**`inet`**" satırında IP adresinizi görebilirsiniz. Bu adresi not edin. İlerleyen adımlarda kullanacağız.

Şimdi Blockchain ağımızın bel kemiğini oluşturan "**main.cpp**" dosyası üzerinde çalışmaya başlayacağız. Bunun için **kodlab-blockchain** dizinimizin altında yer alan **src** klasörüne giriş yapacağız. Terminal üzerinde şu komutu çalıştırıyoruz:

```
cd src
```



Şekil 1.63: Kaynak kodların bulunduğu src klasörüne girmek için "cd src" komutunu çalıştırıyoruz. Bu komutun çalışabilmesi için "kodlab-blockchain" dizini içerisinde olmanız gerekiyor. Klasöre girdikten sonra "ls" komutunu verirseniz tüm kaynak dosyaları listelemiş olursunuz.

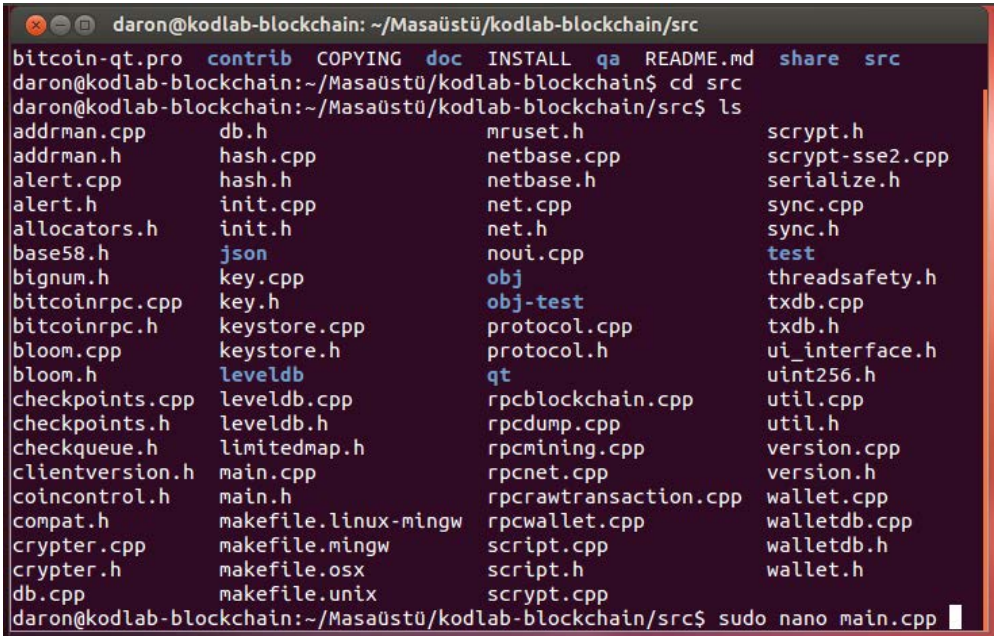
SRC klasörü içerisinde, ağımızı derlerken ihtiyaç duyduğumuz tüm kaynak kod dosyaları mevcut. Bu dosyaların bir kısmının **.h** bir kısmı da **.cpp** uzantılı olduğunu göreceksiniz. Bilmeyen okurlarım için hatırlatayım; bu uzantılar **C++** programlama dilinin kaynak dosyalarına ait. Bu dili bilmiyorsanız ve öğrenmek istiyorsanız piyasada yeteri kadar kaynak olduğunu belirtmek isterim. Bu kitabın konusunu aşmamak için bu dile bir giriş yapmıyorum. Ancak kitaptaki adımları takip ederek C++ dilini bilmeseniz bile kendi Blockchain ağınızı oluşturmuş olacaksınız.

Biz düzenlemelerimizi **.cpp** uzantılı kaynak dosyalarında yapacağız. İşin editleme kısmını çok güçlü bir metin editörü olan **Nano** ile yapacağız. Nano, terminal üzerinde çalışan hem bir kelime işlemci hem de birçok dile destek veren başarılı bir IDE diyebilirim.

SRC klasörü altındayken şu komutu veriyoruz:

```
sudo nano main.cpp
```

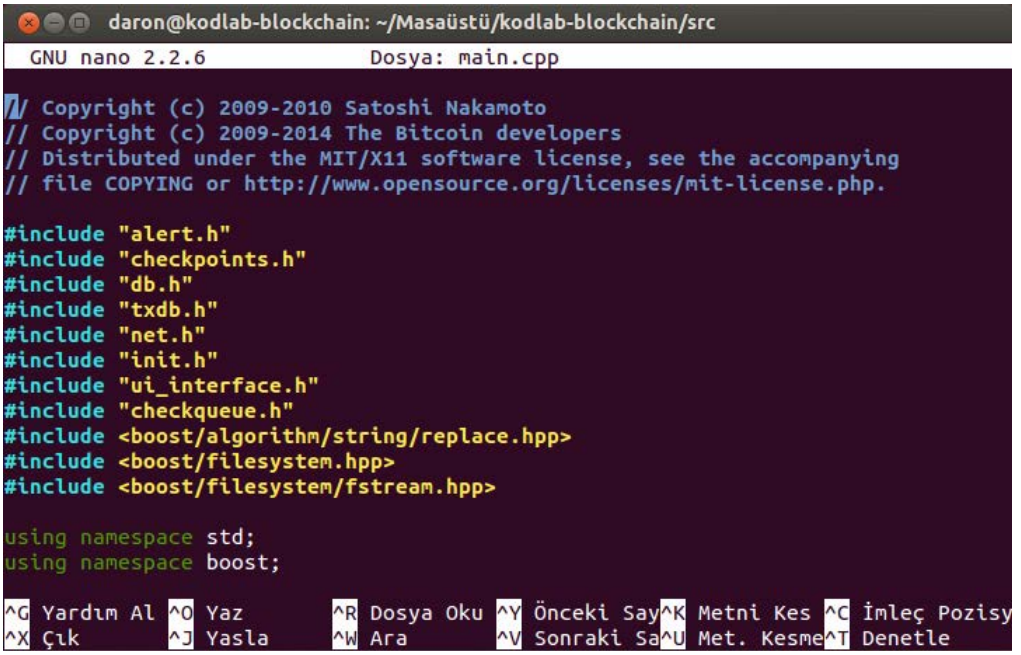
Bu komutu mutlaka "`sudo`" komutuyla verin. **SuperUser** yetkilerini almadan değişiklikleri uygularsanız, kaynak kodların izin yapıları nedeniyle Ubuntu kaydetmenize izin vermeyecektir. Hatırlarsanız `sudo` komutu bize sistemde "**root**" yani SuperUser yetkilerini kazanmamızı sağlıyordu. İkinci komutumuz olan "`nano`", kelime işlemcimizi çağırıyor. Eğer sadece "`sudo nano`" komutunu verirseniz boş bir belge ile Nano uygulaması açılacaktır. Düzenleme yapmak istediğiniz bir dosya varsa komut dizgisine doyanın adını da eklemelisiniz. Yukarıda verdiğimiz komut **main.cpp** dosyasının **Nano** ile açılmasını sağlayacak.



Şekil 1.64: "sudo nano main.cpp" komutunu çalıştırırken kodlab-blockchain dizini altındaki src klasöründe bulunduğunuzdan emin olun. Aksi hâlde komuta bir de dosyanın konumunu eklemeniz gerekecektir. Mesela, "sudo nano src/main.cpp" gibi.

Nano, çağırdığınız metin belgesinin içeriğini yorumlayabilir. Belge bir programlama diline ait ise o dili tanır ve gerekli yordamsal renklendirme şablonlarını uygulamaya alır. Böylece çalışmanız çok pratikleşmiş olur.

Nereye değişiklik yapmanız gerektiğini daha kolay anlayabilirsiniz. Nano ile **main.cpp** kaynak dosyasını çağırdığımızda **Şekil 1.65'**te olduğu gibi bir ekran çıktısıyla karşılaşacaksınız.



Şekil 1.65: main.cpp kaynak dosyasının Nano ile görüntüsü bu şekilde olacak. Görüntünün alt bandında Nano'nun menüsünü görüyorsunuz.

Main.cpp çok büyük bir dosyadır. Bu yüzden değişiklikleri yaparken Nano'nun arama özelliğini kullanmamız gerekecek. **Shift**+**W** tuş kombinasyonuyla bu özelliği aktif edebilirsiniz. Arama kutusu altta yer alan **menü** bandında belirecektir. Arayacağımız anahtar sözcük "**pchMessageStart**"tır. Arama kutucuğuna bu anahtar sözcüğü yazıp **Enter** tuşuna bastığınızda **main.cpp** dosyası içerisinde ilgili bölüme gelmiş olacağız.

```
 GNU nano 2.2.6                Dosya: main.cpp                      Değiştirildi

    nBestInvalidWork = 0;
    hashBestChain = 0;
    pindexBest = NULL;
}

bool LoadBlockIndex()
{
    if (fTestNet)
    {
        pchMessageStart[0] = 0xfc;
        pchMessageStart[1] = 0xc1;
        pchMessageStart[2] = 0xb7;
        pchMessageStart[3] = 0xdc;
        hashGenesisBlock = uint256("0xf5ae71e26c74beacc88382716aced69cddf3dffff2$
    }

    //
    // Load block index from databases
    //
    if (!fReindex && !LoadBlockIndexDB())
        return false;

    return true;
}


Ara [pchMessageStart]:
^G Yardım Al  ^Y İlk Satır  ^T Satıra Git ^W Beg of ParM-J İki Yana  M-B Geriye Doğru
^C İptal      ^V Son Satır  ^R Değiştir   ^O End of ParM-C Harf DuyaM-R Düzenli İfad
```

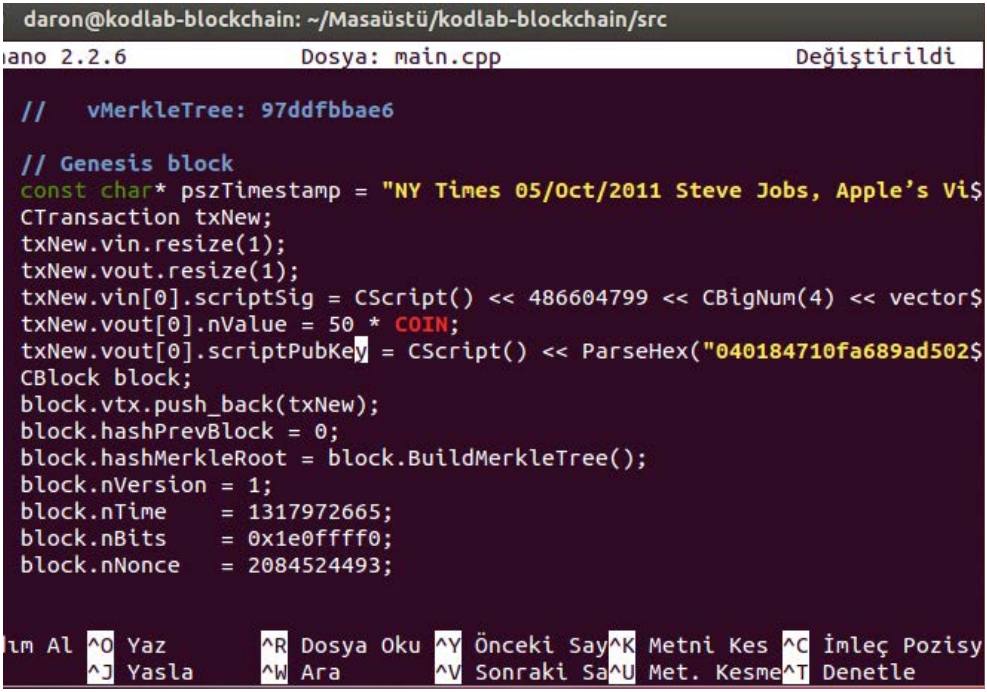
Şekil 1.66: Nano'nun arama özelliği ile "pchMessageStart" başlığına gidiyoruz. Genesis blok burada yer alıyor.

Genesis bloğumuz bu kod bloğu altında yer alıyor. Onu silmemiz gerekiyor. Çünkü yerine kendi ağımıza ait **Genesis** bloğun **hash** değerini yazacağız. Genesis bloğun başlangıç iki karakterini bırakarak geri kalan hash değerini siliyoruz. Ardından `pchMessageStart[0]` başlığının değeri olan `0xfc` üzerinde bir değişiklik yapıyoruz. Ne yaptığınız çok önemli değil. Burada amacımız blok oluştururken Litecoin karakteristiğinden uzaklaşmak. Ben örneğimde bu değeri `0xd2` yapıyorum mesela. Baştaki `0x` karakterleri sabit kaldığı sürece (*Genesis bloğumuzla aynı olmalı*) ne yaptığınız çok fark etmiyor. Ardından gelen `pchMessageStart[1]` başlığının değerini de değiştiriyorum ve `0xc3` yapıyorum. Geri kalan mesaj başlıklarının değerleri aynı kalabilir. Artık Genesis blok oluştururken Litecoin ağından tamamen kopmuş olacağız.



Şekil 1.67: Yukarıda gördüğünüz üzere blok başlık karakteristiğini ve Genesis blok hash değerini değiştirmiş olduk.

Şimdi ağımızın zaman damgasını oluştururken kullanacağımız değerleri güncellememiz gerekiyor. Nano'nun arama kutucuğuna geri dönüyoruz. Anahtar sözcüğümüzü "**scriptPubKey**" olarak giriyoruz. Aradığımız satır "**`txNew.vout[0].scriptPubKey`**" ile başlıyor.



Şekil 1.68: Nano ile arama yaparak "txNew.vout[0].scriptPubKey" satırına gidiyoruz.

Burada **`ParseHex`** değerini değiştireceğiz. İlk beş haneyi silip yerine **`123`** yazmanız yeterli olacaktır. Bunu yaptıktan sonra üst satırlarda yer alan "**`pszTimestamp`**" satırında yer alan açıklamayı silerek değiştireceğiz. Litecoin, bu satırda Steve Jobs'un ölüm tarihine gönderme yapıyor. Biz kendimize göre bir şeyler yazarak devam edelim.

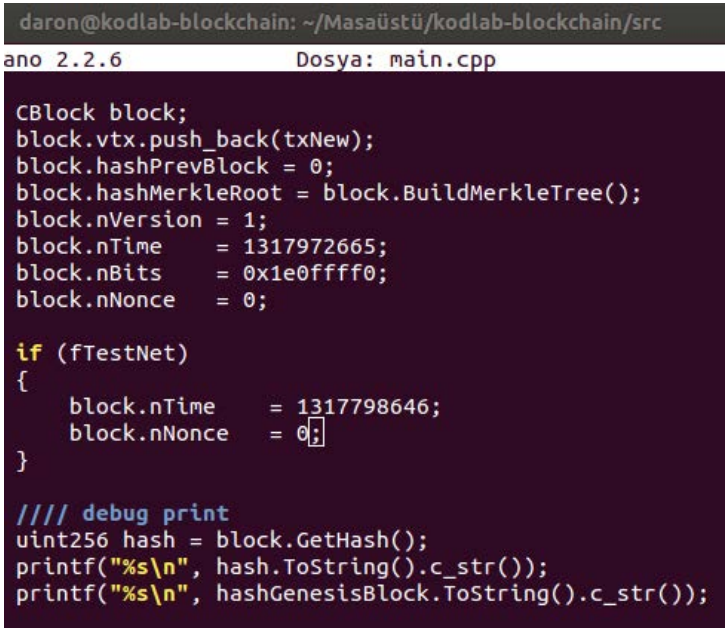
```
daron@kodlab-blockchain: ~/Masaüstü/kodlab-blockchain/src
ano 2.2.6              Dosya: main.cpp                    Değiştirildi

//   vMerkleTree: 97ddfbbae6

// Genesis block
const char* pszTimestamp = "Merhaba blockchain! Ben Kodlab coin!";
CTransaction txNew;
txNew.vin.resize(1);
txNew.vout.resize(1);
txNew.vin[0].scriptSig = CScript() << 486604799 << CBigNum(4) << vector$
txNew.vout[0].nValue = 50 * COIN;
txNew.vout[0].scriptPubKey = CScript() << ParseHex("1234710fa689ad50236$
CBlock block;
block.vtx.push_back(txNew);
block.hashPrevBlock = 0;
block.hashMerkleRoot = block.BuildMerkleTree();
block.nVersion = 1;
block.nTime    = 1317972665;
block.nBits    = 0x1e0ffff0;
block.nNonce   = 2084524493;
```

Şekil 1.69: "ParseHex" değerini değiştirdikten sonra "pszTimestamp" satırındaki açıklamayı da değiştirmeniz gerekiyor.

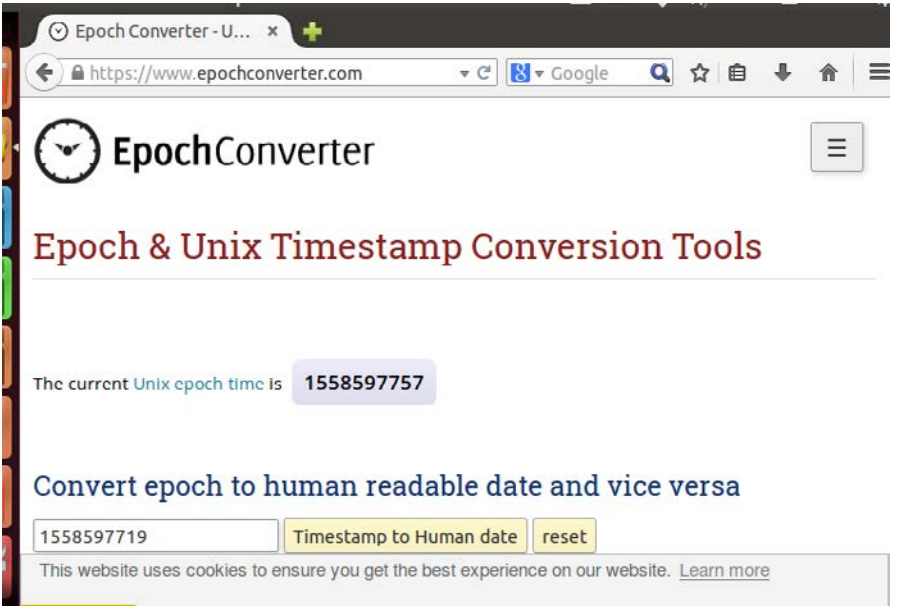
Buradaki değişiklikler bittiğinde sıra `block.nNonce` değerini değiştirmeye geliyor. `nNonce` değeri bizim için çok önemli; çünkü Blockchain ağının alametifarikası olan **zaman damgası**nı bu değere göre oluşturuyoruz. İlerleyen adımlarda oluşturacağımız `MerkleRoot` ve `Genesis Blok` değerleri için bu aşama çok önemli. Şimdi `block.nNonce` değerini silerek `0` değerini girelim.



Şekil 1.70: "block.nNonce" değerini 0 ile değiştiriyoruz.

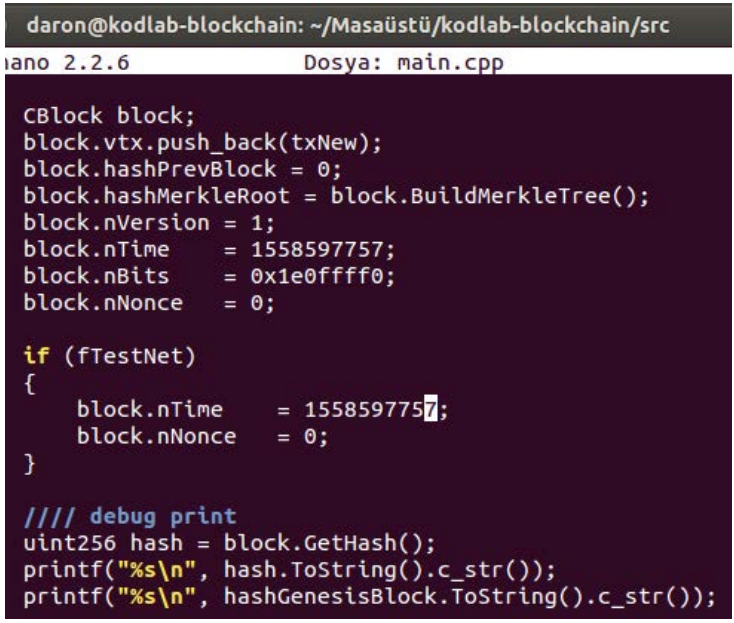
Hemen altında yer alan "`TestNet`" kısmındaki "`nNonce`" değerini de değiştirmeyi unutmayın.

Şimdi ağımızı oluşturduğumuz tarihi belirleyerek "`block.nTime`" değerini de değiştirmemiz gerekiyor. Yalnız bu aşamada alışık olduğumuz tarih yapısını kullanamıyoruz. Unix genellikle "**epoch**" zaman düzenini benimsemiştir. O yüzden biz de güncel tarihi (*gg/aa/yy/s/dk/sn/sl şeklinde*) epoch zaman formatına çevirmeliyiz. Bunun için arama motorlarında "**epoch converter**" araması yaparsanız kullanışlı bir çözüm bulabilirsiniz.



Şekil 1.71: Google karşımıza *epochconverter.com* servisini çıkarttı. Current time yazan kısımdan değeri kopyalıyoruz.

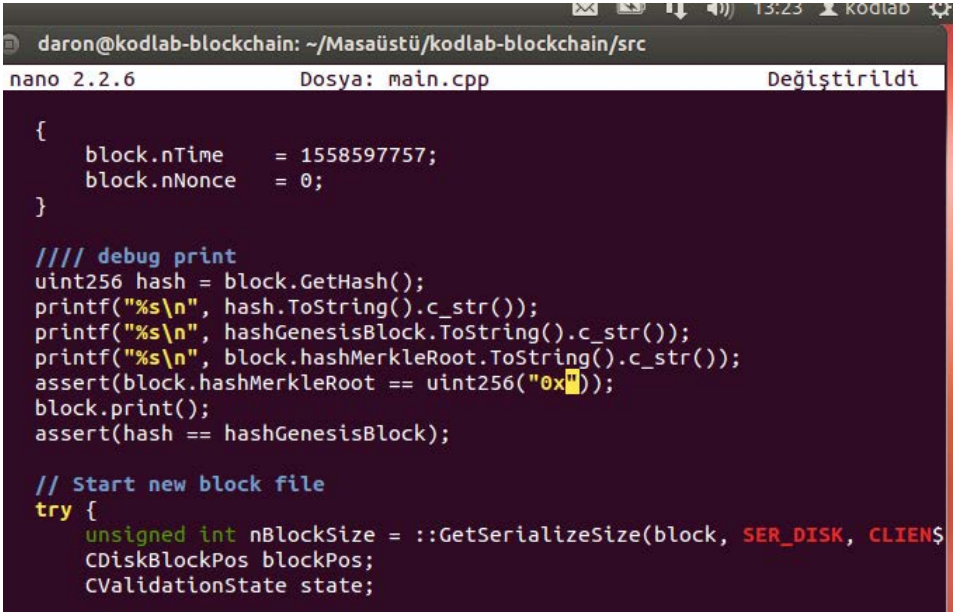
Google üzerinde küçük bir aramanın ardından *epochconverter.com* adresindeki servisi bulduk. Gerçekten pratik. Aradığımız değeri hemen kutucuk içerisinde **current Unix epoch time** satırında görebilirsiniz. Kopyalayıp **main.cpp** içerisindeki ilgili yerlere yapıştırma zamanı geldi.



Şekil 1.72: Kopyaladığımız epoch değerini "block.nTime" yazan yerlere yapıştırıyoruz.

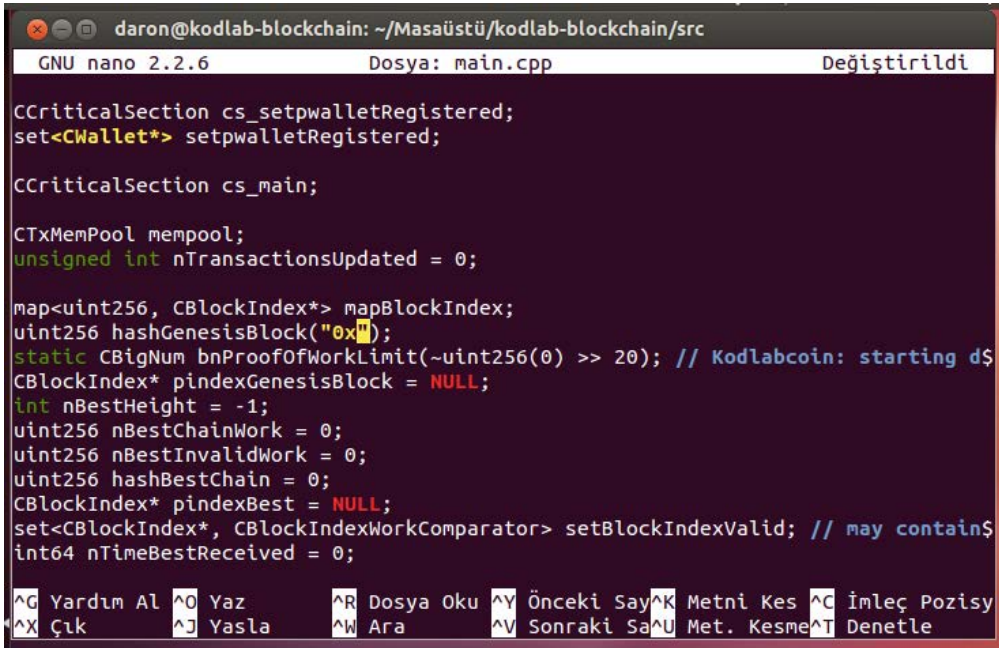
Yeri kaybettiyseniz hemen Nano'nun arama kutucuğuna "**block.nTime**" yazıp karşısında yer alan değerleri değiştirebilirsiniz.

Zaman tarafı Blockchain ağımızın mutabakat yapısı için çok önemli bir nokta. Sanırım şimdi, karşılığını silmediğimiz **MerkleRoot** ve **Genesis blok** dizgisi kaldı. Şu an bulunduğumuz `block.nTime` başlığı altında birkaç satır aşağı inerseniz (`//// debug print` *başlığı altına doğru*) `MerkleRoot` değerini belirleyen `Uint256` satırını göreceksiniz.



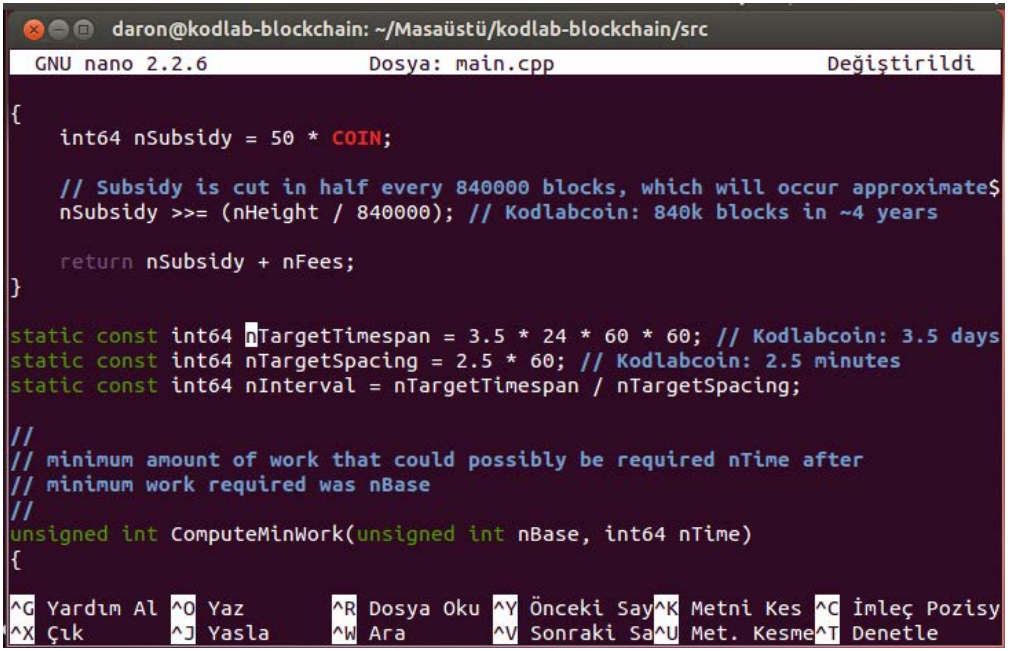
Şekil 1.73: MerkleRoot hash değerini değiştirmeyi unutmayın.

İlk iki hanesi olan `0x` kalmak şartıyla dizinin tamamını silebilirsiniz. Şimdi Nano'un arama kutucuğuna "`Genesis block`" anahtar sözcüklerini yazıp Genesis block değerimizi değiştirelim.



Şekil 1.74: Genesis Block değerini ilk iki hanesi kalacak şekilde silebilirsiniz.

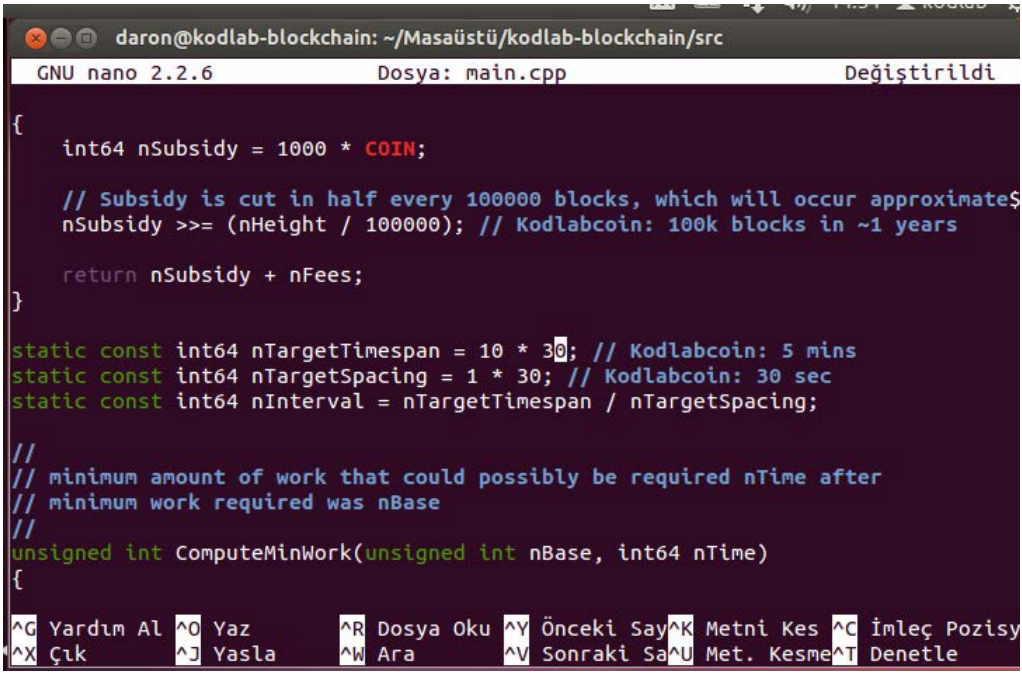
Evet, ağımızın karakteristik yapılarının temeli olan **zaman damgası** ve **başlangıç bloğu**nu yaratmak için gerekli olan **hash** değerlerini değiştirmiş olduk. Şimdi, sırada biraz daha özelleştirme yapmak var. Bölümün başında *ağın iş modeline uygunluğunun tespitinden* bahsetmiştim. Yapacağınız iş modeline göre birtakım zorunlu değişiklikler yapmalıyız demiştim ve kendi modelimden örnek vermiştim. Hızlı bir ağ olmasını istiyorum demiştim. Ağın hızını blok oluşturma süresi belirler. Bu süre mutabakat sürecinde kaç onay istediğinize bağlı olarak değişkenlik gösterebilir. Mesela, Bitcoin her 10 dakikada bir blok oluşturur. Yani gönderiminiz 10 ile 30 dakika içerisinde değişen sürelerde karşıdaki kişiye ulaşır. Burada minimum 3 onay istendiği düşünülürse işlem süresi 30 dakika ile 90 dakika arasında değişkenlik gösterebilir. Demek ki öncelikle blok süresini kısaltmam ve onay sayımı en fazla 3, en az 2 olacak şekilde ayarlamam gerekecek. Böylece ağımın hızı oldukça artacak. Hadi başlayalım. Nano'nun arama kutucuğuna "**nTarget**" yazarak **main.cpp** dosyası içerisinde ilgili bölüme geçiş yapıyoruz. Burada **Şekil 1.75**'teki gibi bir ekran karşılayacak sizi.



Şekil 1.75: Bu ekran bize baz aldığımız Litecoin'in blok başı ödül miktarından "halving" hesabına kadar hemen her özelliğini gösteriyor. Her değeri kendi ihtiyaçlarımıza göre değiştirmeliyiz.

Ekran görüntüsünün üst tarafında "`nSubsidy`" değerini görüyoruz. Bu değer bizim blok başı ödül miktarımızı belirliyor. Bir başka deyişle paramızın arz miktarını bu değerle ayarlıyoruz. Ne kadarlık toplam arza sahip olacağınıza göre bu değerleri düzenlemelisiniz. Mesela, benim örneğimde toplam arzım 1 milyar adet olacak. O yüzden arzı 50 adet kripto para ile başlatamam. Bu değeri 1000 adete çıkartıyorum. Hemen alt satırda yarılanma sürecini belirleyen fonksiyonumuz bulunuyor. **Yarılanma** dediğimiz yapı, her belli sayıda blok çözüldüğünde ödül miktarını yarıya indirerek arzı kısan özel bir yapıdır. 840 bin adet blok çözüldüğünde Litecoin, blok başı ödül miktarını yarılar. Bu değer benim ağım için çok büyük bir değer. Yarılanma için 840 bin bloğun çözülmesini bekleyecek olursam toplam arzımın neredeyse yüzde 80'i dolaşıma girmiş olacak. O yüzden bu değeri 100 bin bloğa çekiyorum. Her 100 bin blok çözüldüğünde blok başı 1000 adet olan ödül miktarım yarıya düşecek.

`nTarget` değerlerini de ağ hızımı artıracak şekilde düzenliyorum. Litecoin her 2,5 dakikada bir blok üretiyor. Ben bu süreci 30 saniyeye indiriyorum. Böylece zaman damga döngüm de 3,5 günden 5 dakikaya inmiş oluyor. Bu tarz bir yapı ağdaki işlem sayısını doyurucu bir süreçte onaylayarak tamamlamak için biçilmiş kaftandır. Eğer ağın hızını artırmak için blok süresini 30 saniyenin altına çekecek olursak birçok blok tam kapasiteyle işleme alınamayacağından ağın hızlı olması bir işe yaramayacaktır. Şimdi siz de kendi ağınızın kurgusunu tanımladıysanız değişiklikleri uygulamaya başlayabiliriz.



Şekil 1.76: Gerekli düzenlemeleri tamamladığınızda artık karakteristiği olan bir ağ yapısı kurgulamış oldunuz.

Son olarak **main.cpp** dosyasına ekleyeceğimiz bir kod bloğu var.

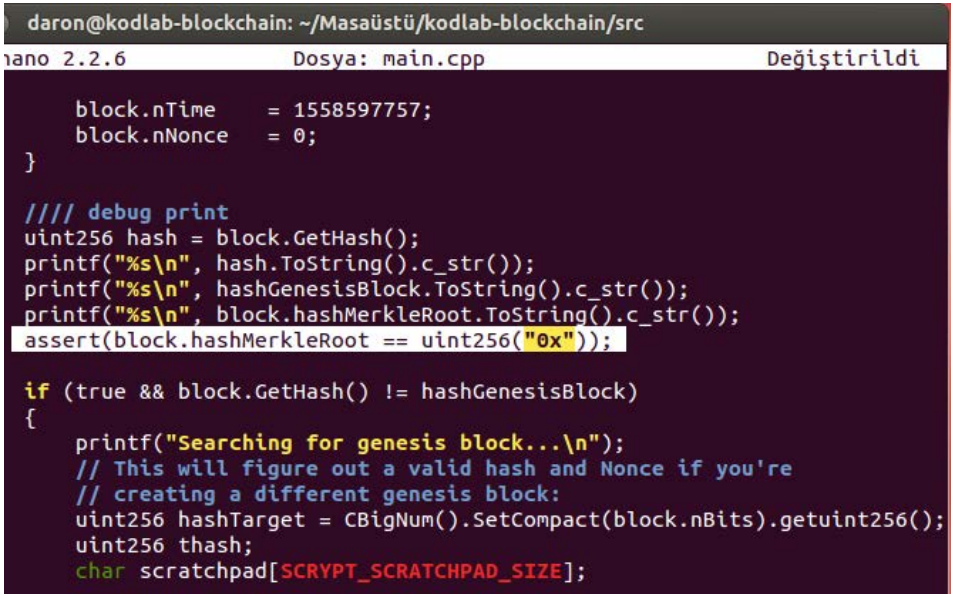
Bu bloğu **hash** değerini sildiğimiz `MerkleRoot` satırının hemen altına ekliyoruz.

Kod bloğumuz burada:

```
if (true && block.GetHash() != hashGenesisBlock)
        {
            printf("Searching for genesis block...\n");
           // This will figure out a valid hash and Nonce if you're
     // creating a different genesis block:
            uint256 hashTarget = CBigNum().SetCompact(block.nBits).
getuint256();
            uint256 thash;
            char scratchpad[SCRYPT_SCRATCHPAD_SIZE];

            loop
            {
                scrypt_1024_1_1_256_sp(BEGIN(block.nVersion),
BEGIN(thash), scratchpad);
                if (thash <= hashTarget)
                    break;
                if ((block.nNonce & 0xFFF) == 0)
                {
                    printf("nonce %08X: hash = %s (target = %s)\n",
block.nNonce, thash.ToString().c_str(), hashTarget.ToString().c_
str());
                }
                ++block.nNonce;
                if (block.nNonce == 0)
                {
                    printf("NONCE WRAPPED, incrementing time\n");
                    ++block.nTime;
                }
            }
            printf("block.nTime = %u \n", block.nTime);
            printf("block.nNonce = %u \n", block.nNonce);
            printf("block.GetHash = %s\n", block.GetHash().
ToString().c_str());

            }
```

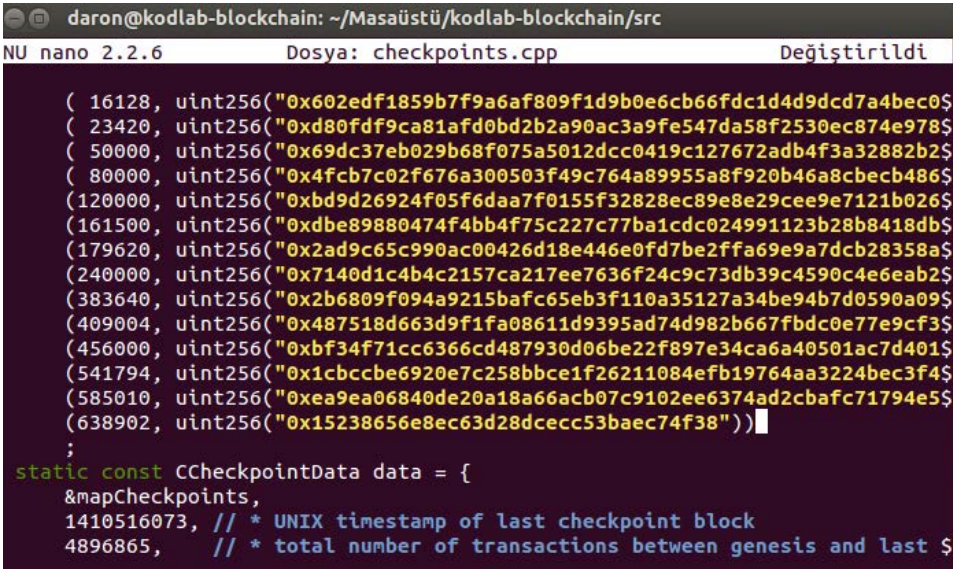


Şekil 1.77: Yukarıdaki kod bloğunu Şekil 1.77 üzerinde taradığım satırın hemen altına yapıştırmanız gerekiyor.

Bu kod bloğunu farklı bir **GenesisBlock** oluşturarak ayrı bir ağ oluşturmak istediğimiz için kullanıyoruz. Yani bir nevi **hard fork** yapmamızda yardımcı oluyor.

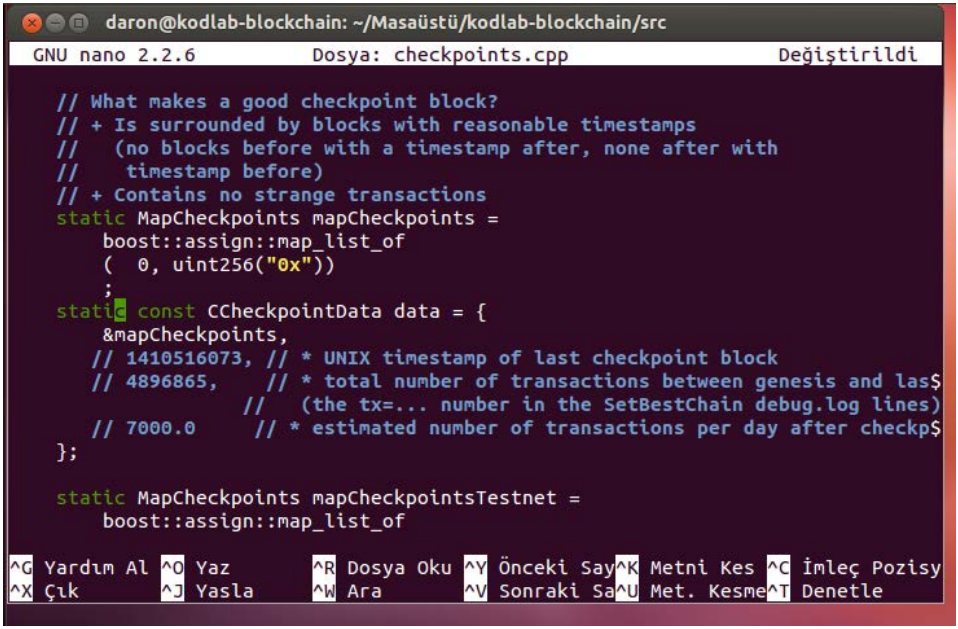
Main.cpp dosyası üzerindeki değişiklikleri tamamlandıktan sonra sıra **checkpoints.cpp** dosyasına geliyor. Bu dosya, **main.cpp** dosyası içerisindeki kodlarımızda bulunan ve sildiğimiz GenesisBlock'a ait olan **checkpoint hash** değerlerini barındırıyor. Bu hash değerlerini sıfırlamamız gerek. Hemen "**kodlab-blockchain/src**" dizini altında şu komutu çalıştırıyoruz:

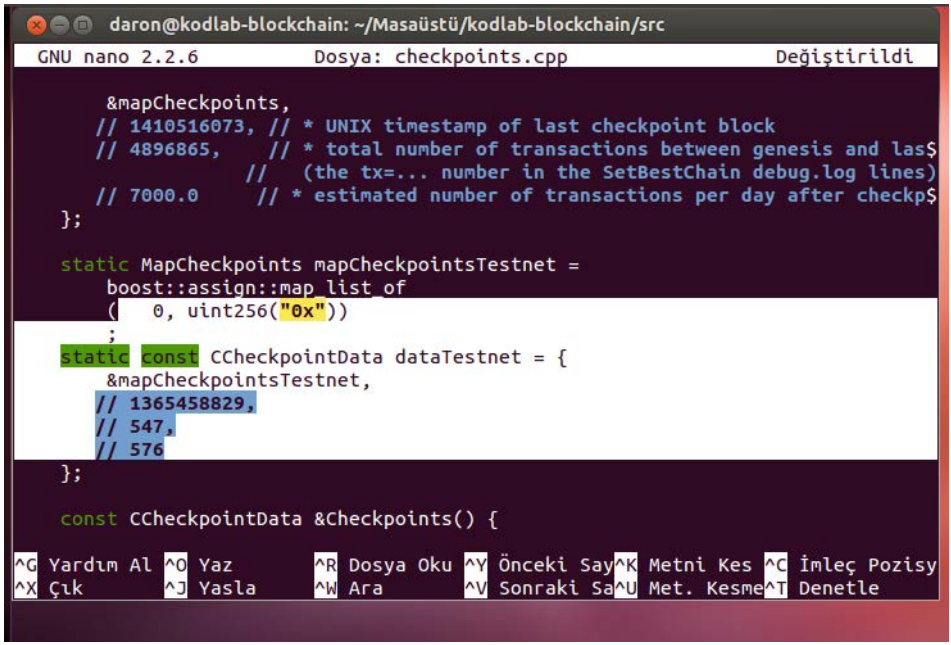
```
sudo nano checkpoints.cpp
```



Şekil 1.78: checkpoints.cpp dosyası içerisinde birçok yer değiştirilecek.

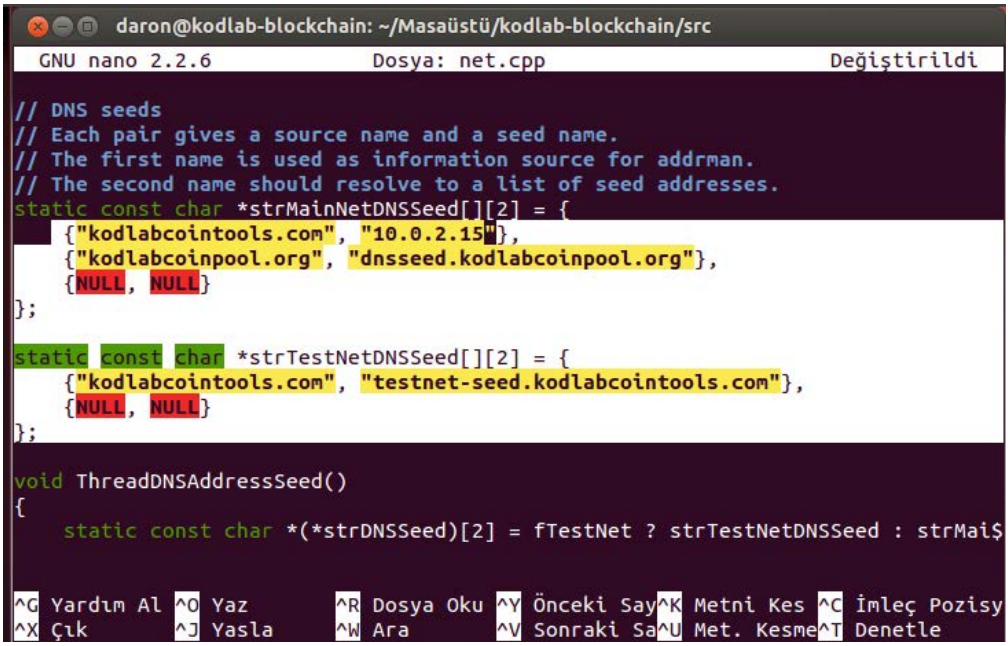
Checkpoints.cpp dosyası açıldığında **Şekil 1.78**'de gördüğünüz ekranla karşılaşacaksınız. Bu ekranda birçok `uint256 hash` bloğu olduğunu göreceksiniz. Bu değerler bizim ağımızla alakalı değil. Bu değerleri kendi ağımız için tekrardan oluşturacağız. O nedenle şimdi Litecoin'e ait kontrol noktalarını siliyoruz. Sadece tek bir satır bırakacağız. En üstteki satıra geldiğimizde tırnak işaretleri arasında kalan hash değerini baştaki iki karakter (`0x`) kalana kadar sileceğiz. Bu işlem bittiğindeyse hemen alt satırda yer alan "`static const CCheckpointData`" başlığı altındaki değerleri çift **/** işaretiyle yorum satırı hâline getiriyoruz. (*Çift / işareti C ve C++ dilinde yorum satırlarını belirtmek için kullanılır ve derleyici tarafından görmezden gelinir.*)





Şekil 1.79 ve 1.80: İşlemler bittiğinde checkpoint.cpp dosyamız bu şekilde görünecek.

Şimdi gelelim en önemli bölüme; yani zincirimizin baş halkasının IP adreslerini belirlemeye. Bu işi **net.cpp** dosyası içerisinde yapıyoruz. Mevcutta bulunan tüm adres ve IP numaralarını silip yerine varsa projenize ait gerçek bir domain, IP adresi veya yerel ağda oluşturduğunuz yerel IP numarasını girmeniz gerekiyor.



Şekil 1.81: Gerçek bir proje yapıyorsanız DigitalOcean, Godaddy ya da evinizde yayına alacağınız bir sabit IP değerini, bu tarayarak belirlediğim alana girmeniz gerekiyor. Sadece eğleniyorsanız yerel IP adresinizi girin.

Şimdi, kaynak kodlar üzerinde yapmamız gereken tüm özelleştirmeleri tamamladık. Artık önceki adımlarda **main.cpp** içerisinde sıfırladığımız **MerkleRoot** ve **GenesisBlock** değerlerini oluşturup ilgili yerlere eklemek kalıyor. Bu işlem tamamlandığında ağınız kullanıma hazır olacak. Yalnız bu işlemleri yapabilmek için kaynak kodlarımızı derlememiz gerekiyor. Derleme işlemi için sistemimize belli başlı kriptografi kütüphanelerini yüklememiz gerek. Şimdi gelin bu işlemi tamamlayalım.

Terminal üzerinden **main.cpp** dosyasını kaydederek kapatıp aşağıdaki komutları tek tek uygulayın.

```
sudo apt-get install build-essential libtool autotools-dev autoconf
pkg-config libssl-dev

sudo apt-get install ntp git build-essential libssl-dev libdb-dev
libdb++-dev libboost-all-dev libqrencode-dev libcurl4-openssl-dev
curl libzip-dev libzip2

sudo apt-get install libboost-all-dev

sudo apt-get install software-properties-common python-software-
properties

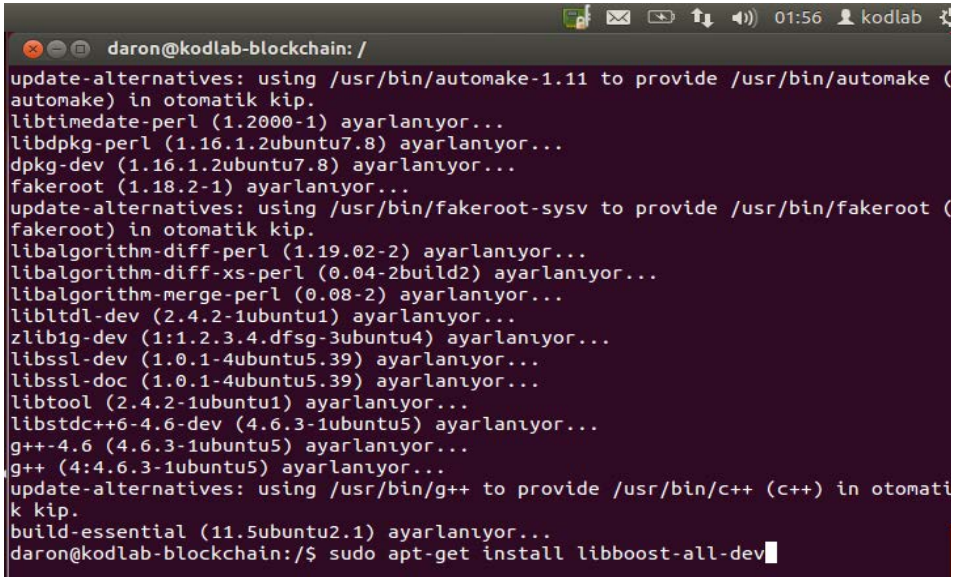
sudo add-apt-repository ppa:bitcoin/bitcoin

sudo apt-get update

sudo apt-get install libminiupnpc-dev

sudo apt-get install libdb4.8-dev

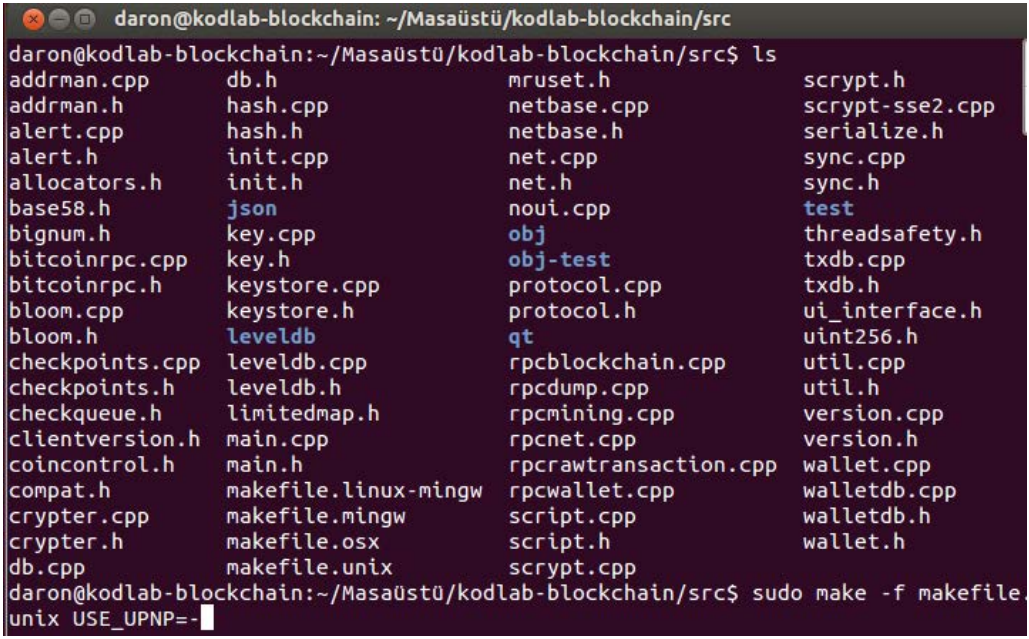
sudo apt-get install libdb4.8++-dev
```



Şekil 1.82: Kurulum işlemleri bittiğinde sistemimiz yeni Blockchain yapımızı derlememiz için hazır olacak.

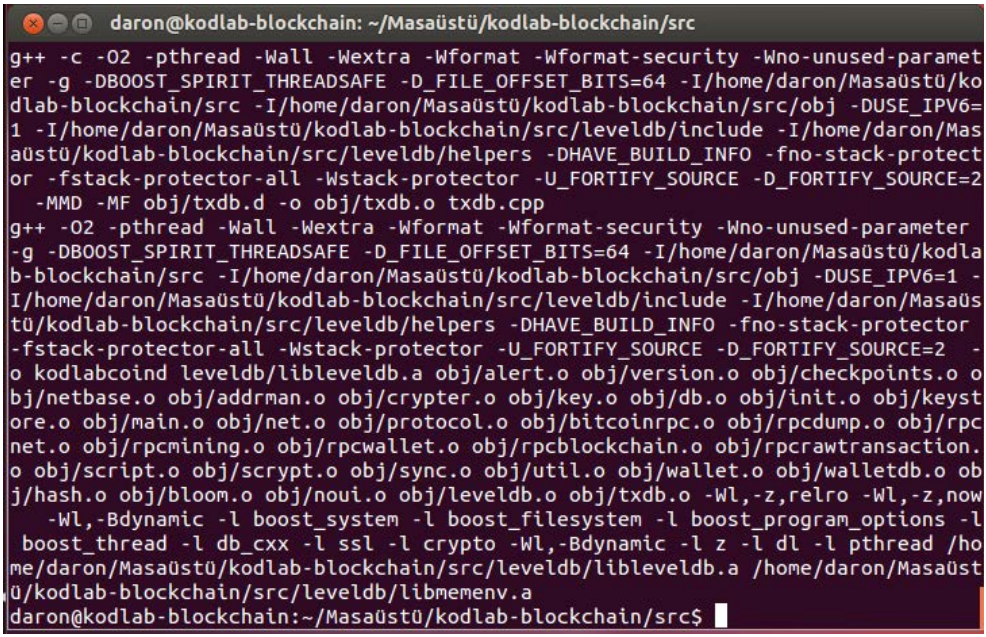
Tüm dosyaları kurduktan sonra artık derleme işlemine geçebiliriz. Terminalde **kodlab-blockchain** dizini altındaki **src** klasörü içerisine girerek aşağıdaki komutu veriyoruz:

```
sudo make -f makefile.unix USE_UPNP=-
```



Şekil 1.83: Derleme işlemine başlıyoruz.

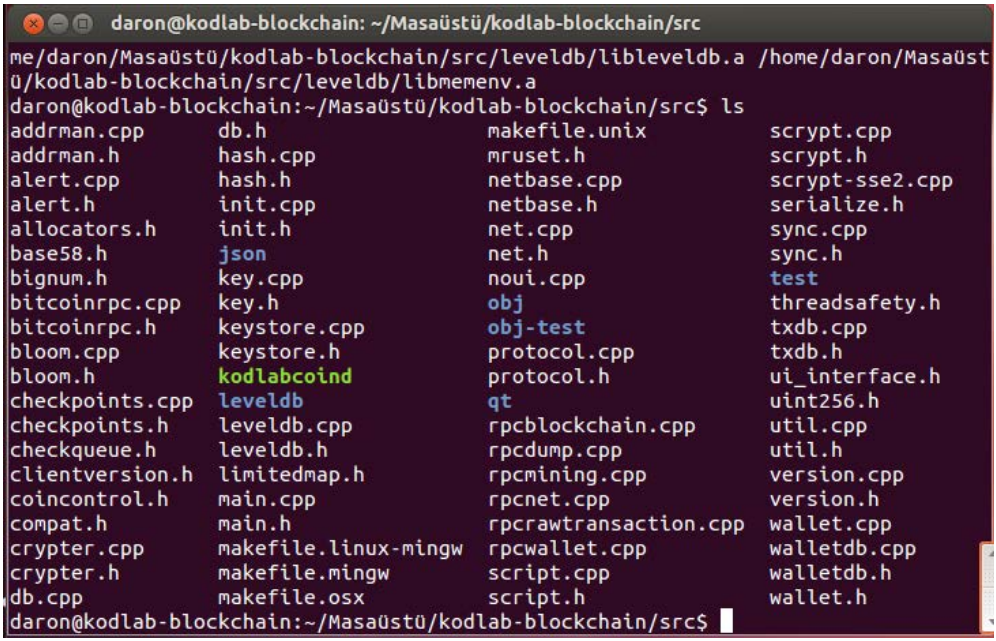
Bu komut dizisi tüm kaynak dosyalarımızı derleyerek ihtiyaç duyduğumuz **kodlabcoind binary** dosyasını oluşturacak. Eğer sisteminizde bir eksik yoksa ya da kaynak dosyaları düzenlerken bir hata yapmadıysanız aşağıdaki gibi bir ekranla karşılaşacaksınız.



Şekil 1.84: Derleme işlemimiz bitti. Artık sıra Blockchain uygulamamızı kontrol etmeye geldi.

Eğer bir hata mesajı aldıysanız, üç noktaya dikkat etmeniz gerekiyor. İlki, derleme yapabilmek için ihtiyaç duyduğumuz dosyaları kurarken atladığınız bir komut olmadığından emin olmak olmalı. Eğer önceden kurduğunuz bir dosyayı kurmaya çalışırsanız sadece uygulamaların kurulu olduğuna dair bir mesaj alırsınız. Dolayısıyla tekrar denemek, sisteme zarar vermeyecektir. Eğer bu aşamada kurmayı atladığınız bir dosya olmadığını fark ederseniz **main.cpp**, **checkpoints.cpp** ve **net.cpp** dosyalarında yaptığımız değişiklikleri tekrar kontrol etmeniz gerekecek. Eğer o adımlarda da bir eksik ya da hata yoksa donanımınızın sistem kaynakları uygulamayı kaldırmıyor olabilir.

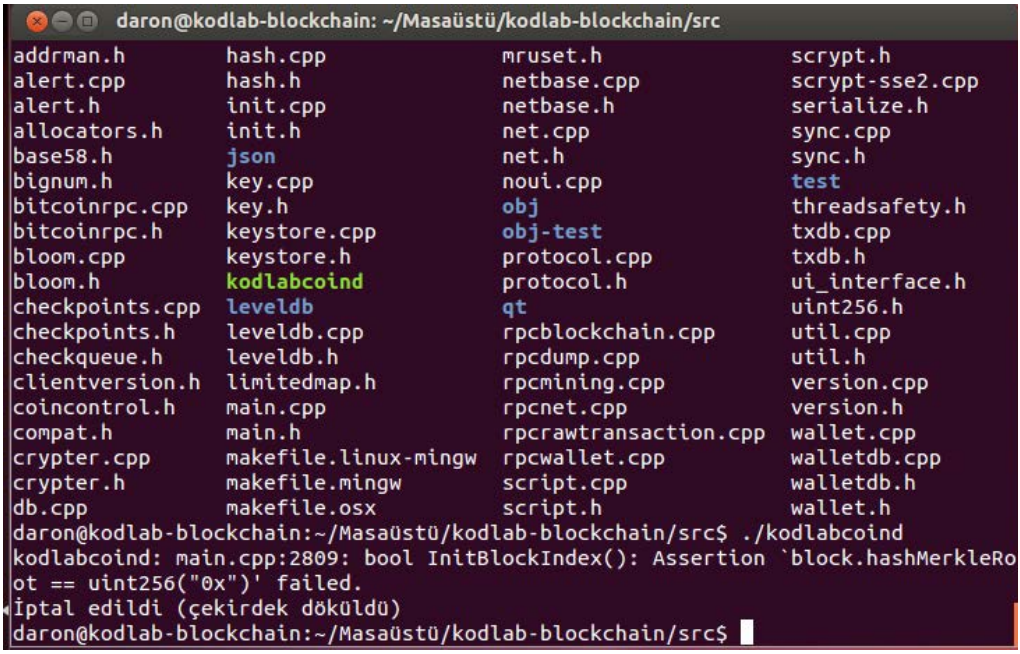
Hiç hata almadıysanız "`ls`" komutunu vererek derleme işleminin sonucunu görme zamanı geldi. Sizi aşağıdaki gibi bir ekran karşılayacak.



Şekil 1.85: Ekran görüntüsünde yeşil renkli olan çalıştırılabilir kodlabcoind dosyamız ağımızın çekirdek uygulamasıdır.

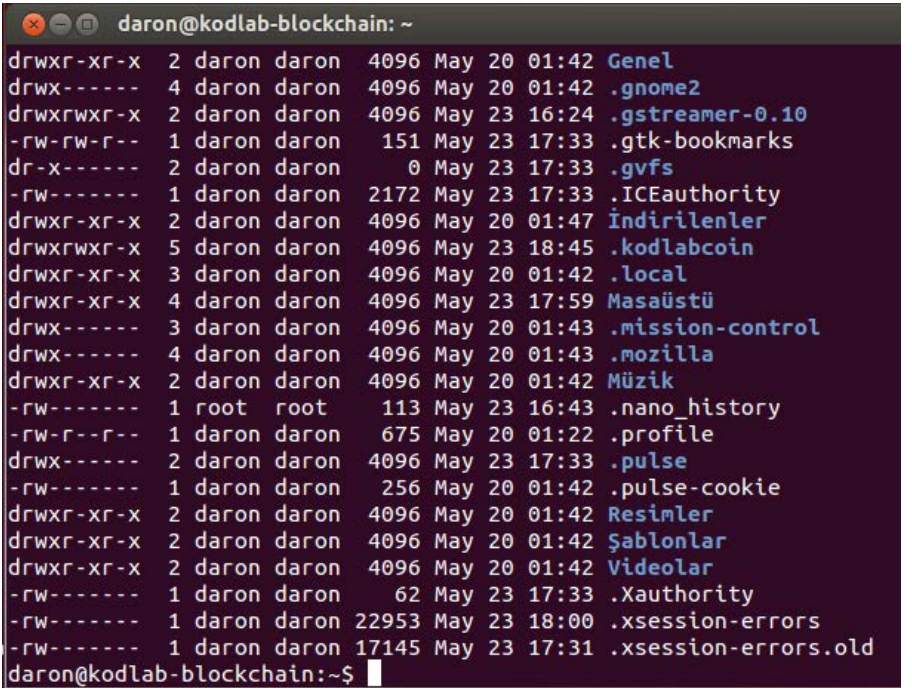
Şimdi terminal üzerinde "`./kodlabcoind`" komutunu çalıştırıyoruz. Bu tarz çalıştırılabilir (*binary*) dosyaları çağırırken önüne . ve / işaretlerini koymamız gerekiyor. Komutu çalıştırdığınızda ekrana bir hata mesajı çıkacak. Bu durum gayet normal. Hatırlarsanız **main.cpp** dosyası içerisinden **MerkleRoot** ve **GenesisBlock** değerlerini silmiştik.

Çekirdeğimiz bu değerleri okuyamadığı için bu hatayı veriyor. Şimdi bu eksikleri giderme zamanı geldi.



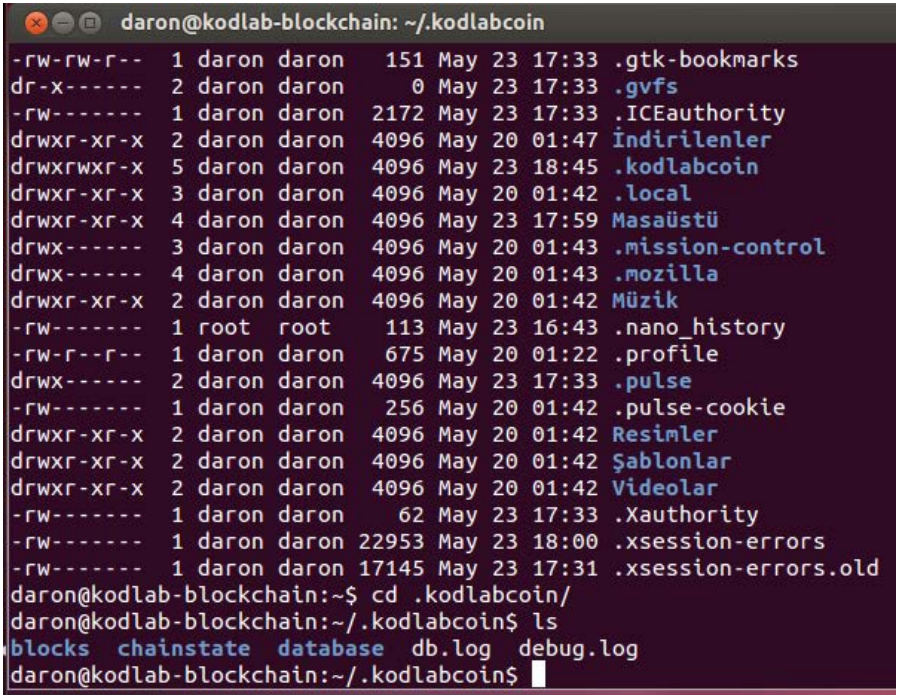
Şekil 1.86: Terminalde ./kodlabcoind uygulamasını çalıştırdığınızda hata mesajı alacaksınız.

Hata mesajımızı aldıktan sonra **blockchain** çekirdek uygulamamızın blokları ve ayarlarını tuttuğu gizli klasöre gitmemiz gerekiyor. Linux işletim sisteminde uygulamalara ait dizinler kullanıcıların ev dizini altında tutulur. Terminalde "`cd . .`" komutunu kullanarak **ev** dizinine kadar çıkıyoruz.



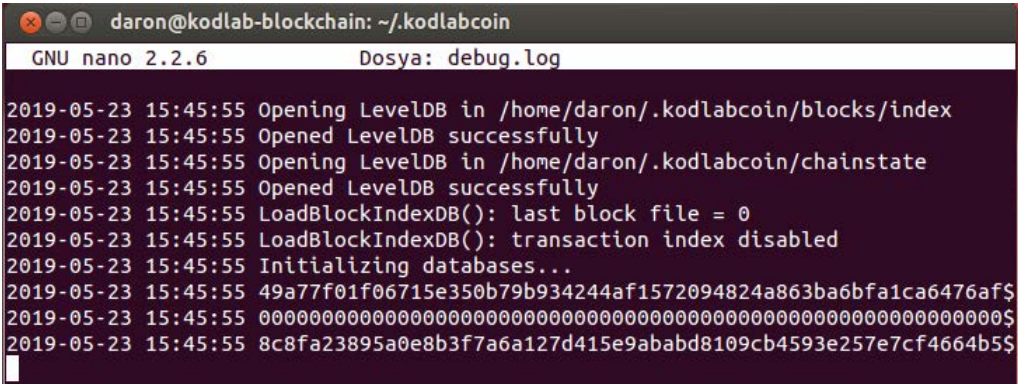
Şekil 1.87: Ev dizini altında "ls la" komutunu vererek gizli dosyalar dâhil tüm klasörleri listeliyoruz. ".kodlabcoin" klasörü aradığımız klasör.

Ev dizini altındayken "`ls la`" komutunu çalıştırıyoruz. "`ls`" komutuna eklediğimiz "`la`" parametresi gizli dosyaları da görmemizi sağlıyor. Eğer sadece `ls` ya da `ls -a` komutunu çalıştırsaydık gizli dosyaları göremezdik. Şimdi sizi, **Şekil 1.87**'de gördüğünüz ekran karşılamalı. Burada artık "`cd .kodlabcoin`" komutunu vererek klasöre giriş yapıyoruz. Bu klasörün içeriğini "`ls`" komutu ile listeliyoruz. Gördüğünüz dosyalar bizim oluşturduğumuz Blockchain yapısının temel bileşenleri oluyor. Bu dosyalar içerisinden **debug.log** adlı dosya, çekirdek uygulamamızın tüm çıktılarını saklar. O yüzden uygulamayı her çalıştırdığınızda bu dosyadaki satır ve veri sayısı artacaktır. Şimdi bu dosyayı "`sudo nano debug.log`" komutunu kullanarak **Nano** ile açıyoruz.



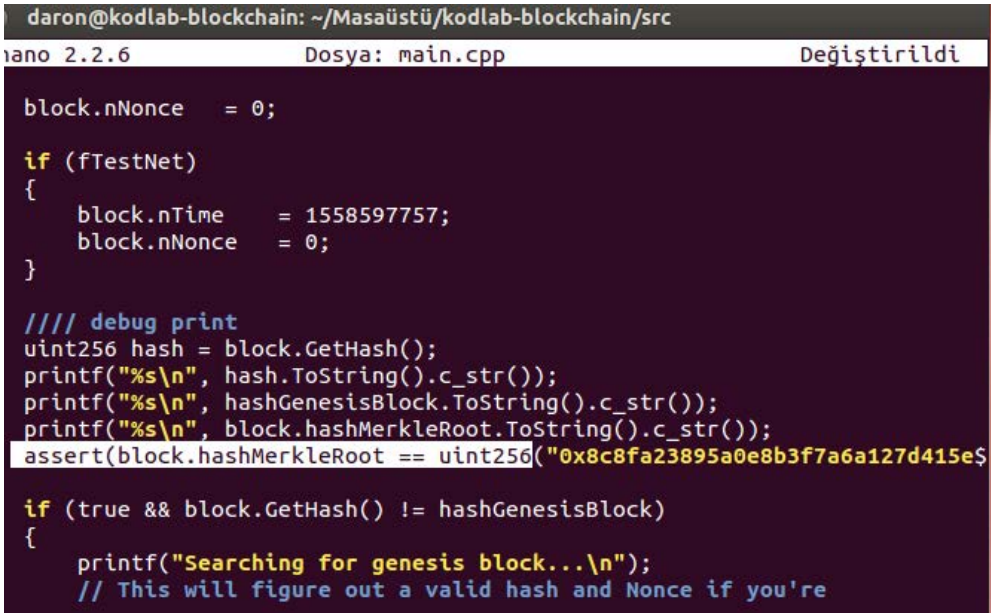
Şekil 1.88: Blockchain ağımızın tüm temel bileşenleri .kodlabcoin gizli klasörü altında yer alıyor.

Nano, **debug** dosyamızı açtığında en alt satıra ilerliyoruz. Orada hayati öneme sahip olan **MerkleRoot** değerimizi göreceksiniz. Bu değeri kopyalayarak bir metin dosyası içerisine alın. Birazdan onu kaynak dosyalarımız içerisinde olması gereken yere ekleyeceğiz.



Şekil 1.89: debug dosyasını açtığınızda en alt satırda 8c8f ile başlayan değer bizim MerkleRoot değerimiz. Kopyalayıp bir metin dosyasına yapıştırıyoruz.

Debug dosyasını kapatıp tekrardan **kodlab-blockchain/src** dizinine gidiyoruz. Burada **main.cpp** dosyamız içerisine **MerkleRoot** değerimizi yerleştireceğiz. Terminalde "`sudo nano main.cpp`" komutunu çağırıyoruz.



Şekil 1.90: MerkleRoot değerini gireceğimiz satırı bulmak için Nano'nun arama kutusuna "block.hashMerkleRoot" sözcüklerini yazmanız yeterli.

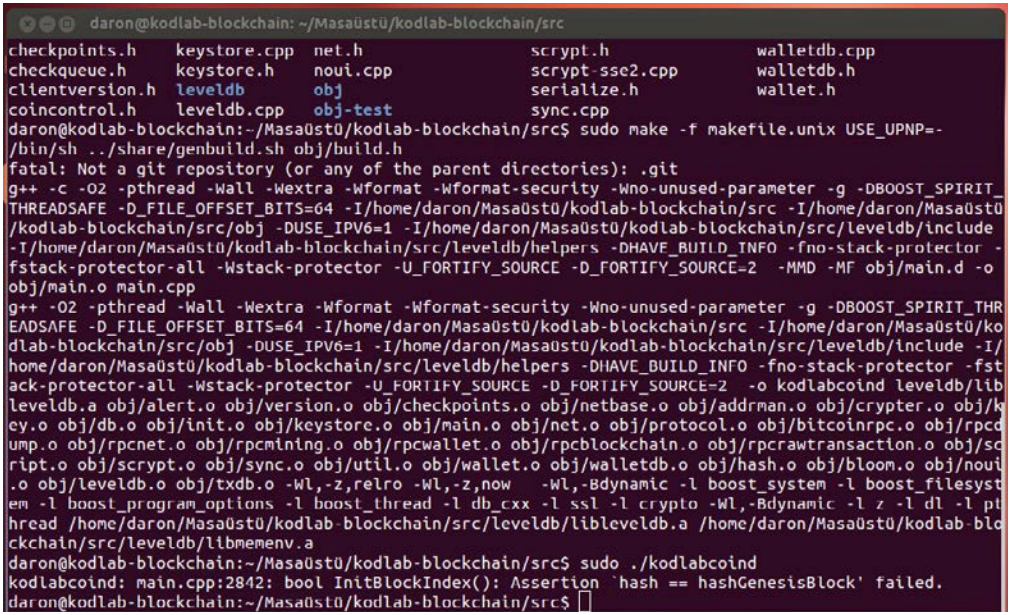
Main.cpp dosyasını açtığınızda Nano'un arama özelliğini açıp (**Shift+W**) "`block.hashMerkleRoot`" anahtar sözcüğümüzü yazıyoruz. Nano bizi **MerkleRoot** değerimizin bulunduğu satıra götürecek. Bu satırda hatırlarsanız Litecoin'e ait olan değeri başındaki iki karakter (`0x`) kalacak şekilde silmiştik. Şimdi buraya **debug.log** dosyasından aldığımız **MerkleRoot hash** değerini yapıştıracağız. Baştaki `0x` karakterlerinin hemen arkasına yapıştırıyoruz. Ben kendi ağımın oluşturduğu **Merkle** değerini `0x` karakterlerinden sonrasına yapıştırdım. Siz de bu işlemi kendi **Merkle** değerinizle yapıp **main.cpp** dosyasını kaydedin. Şimdi artık **MerkleRoot** değerimiz de olduğuna göre **src** klasörü içerisindeki "**kodlabcoind**" çekirdek uygulamamızı silip derleme işlemini yineleyebiliriz.

Derleme işlemi için aşağıdaki komutu kullanıyoruz.

```
sudo make -f makefile.unix USE_UPNP=-
```

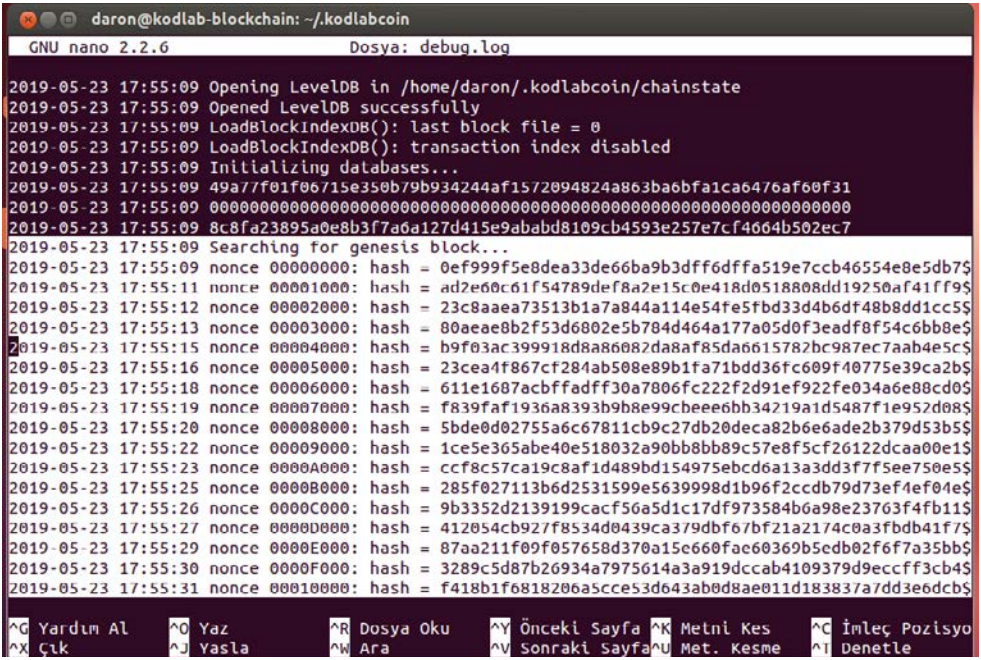
Bu işlem hatasız tamamlanacaktır. Artık elimizde kendi tasarladığımız Blockchain ağımızın çekirdek uygulaması var.

Fakat hâlâ **Nonce** değerimiz ve **Genesis Block hash** değerimiz yok. Bu iki değeri oluşturabilmek için yeni derlediğimiz çekirdeğimizi çalıştırmamız gerekiyor. Bu işlemi başlatıp çekirdek uygulamamız **çakılma** (*crash*) yaşayana kadar bekleyeceğiz. İşlemci gücünüze göre birkaç dakika beklemeniz gerekebilir. Çakılma hatasız gerçekleşince **debug.log** dosyamıza geri döneceğiz.



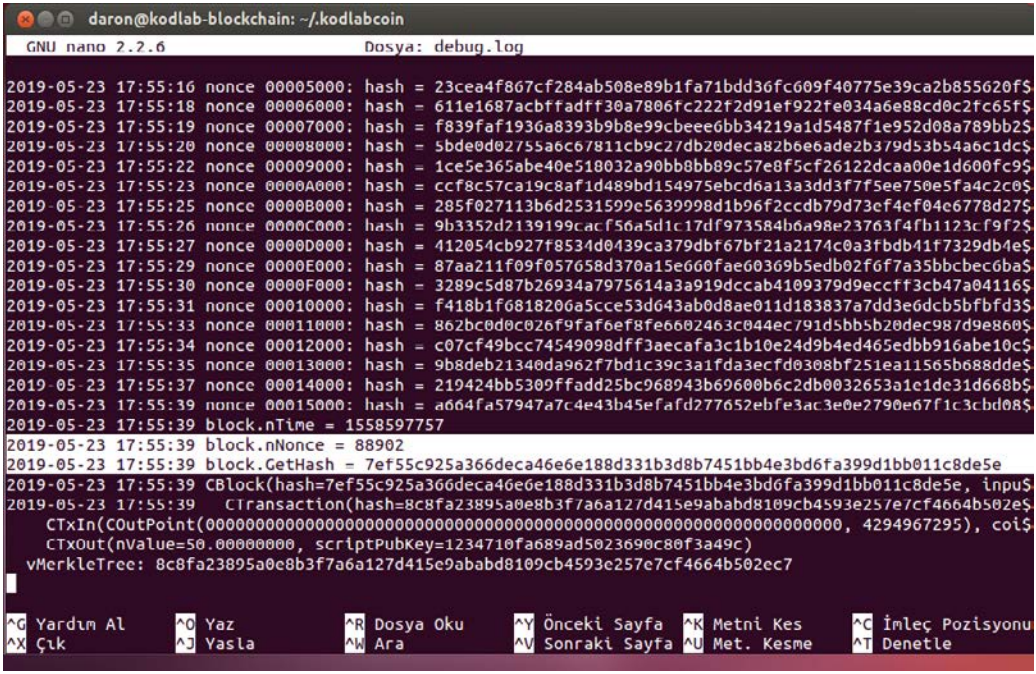
Şekil 1.91: MerkleRoot değerini eklediğimiz main.cpp dosyamızla çekirdek uygulamamızı tekrar derleyip çalıştırıyoruz. Bu defa uygulama çakıldığında nonce ve GenesisBlock değerlerini üretmiş olacağız.

Şekil 1.91'deki gibi bir çakılma hatası aldığınızda artık `nonce` ve `GenesisBlock hash` değerlerimiz üretilmiş demektir. Genesis dediğimiz, ağımızda olmasını istediğimiz şekilde üretilmiş **ilk blok** demektir. Bu blok, aynı zamanda zincirimizin ilk halkası ve mutabakat yapımızın referans aldığı **tek blok** diyebiliriz. Şimdi terminalden önce "`cd`" komutuyla **ev** dizinine, oradan da "`cd .kodlabcoin`" komutuyla **gizli uygulama** dizinimize giriş yapıyoruz. Burada tekrar "`sudo nano debug.log`" komutunu çalıştırarak **debug** dosyasını açıyoruz. Dosyamız içerisinde, en alta doğru ilerledikçe üretilen `hash` değerlerini görmeye başlayacağız.



Şekil 1.92: Taralı olan kısma dikkat edin. Genesis Block üretim sürecini görüyorsunuz. Bu süreç nonce ve Genesis Block hash değerlerinin üretilmesiyle son buluyor.

Debug.log dosyamızın en alt tarafına indiğimizde `nonce` ve `Genesis Block` değerlerini görüyoruz. Bu değerleri kopyalayarak önceki adımlarda oluşturduğumuz ve **MerkleRoot** değerini kaydettiğimiz, **specs.txt** dosyasının içine kaydediyoruz. Böylece elimizde **MerkleRoot, GenesisBlock** ve **nonce** değerleri yer almış oluyor. Artık bu değerleri **checkpoints.cpp** ve **main.cpp** kaynak dosyalarımıza ekleyerek son aşamaya geçebiliriz.

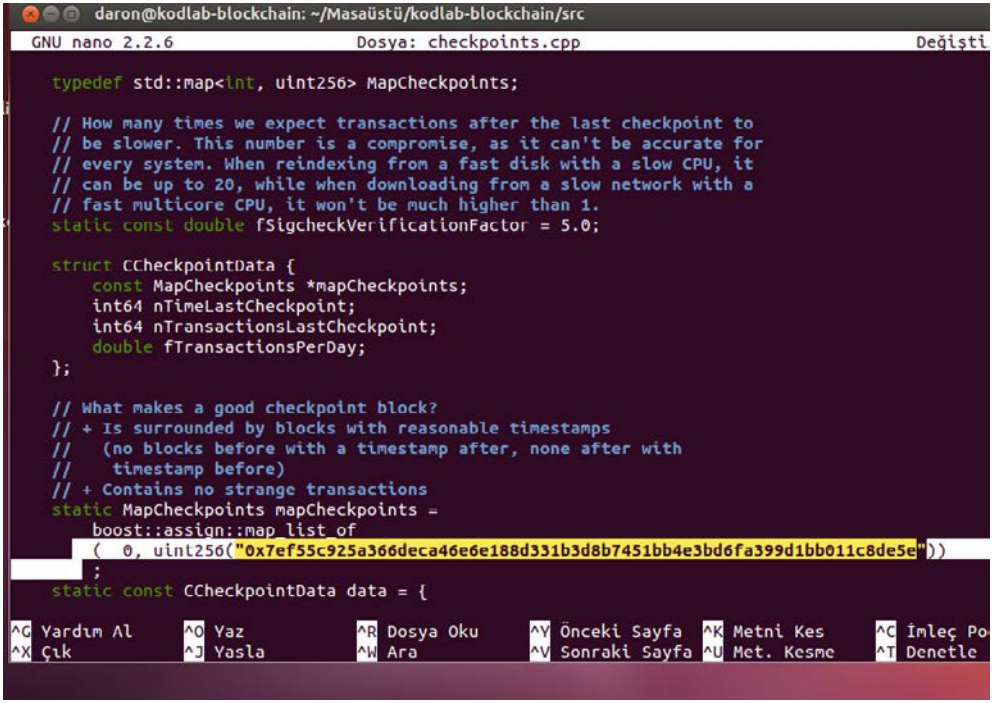


Şekil 1.93: Debug.log dosyamızda oluşturduğumuz nonce ve GenesisBlock hash değerleri görünüyor.

Terminal üzerinde **kodlab-blockchain/src** dizinine girerek aşağıdaki komutu çalıştırıyoruz.

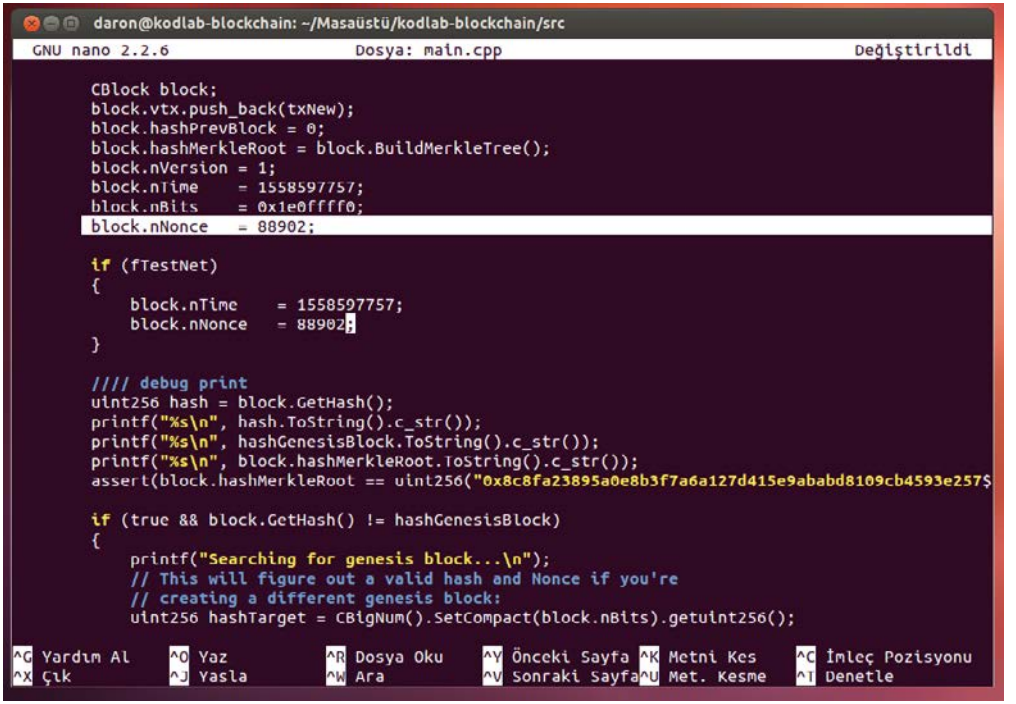
```
sudo nano checkpoints.cpp
```

Checkpoints.cpp dosyası içerisinde önceden sildiğimiz **GenesisBlock hash** kodları yerine, yeni oluşturduğumuz kodu yapıştırıyoruz.



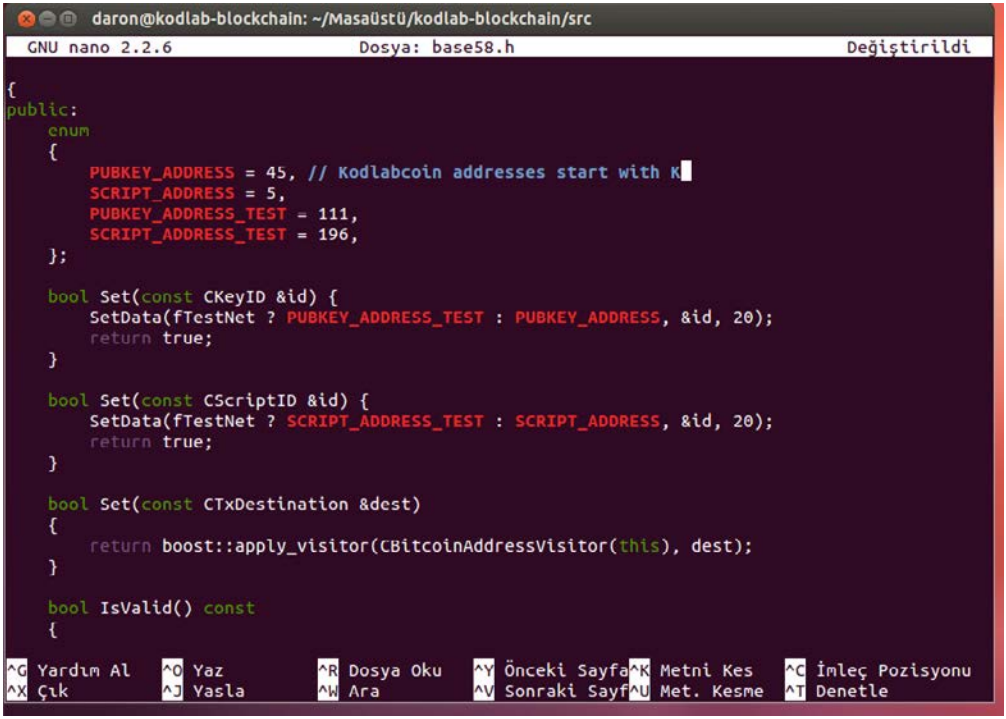
Şekil 1.94: checkpoints.cpp içerisine Genesis Block değerimizi giriyoruz.

Geriye **main.cpp** üzerinde ilgili yerlere **GenesisBlock** ve **nonce** değerlerini girmek kalıyor. Bunun için **main.cpp** dosyamızı açıyoruz. Arama özelliğini kullanarak "**hashgenesisblock**" anahtar sözcüğünü arıyoruz. Karşımıza gelecek olan iki farklı yere **hash** değerimizi giriyoruz. Ardından arama kutucuğuna "**nonce**" sözcüğünü giriyoruz ve ilgili iki yere de **nonce** değerimizi ekliyoruz.



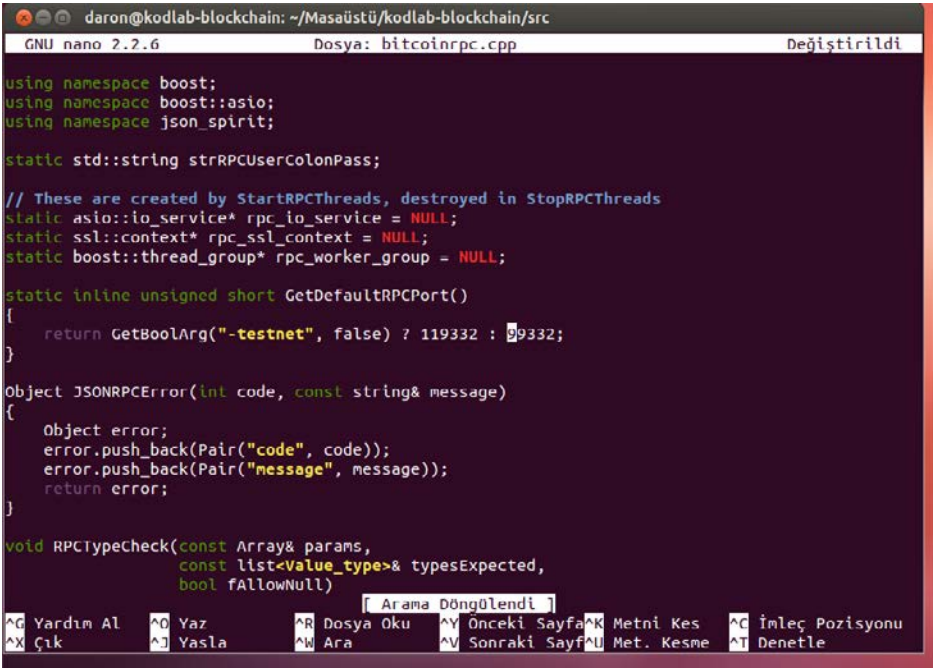
Şekil 1.95: Main.cpp üzerinde Genesis Block ve nonce değerlerini giriyoruz.

Artık majör değişiklik işlemlerimizi bitirdik. Şimdi geriye minör değişiklikler kalıyor. Bunlar cüzdan adreslerinin başlangıcı ve çalışma sırasında kullanılan portlar diyebilirim. Şimdilik bir adres ürettiğimizde, aslında Litecoin'den farklı olmayacak. Litecoin ağıyla bağlantı kuramayacak ve Litecoin alıp gönderemeyecek ama yine de aynı karakterde olacak. O yüzden adres oluştururken **Kodlab Coin** için spesifik bir adres başlangıcı seçmeliyiz. Bunu **base58.h** dosyası içerisinde yapacağız. Bu dosya içinde, Nano ile "**L**" araması yaptığımızda, Litecoin adres başlangıcını oluşturan satıra gideceğiz. Burada gördüğünüz "`PUBKEY_ADDRESS`" değeri olan "`48`", aslen **base58 hash** algoritmasına göre **L** harfinin karşılığı diyebiliriz. Ben kendi örneğimde bu değeri **K** ile değiştirmek istiyorum. Bunun için `48` değerini `45` ile değiştiriyorum. Açıklama satırındaki **L** harfini de **K** yapıyorum ve kaydederek dosyayı kapatıyorum. Siz farklı harf başlangıçları tercih edebilirsiniz. Bu anlamda şu adresi incelemenizi öneririm: *https://en.bitcoin.it/wiki/List_of_address_prefixes*



Şekil 1.96: base58.h dosyası içinde kripto paramızın adreslerini düzenliyoruz.

Şimdi sıra port numaralarına geldi. Geliştirdiğimiz Blockchain ağı, sunucuda çalışırken RPC portları olarak hâlâ Litecoin'in port numaralarını kullanıyor. Bu değerleri ileride borsa gibi yapılarda çakışma yaratmamak adına değiştirmemiz gerekiyor. Port numaraları genellikle **bitcoinrpc.cpp** dosyası içinde yer alır. Kaynak dosyamızı **Nano** ile açıp arama kutucuğuna "9332" yazmanız yeterli. 9332 Litecoin'in kullandığı **port numarası**dır. Bu port numarasını değiştirmediğimiz takdirde kripto paramız Litecoin ile aynı ortamda çalışmayacaktır. Dilediğiniz bir port numarası girebilirsiniz.



Şekil 1.97: bitcoinrpc.cpp dosyası içerisinde port numaralarını değiştiriyoruz.

Tüm işlemlerimiz tamamlandı. Artık son kez derleme işlemimizi yapıp çekirdek uygulamamızı devreye alabiliriz. Derleme işlemi için,

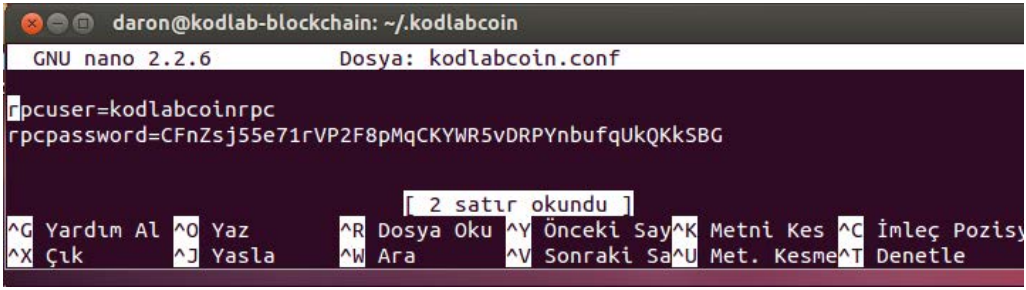
```
sudo make -f makefile.unix USE_UPNP=-
```

komutunu kullanıyoruz. İşlem tamamlandıktan sonra **kodlabcoind** çekirdek uygulamamızı çalıştırıyoruz. Uygulama hemen kapanarak size bir uyarı mesajı gösterecek. Bu uyarıyı görmeniz çok normal, her şeyin yolunda olduğunu gösteriyor. Uyarı içerisinde yer alan kullanıcı adı ve şifresini kopyalayarak **kodlabcoin.conf** dosyası içerisine eklemeniz gerekiyor. Bu **conf** dosyasını, ev dizini içerisinde yer alan **.kodlabcoin** gizli klasörü altında oluşturacağız.



Şekil 1.98: Tüm düzenlemeleri bitirip çekirdek uygulamamızı son kez derledikten sonra artık ağımızı kullanabilmek için conf dosyamızı düzenleyebiliriz.

Çekirdek uygulamamızın bize random atadığı **`rpc user`** ve **`rpc password`** değerlerini **kodlabcoin.conf** adlı ayar dosyamıza ekliyoruz. Bu dosya normalde otomatik olarak oluşturulmuyor. Nano ile bu dosyamızı oluşturabilirsiniz. Yapmanız gereken şey, boş bir metin dosyasına alt alta olacak şekilde kullanıcı adı ve parolayı yazarak **.conf** uzantılı olarak kaydetmek.



Şekil 1.99: Basit bir conf dosyası sadece kullanıcı adı ve şifresinden oluşabilir.

```
rpcuser=kodlabcoinrpc
rpcpassword=6n8LZ17Ftzrek543UYtNuZAu6BzVnE113HdF9pNBqE9T
```

Burada ağımız tarafından size verilen kullanıcı adı ve şifreyi hatırlamak zorunda değilsiniz. Hatta kendiniz de uydurabilirsiniz. Ancak web katmanından çekirdek uygulamamıza erişmek istediğimizde bu bilgilere ihtiyacımız olacak. O yüzden elinizin altında bulunmasında yarar var.

Madencilik yapmak istediğiniz zaman da **config** dosyasına ihtiyacınız olacak; tabii birkaç eklenti yapmak kaydıyla. Şimdi sizinle basit bir **config** dosyasının yapısına bakalım.

```
rpcuser=kodlabcoinrpc
rpcpassword=6n8LZ17Ftzrek543UYtNuZAu6BzVnE113HdF9pNBqE9T
rpcallowip=127.0.0.1
rpcport=99332
server=1
daemon=1
gen=0
```

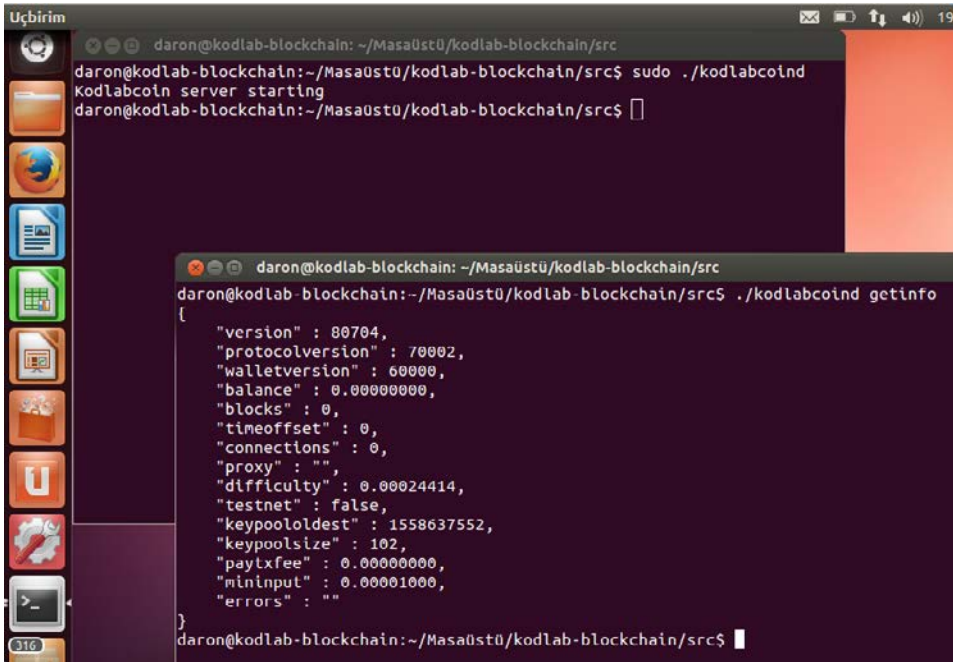
Yukarıdaki örnek **config** dosyası bizim hemen her işlemi yapabilmemiz için yeterli. Bu örneği kendi Blockchain ağınızın ve kripto paranızın özelinde uyarlayarak kullanabilirsiniz. İlk iki satırımız **kullanıcı adı** ve **şifremiz**. Bu satırlardaki bilgiler değişebilir. Yani bağımlı değilsiniz, dilediğiniz kullanıcı adını ya da şifreyi kullanabilirsiniz. Üçüncü satırımız **`rpcallowip`** değerini belirtiyor. RPC katmanını Blockchain üzerinde ve genellikle uzaktan işlemler gerçekleştirmek için kullanıyoruz. Hatta en çok web tabanlı işlemlerde yardımınıza koşacak diyebilirim. Blockchain ağımızın RPC üzerinden gelen komutları hangi IP adresi üzerinden alacağını bu satırda belirtiyoruz. RPC Allow IP değeri **yerel bir adres** olabileceği gibi **haricî bir IP adresi** de olabilir. RPC tarafına kanalize olabilmek için ağımızda özel bir port numarası tanımlamıştık, bu **port numarasını** dördüncü satırımıza yazıyoruz. Ben örnekte Kodlab Coin için belirlediğim port numarasını girdim. Siz kendi port numaranızı girmelisiniz. Bu önemli; çünkü RPC üzerinden çekirdek uygulamamıza, dinlediği port dışında bir porttan ulaşma şansımız yok. Beşinci satırımız, çekirdek uygulamamızı sunucu olarak çalıştırıp çalıştırmayacağımızı belirliyor. **`Server`** değeri **1** döndüğünde çekirdeğimiz sunucu özelliğini açmış demektir. Eğer **`server`** değerini **0** döndürürsek çekirdeğimiz sunucu özelliğini kapatmış demektir.

Config dosyamızda bu değer ne olursa olsun terminal üzerinden açıp kapatma şansımız var. Altıncı satırımız, **daemon** özelliğini açıp kapatmamızı sağlıyor. Bu özelliği açarsanız çekirdek uygulamamız arka planda sürekli çalışacak demektir. Kapatmanız hâlinde işletim sisteminizi başlattıktan sonra çekirdek uygulamamızı da başlatmanız gerekir. Son satırım **`gen`** diye tabir ettiğimiz **generate** özelliğinin **açık** ya da **kapalı** olmasını yönetiyor.

Eğer **gen** değerini **1** yaparsanız çekirdek dosyanız aynı zamanda **madencilik** de yapacaktır.

Bu özelliği açık bırakmak, sistem kaynaklarınızın tükenmesine neden olur.

Evet, geliştirdiğimiz Blockchain ağı üzerinde koşacak olan kripto paramızın ayar dosyası temel olarak bu bileşenlerden oluşuyor. Şimdi ağımızı çalıştırma zamanı geldi.



Şekil 1.100: Terminal üzerinden "sudo ./kodlabcoind" komutunu verdiğimizde Blockchain ağımız çalışmaya başlıyor. Başka bir terminal penceresi açıp "./kodlabcoind getinfo" komutunu vererek ağ durumunu izleyebilirsiniz.

Artık can dostumuz olan **Terminal** uygulamasını açıp Blockchain ağımızın bulunduğu dizine girmemiz gerekiyor. Ben kendi konumumu içeren komutu aşağıda paylaşıyorum.

```
cd Masaüstü/kodlab-blockchain/src
```

Ardından çalıştırılabilir çekirdek dosyamızı çalıştırıyorum.

```
sudo ./kodlabcoind
```

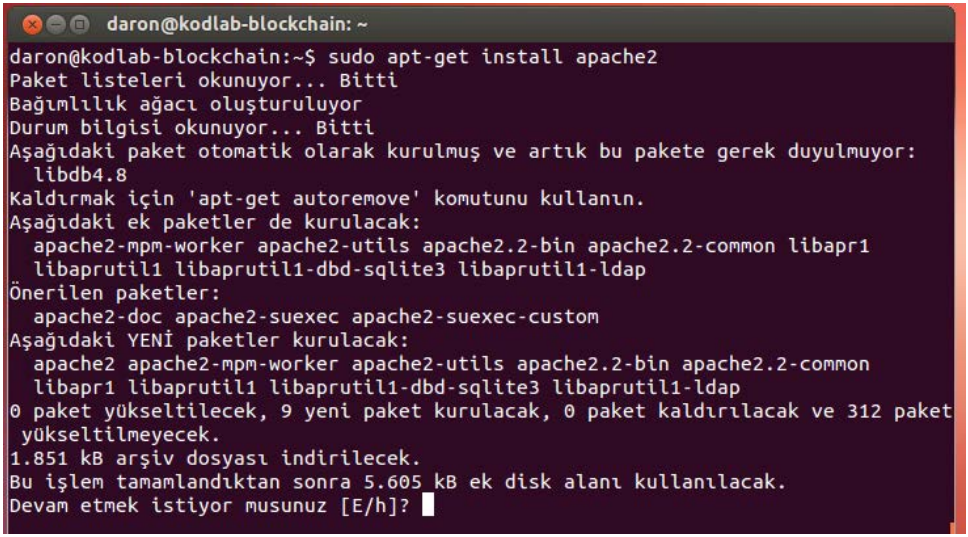
Buraya kadar her şey normal giderse terminal üzerinde "**Kodlabcoin server starting**" şeklinde bir bilgi çıktısı görünecek. Bu çıktıyı gördüğünüz an mutlu olabilirsiniz. Çünkü emeklerinizin karşılığı olan Blockchain ağ hayaliniz artık çalışıyor!

Şimdiye kadar hemen her şey tamam gibi duruyor. Ancak bizim makinemizde çalışan ve sadece bize hizmet veren bir Blockchain ağı istemediğinizi biliyorum. *Peki, ne yapabiliriz bu yapıyla?* Ben kendi örneğimde modellediğim iş fikrimi, web tabanlı ve Blockchain entegreli bir servis hâline getirmeyi hedeflemiştim. *Siz de ağınızı bu yapı içerisinde hazırlayabilirsiniz.*

Blockchain 1.0 ve Web

Buraya kadar hazırladığınız Blockchain ağını ve kripto parayı kullanarak web tabanlı iş modelleri kurgulamak çok kolay bir iş değil. Ancak bu aşamayı geçmeyi başarırsanız Blockchain ağınız ve kripto paranızı web uygulamaları içerisinde kullanmak oldukça kolay bir iş. Şimdi gelin, birkaç küçük örnek ile size bunu nasıl yapabileceğinizi göstereyim.

Öncelikle işe bir sunucu hazırlamakla başlıyoruz. Başlangıçta, sunucu için **en az 2 core CPU, 4 GB RAM** ve **60 GB** sabit disk yeterli olacaktır. Bu donanım üzerinde artık aşina olduğunuz Ubuntu Linux OS'u koşturmanızı şiddetle tavsiye ederim. Ubuntu ile dakikalar içerisinde makinenizi bir web sunucu hâline getirebilirsiniz. İşe öncelikle **Apache**, **PHP** ve **MySQL** kurulumlarıyla başlayacağız. Sunucu yapılandırma işini her okurun kendisinin yapmayacağını varsayarak bu konuya detaylı bir şekilde girmeyeceğim. Ancak aşağıdaki komutları kullanarak kendiniz de kolayca kurabilirsiniz.



Şekil 1.101: Ubuntu üzerinde Apache, MySQL ve PHP kurulumunu tamamlamak oldukça kolaydır.

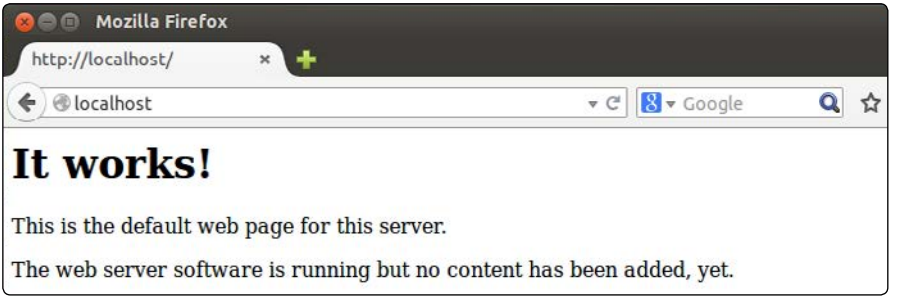
```
sudo apt-get update

sudo apt-get install apache2

sudo apt-get install mysql-server

sudo apt-get install libapache2-mod-php

sudo service apache2 restart
```



Şekil 1.102: Kurulum işlemi tamamlandıktan sonra tarayıcınızın adres kutusuna "localhost" yazarak test yapabilirsiniz.

Kurulum işlemi tamamlandıktan sonra kod yazmaya başlayabiliriz. Bu bölümde **PHP** ile **RPC** kullanarak önceki bölümde hazırladığımız Blockchain ağımızın çekirdek uygulamasına erişip terminalden yaptığımız işlemleri web ara yüzünden yapacağız. Web üzerinden Blockchain ağınızla konuşmaya başladıktan sonra PHP ile yapamayacağınız hiçbir şey yok diyebilirim. Bu aşamada PHP bilmiyorsanız bu bölümde paylaşacağım kodları herhangi bir metin editörü ile **.php** uzantısı vererek kaydedip kullanabilirsiniz. Ancak gelişmiş bir web yapısını Blockchain ağınızla harmanlamayı düşünüyorsanız mecburen **PHP** ya da **başka bir dil** (*mesela node.js*) öğrenmeniz gerekecek.

Başlamadan önce şunu söylemem gerekiyor: Önceki bölümde hazırladığınız Blockchain çekirdeğinin, web sunucusu üzerinden çalıştığını varsayarak anlatım ve kodlama yapacağım. Eğer web sunucunuz üzerinde çekirdek uygulamanızı çalıştırmak istemiyorsanız yerel IP bilgilerini değiştirmeniz gerekecek. Eğer web sunucunuz üzerine çekirdek uygulamanızı yüklemek veya web sunucunuzu Blockchain ağınızın bir düğümü yapmak istiyorsanız Blockchain çekirdeğimizi derlemeden önce Ubuntu'ya kurduğumuz paketleri sunucunuza kurmanız gerek. Ardından çekirdeğiniz web sunucusu üzerinde de çalışacaktır.

PHP ile (*ya da diğer bir dille*) Blockchain ağımıza komut gönderip çıktı almak istiyorsak öncelikle RPC istemcisini sunucu üzerinde web dosyalarıyla aynı yerde barındırmanız gerekiyor. Bu sayede PHP kodlarımız içerisine kolayca çağırabiliriz. Her işlem, PHP üzerinden, RPC katmanı aracılığıyla ağımıza giderek çıktılarını bize döndürecektir. Yani kodlarınız çalışmadığı taktirde öncelikle RPC dosyanızı kontrol etmenizi öneririm. Hadi, ilk kodumuzu yazalım!

```
<?php

require_once 'jsonRPCClient.php';

  $kodlabcoin = new jsonRPCClient('http://kodlabcoinrpc:6n8LZ17Ftzre
k543UYtNuZAu6BzVnE113HdF9pNBqE9T@127.0.0.1:99332');

  echo "<pre>\n";
  print_r($kodlabcoin->getinfo());
  echo "</pre>";
  echo "<br>";

  echo "<pre>\n";
  print_r($kodlabcoin->getbalance());
  echo "</pre>";
  echo "<br>";

  echo "<pre>\n";
  print_r($kodlabcoin->getmininginfo());
  echo "</pre>";
  echo "<br>";

?>
```

Bu kod bloğunu, hiç PHP bilmediğinizi varsayarak açıklamaya çalışacağım.

İlk satırla başlayalım. PHP kodları **<?php** başlığıyla başlar ve **?>** ile biter. Bu sayede HTML içerisine de kolayca PHP kodu yazabiliriz. Ancak bu örneğimizde HTML çıktıyı PHP ile üretiyorum. Yani **saf PHP** kodluyorum, herhangi bir framework kullanmıyorum. Bu alanda iyi bir framework ne yazık ki henüz yok.

İkinci satırımızda PHP kod bloğumuza bir başka PHP dosyasını çağırıyoruz. Bu dosya bizim işlemlerimiz için hayati öneme sahip olan **jsonRPCClient.php** dosyası. Ben bu dosyayı diğer PHP dosyalarımla aynı dizinde saklıyorum. Dolayısıyla herhangi bir yol belirtmiyorum. Aksi durumda çağırdığınız dosyanın yolunu da belirtmeniz gerekir.

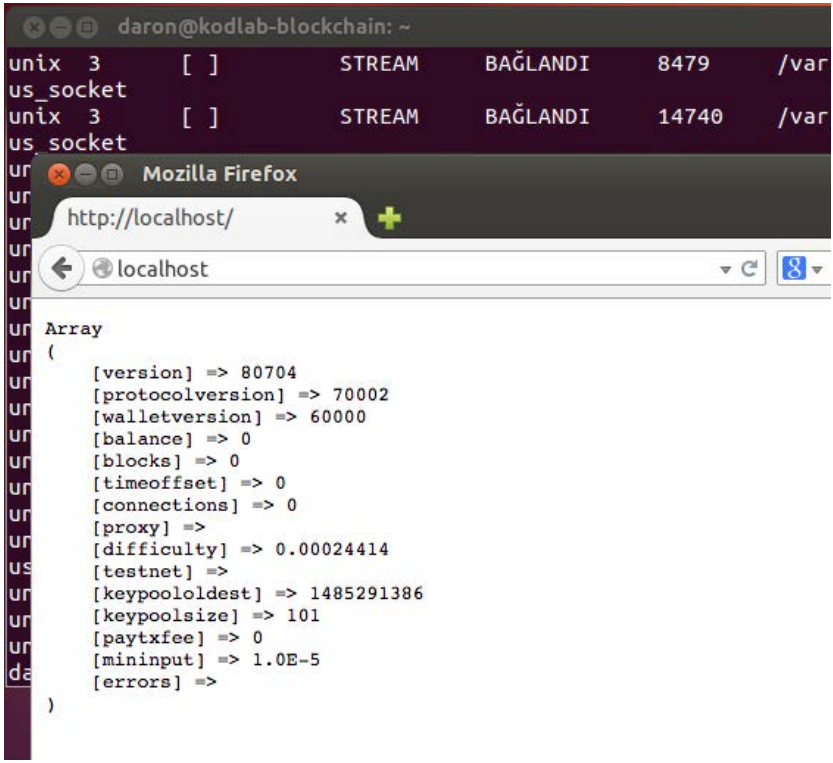
Üçüncü satırımızda `$` işaretini kullanarak PHP içerisinde bir **değişken** tanımlıyoruz. Bu değişken tahmin edeceğiniz üzere `kodlabcoin` adını verdiğim ve direkt olarak çekirdek uygulamamıza erişim sağlamamıza yarayan (*kullanıcı adı ve şifremizi de içeren*) anahtar değişkenimiz.

Dördüncü satırda `echo` komutunu görüyoruz. Bu komut bizim için ekrana yazı yazıyor. Bu satırda `<pre>\n` HTML kodunu yazdırıyoruz.

PHP, **sunucu** tarafından yorumlanan bir dildir. HTML ise **tarayıcılarımız** tarafından yorumlanır. Bu nedenle, bu dosyanın kaynak kodlarına bakacak olursak PHP kodlarını göremeyiz. HTML kodlarını PHP ile yazdırdığımızda tarayıcı sadece yazdırdığımız HTML çıktılarını görecek ve onları yorumlayarak bize **nihai sonucu** gösterecektir.

Beşinci satırımızda ekrana yazı yazdırmaya yarayan `print_r` (`$kodlabcoin->getinfo()`) komutunu kullandığımı görüyorsunuz. Bu komutu genellikle fonksiyon ya da değişken sonuçlarını yazdırmak için kullanırım. Bunun yerine bir önceki satırda kullandığım `echo` komutunu kullansaydım fonksiyonun sonucu değil, yazdığım kod, **olduğu gibi** ekrana basılacaktı. (`$kodlabcoin->getinfo()`) buradaki en dış parantezler `print_r` komutuna ait. `$kodlabcoin` değişkenine zaten çekirdek uygulamamıza erişim için kullanıcı adı ve şifre bilgileriyle **IP** ve **port** değerlerini girmiştim. `->` operatörü RPC dosyamıza ait bir işlemi yani `getinfo()` komutunu, Blockchain ağımıza sorup dönen sonucu `print_r` komutu sayesinde ekrana yazıyor. `getinfo()` ağ ile ilgili hemen her bilgiyi bizim için listeler.

Geriye kalan kodlar aynı olduğundan sadece RPC tarafına ait komutlara açıklama getirmek istiyorum. `getbalance()` komutu ağımızdaki **kripto para miktarı**nı ekrana yazar. `getmininginfo()` ise yapılıyorsa madencilik bilgilerini paylaşır.



Şekil 1.103: Gördüğünüz gibi PHP ve RPC ile Blockchain ağımızı web servislerinde kullanabiliyoruz.

Şimdi PHP ile nasıl **Kodlab Coin** adresi oluşturacağız ona bakalım.

```
<?php

require_once 'jsonRPCClient.php';

  $kodlabcoin = new jsonRPCClient('http://kodlabcoinrpc:6n8LZ17Ftzre
k543UYtNuZAu6BzVnE113HdF9pNBqE9T@127.0.0.1:99332');

  echo "<pre>\n";
  print_r($kodlabcoin->getnewaddress());
  echo "</pre>";
  echo "<br>";
?>
```

Bu kodda geçen **`getnewaddress()`** komutu bizim için yeni cüzdan adresleri oluşturur. **Üstelik sayfayı her yenilediğinizde!** Buna dikkat etmeyi unutmayın. Şimdi biraz daha gelişmiş bir örnek yapalım.

Örneğimizde bir üyelik sistemimiz var ve üyelerimizin hesapları arasında Kodlab Coin gönderimi yapabileceğimiz online bir hizmet sayfası hazırlıyoruz.

```
<?php

include ("session_tasi.php");
include ("fonksiyonlar.php");
include ("date_func.php");

// veritabanı bağlantısını yapıyoruz.
db_baglanti();

$query1 = "SELECT * FROM user WHERE kullanici='".$_
SESSION['kullanici']."'";
$result1 = mysql_query($query1);
$_yaz = mysql_fetch_array($result1);

require_once 'jsonRPCClient.php';

  $litecoin = new jsonRPCClient('http://kodlabcoinrpc:6n8LZ17Ftzrek5
43UYtNuZAu6BzVnE113HdF9pNBqE9T@127.0.0.1:99332');
  $wallet1id=$_SESSION['kullanici'];

// girilen adresin KodlabCoin cüzdan adresi olup olmadığına bakıyoruz
if (isset($_POST['addresstokodlabcoin'])){

$cashoutto = $_POST['addresstokodlabcoin'];
$cashoutvalue=$_POST['kodlabcoinamount']; //formumuzdan kaç kodlab coin
gönderileceğine bakıyoruz.

// aşağıdaki fonksiyon yukarıdaki bilgiler doğrultusunda bir adresten başka bir adrese yüzde 0.1
komisyon oranı düşülerek Kodlab Coin yolluyor
$cashouttotal=floatval($cashoutvalue-($cashoutvalue * 0.1));
     if ($cashouttotal>0){

            $kodlabcoin->sendfrom($wallet1id, $cashoutto,
$cashouttotal, 0);
```

```
    echo '<script type="text/javascript">';
    echo 'alert("Transaction is succesiful!");';
    echo 'location.href="wallet.php"';
    echo '</script>';
    } else {
    echo '<script type="text/javascript">';
    echo 'alert("Not enough for transaction fee");';
    echo 'location.href="wallet.php"';
    echo '</script>';
    }

}

?>
```

Yukarıda paylaştığım PHP dosyası, bir formdan gelen veriyi alarak RPC üzerinden Blockchain ağımıza iletip çekirdek uygulamamızdan geri dönen çıktıları da işleme alabildiğimizi gösteren küçük bir örnek sadece. Siz çok daha fazlasını yapabilirsiniz.

Bu konuya kitapta yer vermemin sebebi ufkunuzu açmak ve RPC kullanımını göstermekti. Sanırım amacıma ulaştım. O hâlde **Blockchain 1.0** uygulamalarını burada bırakıp Ethereum ile **Blockchain 2.0**'a geçelim.

Blockchain 2.0 ve Ethereum

Bu bölümde Ethereum ağında Blockchain uygulamalarına değineceğim. Genellikle akıllı sözleşme olarak tanımladığımız bu uygulamalar **Ethereum** ekibi tarafından geliştirilmiş olan **Solidity** programlama dili ile geliştiriliyor ve Ethereum ana ağı üzerinde **yan ağ** olarak faaliyet gösteriyor. Bu konuya değinebilmek için Solidity diline kısa bir giriş yapmak istiyorum. Aslen başlı başına bir kitap konusu olabilecek bu dili, fikir sahibi olmanız ve örnek uygulamalarımızı anlayabilmeniz için basit bir düzeyde anlatacağım.

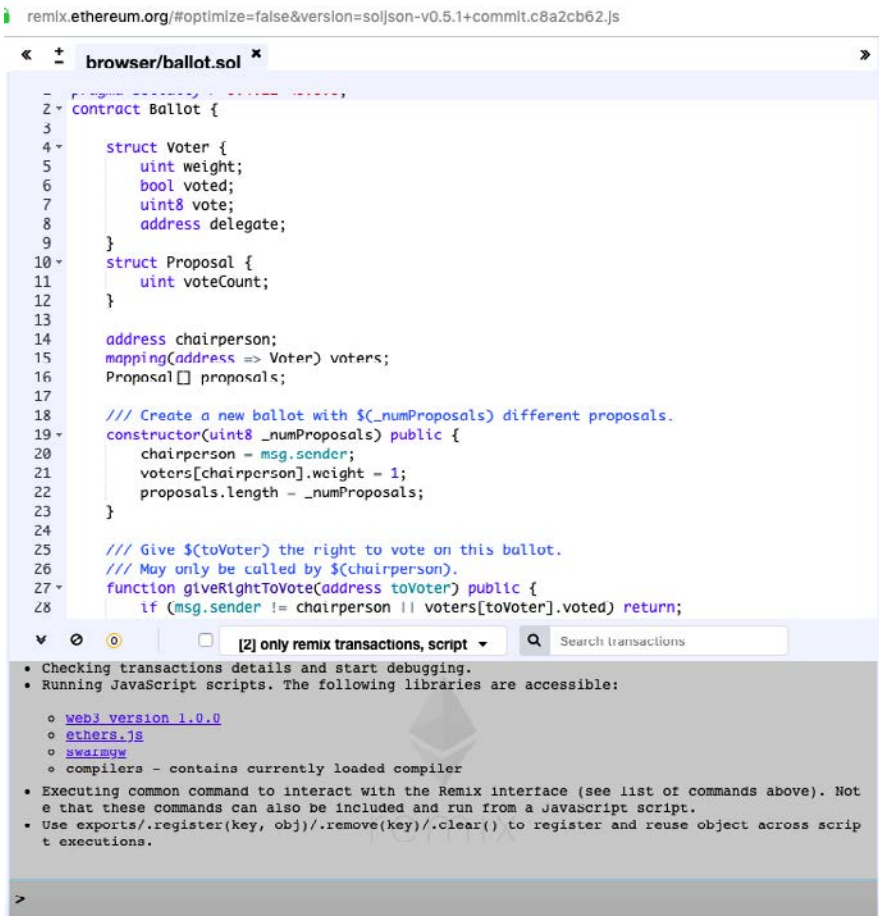
Solidity Programlama Dili

Solidity, akıllı sözleşmeler diye tabir ettiğimiz, Ethereum ağı üzerinde ve Ethereum sanal makinesi tarafından yorumlanan uygulamalar geliştirmeyi kolaylaştırmak için geliştirilmiş **nesne yönelimli, üst düzey bir programlama dili** olarak karşımıza çıkıyor. Akıllı sözleşmeler, Ethereum ağındaki işlemlerin davranışını yöneten programlardır. Yani sözleşme diye adlandırdığımıza bakmayın, aslen hepsi birer uygulama olarak kodlanıyor. Yapı gereği (*çok benzemese de*) **Java Snippet** dosyalarına benzetebiliriz.

Solidity, geliştirilirken hızlıca yazılımcılar arasında yayılma göstermesi hedeflenmiş. Bu yüzden fazlaca **C++**, **Python** ve **JavaScript**'ten esintiler taşıyor. Ancak en çok da Javascript'ten etkilenerek oluşturulmuş ve **Ethereum Sanal Makinesi**'ni (*ESM*) hedef alarak tasarlanmıştır diyebiliriz.

Solidity **statik** olarak yazılmıştır. Diğer tüm özelliklerinin yanında, nesneler arası miras ilişkisini, kütüphaneleri ve kullanıcı tanımlı karmaşık türleri desteklemektedir. Yani bu dille yapamayacağınız bir şey yok desem yalan olmaz. Kolayca *oylama*, *kitle fonlaması*, *açık artırma* ve *çoklu imza cüzdanları* gibi yüksek yapılı ve kompleks uygulamalar geliştirebilirsiniz. Kitap boyunca uygulama tabirini kullanacak olsam da aslen bunların tamamına "**akıllı sözleşme**" dendiğini aklınızdan çıkartmayın.

Şimdi gelelim, **Solidity** ile kodlama yapacağımız geliştirme ortamını hazırlamaya. Solidity ile Ethereum üzerinde akıllı sözleşme geliştirmek için kullanabileceğimiz birçok yol var. Ben bunların en popüler olanlarını sizlerle paylaşmak istiyorum. Önceliği, favorim olan **Remix**'e vermek istiyorum.



Şekil 1.104: Remix ihtiyaç duyduğunuz hemen her şeyi sunan online bir IDE seçeneği.

Remix: Online olarak çalışan bu IDE, direkt olarak Ethereum ağına bağlı ve yazdığınız kodları anlık olarak işleyip sonuçlarını hemen görebilmenize olanak veriyor. *remix.ethereum.org* adresinden ulaşabileceğiniz bu online IDE ile yazdığınız kodu **anlık** olarak izleyebilir, ihtiyaçlarınız doğrultusunda derleyici opsiyonlarını değiştirebilirsiniz.

Tamamen online olmasının en büyük handikabı, çok büyük yapılı uygulamaları yazarken çakılma yaşamayacağınızın bir garantisinin olmaması. Benim başıma birkaç kez gelmişti. Bu yüzden ekip hâlinde ve kompleks yapılar üzerinde çalışmayı hedefliyorsanız diğer IDE alternatiflerime bakmanızı öneririm.

node.js: Eğer geliştirme yaparken **Javascript** ve **node.js** ile haşır neşir olduysanız size sevineceğiniz bir haberim var. Node.js ile kurup kullanabileceğiniz bir Solidity IDE'miz var: **Solc-js**. ve Node.js ile hayatımıza giren **Node Package Manager** (*kısaca npm*) ile kolayca kurulan Solc-js, size Node.js ortamında Solidity ile akıllı sözleşme yazma olanağı sunuyor.

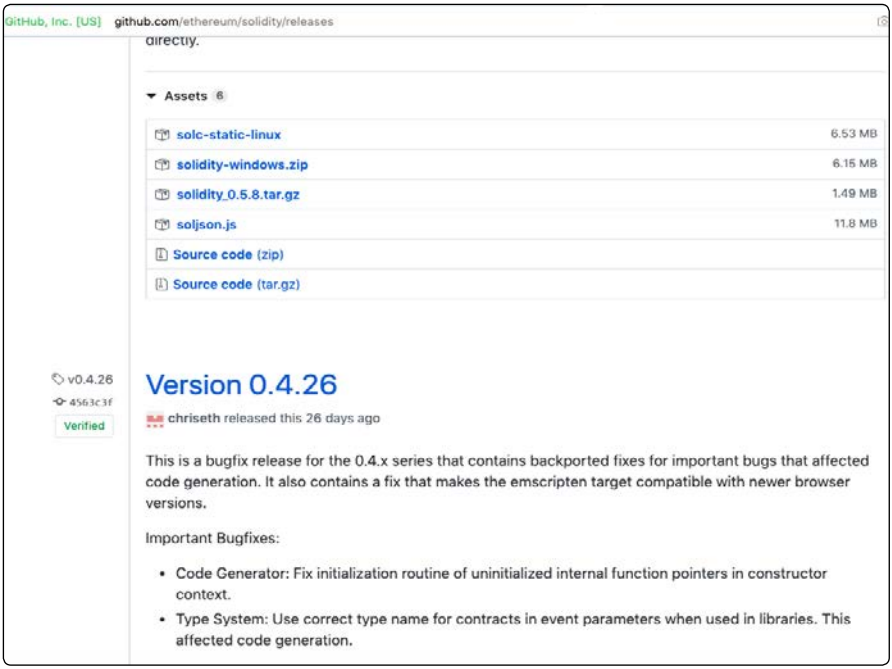
```
npm install -g solc
```

Terminal üzerinden yukarıdaki komutu kullanarak Solc-js'yi kurabilirsiniz. Son yıllarda oldukça popüler olan **Docker** de **Solc**'a destek veren yapılardan. Eğer hâlihazırda Docker ile çalışıyorsanız kolayca Solc'u kurup kullanmaya başlayabilirsiniz.

```
docker run ethereum/solc:stable --version
```

Yukarıdaki komutu kullanarak Docker derleyicisiyle Solc'u kurmuş olacaksınız.

İkili paketler: Eğer herhangi bir derleyici kullanmak yerine alıştığınız IDE ortamlarını kullanmak istiyorsanız sorun değil. Windows ve Linux işletim sistemlerine yönelik hazırlanmış, kurulabilir Solidity paketleri mevcut.



Şekil 1.105: Solidity'nin Github reposu üzerinden kurulabilir paketlere ulaşabilirsiniz.

Hele ki Ubuntu Linux kullanıyorsanız çok şanslısınız; çünkü Ethereum ekibi, Ubuntu kullanıcıları için özel bir **repo** hazırlamış. Bu repoyu sisteminize kurarak en güncel yapılara kavuşabilirsiniz. Kurulum için şu komutları kullanıyoruz:

```
sudo add-apt-repository ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install solc
```

Eğer ekip tarafından her gece düzenlenen "**gecelik sürümleri**" kullanmayı tercih ediyorsanız aşağıdaki komutları kullanabilirsiniz. Fakat ben bunu önermiyorum.

```
sudo add-apt-repository ppa:ethereum/ethereum
sudo add-apt-repository ppa:ethereum/ethereum-dev
sudo apt-get update
sudo apt-get install solc
```

Ubuntu dışında bir Linux dağıtımı kullanıyorsanız kaynakların biraz zayıf olduğunu söylemem gerekiyor. Ancak aşağıdaki komut, Ubuntu dışındaki Linux dağıtımları için bir nebze çözüm olabilir.

```
sudo snap install solc
```

MacOS veya Windows kullanıyorsanız sisteminize özel çözümler bulunuyor. Windows kullanıcıları **Visual Studio** gibi güçlü bir IDE kullanabilirken Mac kullanıcıları **Xcode IDE** ortamını tercih edebiliyor. Windows kullanıcıları `git` reposunu klonlayarak Visual Studio üzerinde akıllı sözleşme geliştirmeye başlayabilirler.

```
git clone --recursive https://github.com/ethereum/solidity.git
cd solidity
```

Mac üzerinde Homebrew imdada yetişiyor.

```
brew update
brew upgrade
brew tap ethereum/ethereum
brew install solidity
```

Eğer yukarıdaki komutları çalıştırdığınızda uygun sürüm bulunamazsa aşağıdaki komutu kullanabilirsiniz.

```
brew unlink solidity
# Install 0.4.8
brew install https://raw.githubusercontent.com/ethereum/homebrew-eth
ereum/77cce03da9f289e5a3ffe579840d3c5dc0a62717/solidity.rb
```

Kurulum ve ön hazırlık aşamalarını tamamladıktan sonra artık Solidity'nin nasıl bir dil olduğuna yakından bakma zamanı geldi. Önceki bölümde de bahsettiğim gibi normal şartlarda Solidity başlı başına bir kitap olabilecek nitelikte. Yani burada dile küçük bir adım atacak, basit örnekler ve sonrasında kapsamlı örnekler vererek fikir sahibi olmanızı sağlayacağım.

Hızlandırılmış Solidity

Solidity de esinlendiği Javascript, C++ ve Python gibi **belli başlı bölümler**den oluşur. Kodlamalar içerisinde isteğe bağlı sayıda **sözleşme tanımı, import** ve **pragma yönergeleri** içerir. Hemen her dilde olduğu gibi Solidity de haricî kaynakları kodlarımız içerisine eklememize izin verir.

`Pragma` anahtar sözcüğü, belirli derleyici özelliklerini veya denetimlerini etkinleştirmek için kullanılır. Bir **`pragma`** yönergesi her zaman yerel olarak kullanılır. Eğer kod yazarken modüler çalışıyorsanız ve birden çok yerde birden çok özelliği etkinleştirmek istiyorsanız projenizdeki tüm kaynak kod dosyalarınıza, **`pragma`** yönergesini eklemeniz gerekiyor. Eğer import yönergesiyle kodunuz içerisine başka bir dosya çağırırsanız o dosyadan gelen **`pragma`**, içe aktarılan dosyaya otomatik olarak uygulanmaz. Yani o kodlara da **`pragma`** yönergesini eklemeniz gerekecek.

Projelerde kullanılan **`pragma`** yönergeleri bazen farklı sürümlerle gelen farklı özellikler nedeniyle kodunuzda uyumsuzluklara yol açabilir veya dosyanızın çalışmasını engelleyebilir. Bu tip durumların önüne geçmek için pragma sürümünün kodun başında açıklanması gerekir. Bu sayede yeni pragma versiyonları ile gelecek değişikliklerin kodunuzda sorun yaratmasına engel olmuş olursunuz. Hele ki Ethereum ekibinin gecelik sürümler çıkarttığını göz önüne alırsak özellikle **0.x.0** veya **x.0.0** şeklinde görünen ve büyük değişiklikler içeren sürümler yayımlandıkça değişiklik listesini incelemenizi öneririm.

Standart bir pragma sürümü aşağıdaki gibi görünür:

```
pragma solidity ^0.5.2;
```

Kaynak kodunuz yukarıdaki gibi bir **`pragma`** yönergesi içeriyorsa akıllı sözleşmenizin kodları **0.5.2** sürümünden önceki bir derleyici ile derlenmeyecek ve ayrıca **0.6.0** sürümünden başlayan bir derleyici üzerinde çalışmayacak demektir. Mevcut sürümden daha yeni bir sürümle derlemenin yapılamamasını **^** işaretini sürüm numarasının önüne ekleyerek sağlıyoruz. Bunun arkasındaki fikir, 0.6.0 sürümüne kadar herhangi bir kırılma değişikliği yaşanmayacağından, kodumuzun **istediğimiz şekilde** derleneceğinden her zaman emin olabileceğimizdir. Çıkan hata düzeltme sürümleri ise kaynak kodumuzda sorunsuz çalışır. Güvenlik yamalarını değişiklik içeren sürüm güncellemelerinden saymıyoruz.

Pragmalar, **deneysel pragma** olarak bilinen başka bir türe de sahiptir. Derleyici veya dilin henüz varsayılan olarak etkin olmayan özelliklerini denemek için kullanılır. Deneysel pragmalarla açabileceğiniz yeni nesil özellikler, kendilerini ispat ettiklerinde standart pragmalarla da kullanılabilir hâle gelirler.

Bu yüzden özellikle son kullanıcıları ilgilendiren halka açık sözleşmeler geliştirirken deneysel pragmaları kullanmanızı önermiyorum. Kendiniz için bir **sandbox** ortamı yaptıysanız ve burada yenilikçi kodlar denemek istiyorsanız deneysel pragmaları kullanabilirsiniz. İnternette deneysel pragmalar hakkında kaynak bulabilirsiniz.

`Pragma` kadar çok kullanacağımız bir diğer yönergemiz **`import`** demiştim. Solidity, yüksek yapılı dillerden alıştığımız varsayılan **`export`** ifadesini tam olarak yorumlayamaz, ama JavaScript'te (*ES6'dan itibaren*) bulunana benzer **`import`** ifadesini destekler. Böylece kolayca haricî kütüphanelerden ya da önceden yazdığınız kodlardan kolayca yararlanabilirsiniz.

Kullanıma örnek olarak aşağıdaki kodda yer aldığı şekilde **`import`** ifadesini projelerinizde kullanabilirsiniz.

```
import "filename";
```

Bu ifade, tüm global sembolleri "**filename**" belgesinden, çalışmakta olduğumuz dosyanın içine aktarır. Yine de yukarıdaki kullanım pratikte çok önerilmez,

çünkü ad alanını tahmin edilemez bir şekilde büyütür ve gereksiz birçok isimle uygulamayı şişirir. "**filename**" belgesi içine yeni üst düzey ögeler eklerseniz bu değişiklik, "**filename**" dosyasından yukarıdaki şekilde içe aktarılan tüm dosyalara otomatik olarak yansıtılır. Haricî dosyalarla çalışırken içe aktardığınız veriler için kullanacağınız açıklamada daha spesifik ifadeler kullanmanızı tavsiye ederim. Böylece olası karışıklıkların önüne geçmiş olursunuz.

Bunu küçük ve basit bir örnekle anlatmaya çalışayım. Aşağıdaki koda bakacak olursanız, "**filename**" belgesindeki tüm global sembolleri içeren "**`symbolName`**" isimli yeni bir global sembol oluşturduğumuzu görürsünüz. Bu tarz bir uygulama size gereksiz gelebilir, ancak bazen özel bir ayıklama sonucu derlediğimiz global sembol isim alanları kullanmak isteriz.

Aşağıdaki kod bu tarz uygulamalar için giriş mahiyetinde bir örnektir.

```
import * as symbolName from "filename";
```

Bir ad çakışması varsa içe aktarırken sembolleri de yeniden adlandırabilirsiniz. Bu kod, "**filename**" belgesi içindeki "**`symbol1`**" ve "**`symbol2`**" isimli iki global sembole kendi dosyamız içinde sırasıyla "**`alias`**" ve "**`symbol2`**" isimli iki yeni **global sembol** tanımlar.

```
import {symbol1 as alias, symbol2} from "filename";
```

ES6'nın bir parçası olmayan ama yine de yeterli olabilecek bir başka kullanıma bakalım:

```
import "filename" as symbolName;
```

Yukarıdaki kod, yukarıdaki örnekte kullandığımız **`import * as symbolName from "filename"`**;' ye eşdeğerdir.

Haricî dosyalarla çalışarak kodlama yapıyorsanız çağırdığınız dosyaların konumları zaman zaman sorun çıkartabilir. Diğer programlama dillerinde olduğu gibi içe aktarılmak istenen dosya adı her zaman "`/`" işareti ile hedefe yönlendirilir. Geçerli olan dosya konumu "`.`" işareti ile belirtilirken "`..`" ana dizin dosyasını işaret eder. "`.`" veya "`..`" kendisinden sonra "`/`" işareti dışında bir karakter tarafından takip ediliyorsa geçerli dosya komutu veya ana dizin kastedilmiyor demektir.

"." veya ".." ile başlamadıkça tüm dosya komutları, geçerli olan dizin referans kabul edilerek **direkt yönlendirme** olarak kabul edilir.

Üzerinde çalıştığınız kaynak kod dosyanızla aynı dizinde bulunan başka bir dosyayı içeriye çağırmak için, `import ./x as x` ifadesini kullanırız. Eğer `import "x" as x` ifadesini kullanırsanız `import` yönergemiz farklı bir dosyayı çağırmaya çalışabilir.

Kodlama yaparken her zaman `import "./filename.sol";` gibi göreceli komutları kullanmayı tercih edin ve ".." gibi konum belirteçlerini kullanmaktan kaçının. İkinci yöntemi kullanmanız durumunda, **global konum belirteçleri**ni kullanarak yeniden konumlandırma ayarları yapmak muhtemelen daha iyi sonuç verecektir. Tabii benimki sadece bir öneri, siz dilediğiniz yöntemi kullanmakta özgürsünüz.

Bu arada hangi derleyiciyi kullandığınızdan bağımsız olarak derleyicinizi, bir dosya konumunun nasıl ifade edilmesi gerektiği konusunda dilediğiniz gibi ayarlayabilirsiniz. Örneğin, `github.com/ethereum/dapp-bin/library` sanal dizininden içe aktarılan her şeyin aslında yerel dizininiz olan `/usr/local/dapp-bin/library` üzerinden okunması için derleyicinizde yeniden düzenleme yapabilirsiniz.

Birden fazla yorum ayarı uyguladığınızda, önce en uzun anahtara sahip olan yorumlama yöntemi denenir. Söylememe gerek yok sanırım, ama boş bir ön ek kullanmanıza izin verilmez. Kullandığınız ortam, aynı adı taşıyan bir kütüphanenin farklı sürümlerini içe aktarmak için paketleri yapılandırmanıza izin veren bir fonksiyona sahip olabilir. Böylesi bir durumda kodlamaya hangi sürüm ile başladığınızı unutmayın ki sonradan derleme aşamasında çakışma yaşamayın.

Bunu daha iyi anlatabilmek için, kitabımızın ilerleyen bölümlerinde kodlama yaparken de kullanacağım **Remix** üzerinde küçük bir örnekleme yapalım.

Remix, GitHub için ön tanımlı bir **konum yorumlama** özelliği sunar. Bu sayede siz hiçbir düzenleme yapmasanız bile hedef dosyayı ağ üzerinden otomatik olarak alır. Tabii sadece GitHub üzerinde yapar bunu. Yinelenebilir eşlemeyi farklı biçimlerde içe aktarabilirsiniz.

Örneğin:

```
::
import "github.com/ethereum/dapp-bin/library/iterable_mapping.sol"
as it_mapping;
```

Bir programlama dilinde kod blokları arasında yorumlar ve açıklamalar yazmak bence çok önemli bir durum. Hele ki **spagetti** tabirini kullandığımız ve bizden sonra projeye dâhil olan yazılımcıların kodu çözümlemesine engel olan tarzda kodlama yapmak istemiyorsanız yorum satırları ve açıklamalar çok önemlidir. **Solidity** bu anlamda bize alışık olduğumuz bir yapı sunar. Tek satırdan oluşan yorumlar için "//" işaretlerini kullanıyoruz. Çoklu satır yorumları için "/…/" işaretlerini kullanıyoruz.

Hemen örnekleyelim.

```
// Bu tek satırlık bir yorum.

/*
Bu çok satırlı
bir yorum.
*/
```

Bu yöntemlerden hariç bir yorum işaretimiz daha var. **Natspec yorumu** adı verilen bu yorum türü üçlü eğik çizgi "///" veya çift yıldız işareti "/ ** … * /" ile yazılır ve doğrudan işlev bildirimlerinin veya ifadelerinin üzerinde kullanılır. İşlevleri belgelemek, resmî doğrulama için koşulları açıklamak ve bir işlevi çağırmaya çalıştıklarında kullanıcılara gösterilen bir onay metni sağlamak için tercih edilir.

Aşağıdaki örneğimiz, sözleşmenin başlığını, iki işlev parametresinin ve iki dönüş değişkeninin açıklamasını içeriyor.

```
pragma solidity >=0.4.0 <0.6.0;

/** @title Hesap makinesi. */
contract HesapMakinesi {
    /** @dev Alan hesaplamalarında kullanabilirsiniz.
        * @param w Boy.
        * @param h En.
        * @return s Alan.
```

```
        * @return p Çevre.
        */
    function rectangle(uint w, uint h) public pure returns (uint s,
uint p) {
        s = w * h;
        p = 2 * (w + h);
    }
}
```

Sözleşme Dediğimiz Uygulamalar

Önceki bölümlerde akıllı sözleşme dediğimiz yapının aslen Ethereum ağı üzerinde geliştirdiğimiz uygulamalar olduğunu söylemiştim. Şimdi bu yapıya biraz daha yakından bakacağız. **Solidity** dilinde yazılan sözleşmeler, nesne yönelimli dillerdeki sınıflara benzer. Her sözleşme, durum değişkenleri, fonksiyonlar, fonksiyon düzenleyicileri, olaylar, yapı tipleri ve enum tipleri bildirimlerini içerebilir. Ayrıca, sözleşmeler bu bileşenleri diğer sözleşmelerden de **`import`** yoluyla alabilir.

Şimdi bu yapıya yakından bakalım.

Durum değişkenleri, değerleri kalıcı olarak sözleşme deposunda saklanan değişkenlerdir.

```
pragma solidity >=0.4.0 <0.6.0;

contract SimpleStorage {
    uint storedData; // State variable
    // ...
}
```

Fonksiyonlar, bir sözleşmedeki yürütülebilir kod birimleridir.

```
pragma solidity >=0.4.0 <0.6.0;

contract SimpleAuction {
    function bid() public payable { // Fonksiyon
        // ...
    }
}
```

Fonksiyonlar, kodlama aşamasında kaynak kod içerisinde dâhilî veya haricî olarak kullanılabilir. Bunun yanında diğer sözleşmelere karşı farklı görünürlük seviyelerine sahip olabilirler. Son olarak fonksiyonlar parametreleri ve döngü (*return*) değerlerini kabul ederler.

Fonksiyon değiştiricileri, fonksiyonların semantiğini bildirimsel bir şekilde değiştirmek için kullanılırlar.

Bir örnek verelim:

```
pragma solidity >=0.4.22 <0.6.0;

contract Purchase {
    address public seller;

    modifier onlySeller() { // Fonksiyon değiştirici
        require(
            msg.sender == seller,
            "Only seller can call this."
        );
        _;
    }

    function abort() public view onlySeller { // Değiştirici kullanılıyor
        // ...
    }
}
```

Olaylar, EVM kayıt sistemleri için kolaylık sağlayan arayüzlerdir.

```
pragma solidity >=0.4.21 <0.6.0;

contract SimpleAuction {
    event HighestBidIncreased(address bidder, uint amount); // Olay

    function bid() public payable {
   // ...
        emit HighestBidIncreased(msg.sender, msg.value); // Tetikleyici Olay
    }
}
```

Yapılar, çeşitli değişkenleri gruplayabilen özel tanımlanmış türlerdir. Aşağıdaki örnek kodda tanımlanmış bir yapı tipi paylaşıyorum.

```
pragma solidity >=0.4.0 <0.6.0;

contract Ballot {
    struct Voter { // Yapı tipi
        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }
}
```

Enumlar, sonlu bir "**sabit değer**" kümesine sahip özel türler oluşturmak için kullanılabilir.

```
pragma solidity >=0.4.0 <0.6.0;

contract Purchase {
    enum State { Created, Locked, Inactive } // Enum değer
}
```

Sözleşme türleri: Solidity, PHP gibi dinamik bir dil değildir. Tersine, statik olarak kurgulanmış bir dildir. Bu nedenle kod yazarken tanımlayacağımız her değişkenin (*global ve yerel*) türünün mutlaka belirtilmesi gerekir.

Solidity, karmaşık sözleşme türleri oluşturmaya imkân verebilmek için birleştirilebilir birkaç temel sözleşme tipi sağlar. Bunların dışında **kütüphaneler** ve **arayüzler** olarak adlandırılan özel tür sözleşmeler de bulunur.

Ayrıca, türler, operatör içeren ifadelerde birbirleriyle etkileşime girebilir.

Dikkat edilmesi gereken bir diğer nokta da "`Undefined`" veya "`null`" değerleri kavramının **Solidity** dilinde yer almadığıdır. Bunun yerine yeni bildirilen değişkenler her zaman türüne bağlı olarak varsayılan bir değere sahip olmalıdır. Beklenmeyen değerleri işlemek için tüm işlemi geri döndürecek `revert` fonksiyonunu kullanmalı veya başarıyı gösteren ikinci bir `bool` değeri olan bir **tuple** döndürmelisiniz.

`Bool` her zaman `True` veya `False` sabit değerini alır.

Operatörler ve İşlemler: Programlama dillerinde olmazsa olmaz yardımcılarımız operatörler ve işlem sabitleridir.

Solidity aşağıdaki operatörleri kullanımımıza sunar:

!	(mantıksal olumsuzlama)
&&	(mantıksal bağlantı, "ve")
||	(mantıksal bağlantı kesilmesi, "veya")
==	(eşitlik)
! =	(eşitsizlik)

Karşılaştırma operatörleri;

`<=, <, ==,! =,> =,>` (Bool olarak değerlendir)

Bit operatörleri;

`&, |, ^` (bitsel özel ya da), `~` (bitsel olumsuzlama)

Vardiya operatörleri;

`<<` (sola kaydırma), `>>` (sağa kaydırma)

Aritmetik operatörler;

`+, -, unary -, *, /,%` (modulo), `**` (üstelleştirme)

`||` ve `& &` operatörleri ortak kısa devre kurallarını uygular. Bunun anlamı `f (x)` || `g (y)` işleminde, `f (x)` doğru olarak değerlendirilirse işlemin olası yan sonuçları olsa bile `g (y)` fonksiyonu değerlendirmeye alınmaz.

Tam Sayılar; int/uint: Çeşitli boyutlarda işaretli ve işaretsiz tam sayıları tanımlamak için kullanılır. `uint` ve `int`, sırasıyla `uint256` ve `int256`'nın yerine de kullanılabilir.

Bit işlemleri, bir sayının iki farklı tamamlayıcı temsil değeri üzerinde yapılan işlemlerdir.

Örneğin `~int256(0) == int256(-1)`.

Dört işlem: Toplama, çıkarma ve çarpma işlemlerinde standart semboller kullanılır. Bu işlemler, tam sayıların tamamlayıcı temsil değeri için de geçerlidir. Örneğin:

```
uint256 (0) - uint256 (1) == 2 ** 256 - 1
```

Güvenliği artırılmış merkezsiz uygulamalar tasarlarken bu kullanımları ve sonuçlarını hesaba katmanız faydalı olacaktır.

Bölme: Bir işlemin sonucunun türü her zaman işlenenlerden birinin türü olacağından, tam sayıların bölünmesi her zaman bir tam sayı ile sonuçlanır. Solidity dilinde, bölme sıfıra doğru yuvarlar. Bu,

```
int256 (-5) / int256 (2) == int256 (-2)
```

anlamına gelir. Bununla birlikte eğer bir değeri sıfıra bölmeye kalkarsanız işleminiz hataya neden olur.

Üstel İfadeler: Üstel ifadeler yalnızca atanmamış türler için kullanılabilir. Uygulamanızda kullandığınız türlerin, sonucu içerecek kadar büyük olduğundan ve potansiyel wrap hatası için önlem aldığınızdan emin olun. Ayrıca EVM tarafından `0 ** 0` değerinin ön tanımlı olarak **1** tanımlandığını unutmayın.

Adres yapıları: Solidity adres konusunda iki farklı yapıya sahiptir. İlki 20 baytlık bir değere sahip olan standart **Ethereum** adresleridir. Diğeri ise adres yapısından temelde farksız olan ödeme adresidir. Ödeme adresleri ek üyelerle transfer veya gönderme amacıyla kullanılır.

Bu ayrımın ardındaki fikir, ödeme adresi Ether gönderebileceğiniz bir adres olduğu hâlde düz bir adres ile Ether gönderilemeyeceğidir.

Adres tür dönüşümleri: Ödeme adresinden normal adrese örtülü dönüşümlere izin verilirken normal adresten ödeme adresine yapılan işlemlerde dönüşüm mümkün değildir. Herhangi bir adres, dolaylı olarak ödeme adresine dönüştürülebilir.

Balance ve transfer: Balance özelliğini kullanarak bir adresin bakiyesini sorgulamak ve transfer işlevini kullanarak ödenebilir bir adrese **Ether** (*wei birimlerinde*) göndermek mümkündür.

```
address payable x = address(0x123);
address myAddress = address(this);
if (x.balance < 10 && myAddress.balance >= 10) x.transfer(10);
```

Cari sözleşmenin bakiyesi yeterince büyük değilse veya Ether transferi alıcı hesap tarafından reddedilirse **transfer işlevi başarısız olur**. Aktarma işlemi hata olarak geri döndürülür.

`Send`, düşük seviyede transferle aynı işleve sahiptir. Yürütme başarısız olursa mevcut sözleşme bir istisna sebebiyle durmayacak; ancak gönderme **`false`** döndürecektir.

`Send` kullanımında bazı tehlikeler de mevcuttur: Çağrı yığını derinliği **1024**'te ise (*ki bu her zaman arayanlar tarafından zorunlu tutulabilir*) aktarım başarısız olabilir veya alıcı Gas'ın bitmesi durumuyla karşılaşılabilir. Bu nedenle, güvenli Ether transferlerini yapmak için, her zaman **`send`** döngü değerini kontrol edin, transfer kullanın veya daha iyisi, alıcının para çekme isteği göndereceği bir yöntem tercih edin.

Call, Delegatecall ve Staticcall: ABI'ya uymayan sözleşmelerle arayüz oluşturmak veya kodlama üzerinde daha doğrudan kontrol sahibi olmak için, **`call`**, **`delegatecall`** ve **`staticcall`** fonksiyonları sağlanır. Hepsi **tek bayt**lık bir bellek parametresi alır ve başarı koşulunu (*bool olarak*) ve döndürülen verileri döndürür. **`abi.encode`**, **`abi.encodePacked`**, **`abi.encodeWithSelector`** ve **`abi.encodeWithSignature`** fonksiyonları yapılandırılmış verileri kodlamak için kullanılabilir.

Bir örnek verelim:

```
bytes memory payload = abi.encodeWithSignature("register(string)",
"MyName");
(bool success, bytes memory returnData) = address(nameReg).
call(payload);
require(success);
```

Bütün bu fonksiyonlar düşük seviyeli fonksiyonlardır ve dikkatli kullanılmalıdır. Özellikle, bilinmeyen herhangi bir sözleşme kötü amaçlı olabilir ve kullanmanız durumunda, sözleşmenizi geri döndürebilecek bir sonuca neden olabilirsiniz. Bu nedenle işleminiz geri döndüğünde durum değişkenlerinizdeki değişikliklere hazırlıklı olursunuz. Diğer sözleşmelerle etkileşimin en iyi yolu, sözleşmede bulunan **`nesne(x.f())`** üzerinde bir fonksiyon çağırmaktır.

Verilen **`gazı.gas()`** değiştiricisi ile ayarlamak mümkündür.

```
address(nameReg).call.gas(1000000)(abi.encodeWithSignature("register
(string)", "MyName"));
```

Benzer şekilde, verilen **`Ether`** değeri de kontrol edilebilir.

```
address(nameReg).call.value(1 ether)(abi.encodeWithSignature("regist
er(string)", "Daron"));
```

Son olarak bu değiştiriciler birleştirilebilir, değiştiricilerin sırası önemli değildir.

```
address(nameReg).call.gas(1000000).value(1 ether)(abi.encodeWithSign
ature("register(string)", "Daron"));
```

Benzer şekilde, **`delegatecall`** fonksiyonu da kullanılabilir. Buradaki fark, verilen adresin yalnızca kodunun kullanılması, diğer tüm hususların (*depolama, denge*) mevcut sözleşmeden alınmasıdır. **`Delegatecall`**'ın amacı başka bir sözleşmede saklanan kütüphane kodunu kullanmaktır. Kullanıcı, her iki sözleşmedeki depolama düzeninin de kullanılacak **`delegatecall`** fonksiyonu için uygun olmasını sağlamalıdır.

Bellek dizilerini ayırma: Çalışma zamanına bağlı uzunluktaki bellekte diziler oluşturmak için "**`new`**" anahtar sözcüğünü kullanmanız gerekir. Depolama dizilerinin aksine, bellek dizilerini yeniden boyutlandırmak mümkün değildir (*örneğin ".onth" üyesini atayarak*). İstenilen boyutu önceden hesaplamanız veya yeni bir bellek dizisi yaratmanız ve her ögeyi kopyalamanız gerekir.

```
pragma solidity >=0.4.16 <0.6.0;

contract C {
    function f(uint len) public pure {
        uint[] memory a = new uint[](7);
        bytes memory b = new bytes(len);
```

```
        assert(a.length == 7);
        assert(b.length == len);
        a[6] = 8;
    }
}
```

Dizi Değişmezleri: Bir dizi değişmezi, köşeli parantez "`[...]`" içine alınmış bir veya daha fazla ifadenin virgülle ayrılmış listesi şeklinde görünür. Örneğin `[1, a, f (3)]`. Tüm ögelerin örtük olarak dönüştürülebileceği ortak bir tür olması gerekir. Bu, dizinin temel türüdür. Dizi değişmezleri her zaman **statik boyutlu bellek dizileri**dir.

Aşağıdaki örnekte, `[1, 2, 3]` türü `uint8[3]` bellek türündedir. Bu sabitlerin her birinin tipi `uint8` olduğundan, sonucun bir `[3]` bellek olmasını istiyorsanız ilk ögeyi `uint`'e dönüştürmeniz gerekir.

```
pragma solidity >=0.4.16 <0.6.0;

contract C {
    function f() public pure {
        g([uint(1), 2, 3]);
    }
    function g(uint[3] memory) public pure {
        // …
    }
}
```

Sabit boyutlu bellek dizileri, dinamik boyutlu bellek dizilerine atanamaz. Aşağıdaki gibi bir kod yazmamız ne yazık ki mümkün değildir.

```
pragma solidity >=0.4.0 <0.6.0;

// Bu derlenmeyecek
contract C {
    function f() public {
        // sabit boyutlu dizileri
        // dinamik boyutlu dizilere atayamayız
        uint[] memory x = [uint(1), 3, 4];
    }
}
```

Gelecekte bu kısıtlamanın kaldırılması planlanıyor.

Solidity üzerinde dizilerle çalışmak biraz meşakkatli olabilir. Ancak diziler için barındırdığı organlar, Solidity'yi çok pratikleştiriyor. Gelin, bu organlara yakından bakalım.

length: Her dizinin eleman sayılarını içeren bir `.length` ögesi vardır. Bellek dizilerinin uzunluğu oluşturulduktan sonra sabittir. Dinamik boyutlu dizilerde (*yalnızca depolama için kullanılabilir*), `length` ögesi diziyi yeniden boyutlandırmak için atanabilir.

Geçerli uzunluk dışındaki ögelere erişmek, diziyi otomatik olarak yeniden boyutlandırmaz. Uzunluğun arttırılması, diziye sıfırdan başlatılmış yeni ögeler ekler. Uzunluğu azaltmak, **örtük** bir işlem gerçekleştirir.

push: Dinamik depolama dizileri ve baytlar, dizinin sonuna bir eleman eklemek için kullanabileceğiniz `.push` adlı bir fonksiyona sahiptir. Öge yine sıfırda başlatılır. Fonksiyon yeni uzunluğu döndürür.

pop: Dinamik depolama dizileri ve baytlar (*string değil*), bir elemanı dizinin sonundan çıkarmak için kullanabileceğiniz `.pop` adlı bir fonksiyona sahiptir. Bu aynı zamanda dolaylı olarak kaldırılan öge için `.delete` fonksiyonunu çağırır.

Struct: Yapılarla çalışmak istediğinizde, Solidity yeni bir türü tanımlarken "`struct`" biçiminde atamanıza imkân sunar. Bu şekilde kolayca "**yapı**" oluşturup kullanabilirsiniz. Aşağıdaki örnekte bunu deneyelim.

```
pragma solidity >=0.4.11 <0.6.0;

contract CrowdFunding {
    // İki alanlı yeni bir tür tanımlıyoruz.
    struct Funder {
        address addr;
        uint amount;
    }

    struct Campaign {
        address payable beneficiary;
        uint fundingGoal;
        uint numFunders;
```

```
        uint amount;
        mapping (uint => Funder) funders;
    }

    uint numCampaigns;
    mapping (uint => Campaign) campaigns;

    function newCampaign(address payable beneficiary, uint goal)
public returns (uint campaignID) {
        campaignID = numCampaigns++;
        campaigns[campaignID] = Campaign(beneficiary, goal, 0, 0);
    }

    function contribute(uint campaignID) public payable {
        Campaign storage c = campaigns[campaignID];
        c.funders[c.numFunders++] = Funder({addr: msg.sender,
amount: msg.value});
        c.amount += msg.value;
    }

    function checkGoalReached(uint campaignID) public returns (bool
reached) {
        Campaign storage c = campaigns[campaignID];
        if (c.amount < c.fundingGoal)
            return false;
        uint amount = c.amount;
        c.amount = 0;
        c.beneficiary.transfer(amount);
        return true;
    }
}
```

Örnek kodumuzun başlangıcından da anlayabileceğiniz üzere, kitlesel fonlama alanında işlev görecek bir **akıllı sözleşme** yazdık. Bu örnek sözleşmemiz, kitle fonlamasının ihtiyaç duyduğu tam işlevselliği sağlamıyor. Fakat yapıları anlamak için gereken temel kavramları içeriyor. Yapısal türler, eşleme ve dizilerin içinde kullanılabilir; **eşleme** ve **diziler** içerebilir. Ama bir yapının kendi türünde bir öge içermesi mümkün değildir. Burada bir ayrıcalıklı durum var. Yapının kendisi bir eşleme ögesinin değer türü olabilir veya türünde dinamik olarak boyutlandırılmış bir dizi içerebilir.

Yapının boyutunun sonlu olması gerektiğinden, bu kısıtlama oldukça gereklidir.

*Örneğimizdeki fonksiyonlarda, bir yapı türünün veri konumu **storage** olan yerel bir değişkene nasıl atandığını fark ettiniz mi?* Değişken yapıyı kopyalamaz, ancak yalnızca bir referansı saklar, böylece yerel değişken üyelerine yapılan atamalar aslında duruma yazabilir. Tabii ki `campaigns[campaignID].amount = 0` ifadesindeki gibi, yerel bir değişkene atamadan yapı üyelerine doğrudan erişebilirsiniz.

Eşleme Türleri: Eşleme türlerini syntax ile `mapping(_KeyType => _ValueType)` şeklinde gösterebilirsiniz. `_KeyType` herhangi bir temel tür olabilir.

Bu, yerleşik değer türlerinden herhangi birinin **byte** ve **string** olabileceği anlamına gelir. Sözleşme türleri, numaralandırmalar, eşlemeler, yapılar, bayt ve dizgiler dışında herhangi bir dizi türü gibi kullanıcı tanımlı veya karmaşık türlere izin verilmez. `_ValueType` eşleme dâhil olmak üzere herhangi bir tür olabilir.

Eşleme dediğimiz olguyu, her olası anahtarın var olduğu ve bayt temsilinin tümü sıfır olan bir türün, varsayılan değeri olan bir değere eşlenecek şekilde başlatılan **hash** tabloları olarak düşünebilirsiniz. Benzerlik burada biter, anahtar veriler bir eşlemede depolanmaz, değere bakmak için sadece **keccak256 hash** kullanılır. Bu nedenle, eşlemelerin ayarlanan bir anahtarın ya da değerin gözünde herhangi bir uzunluğu veya kavramı yoktur.

Eşlemeler yalnızca bir veri storage konumuna sahip olabilir ve bu nedenle durum değişkenleri için, fonksiyonlardaki depolama referans türleri veya kütüphane işlevleri parametreler olarak kullanılır. Ancak bunlar, herkese açık durumdaki sözleşme fonksiyonları veya iade parametreleri olarak kullanılamazlar.

Eşleme türündeki değişkenleri genel olarak işaretleyebilirsiniz ve Solidity sizin için bir alıcı oluşturur. `_KeyType`, alıcı için bir parametre hâline gelir. `_ValueType` bir değer türü veya yapı ise alıcı `_ValueType` değerini döndürür. `_ValueType` bir dizi veya eşleme ise alıcıda her `_KeyType` için bir parametre tekrarlanır.

Örneğin, **`_ValueType`** bir eşleme olsun:

```
pragma solidity >=0.4.0 <0.6.0;

contract MappingExample {
    mapping(address => uint) public balances;

    function update(uint newBalance) public {
        balances[msg.sender] = newBalance;
    }
}

contract MappingUser {
    function f() public returns (uint) {
        MappingExample m = new MappingExample();
        m.update(100);
        return m.balances(address(this));
    }
}
```

Silmek: **`delete a`**, türün başlangıç değerini **a** olarak atar. Yani tam sayılar için **a = 0**'a eş değerdir. Ancak aynı zamanda, sıfır değerine sahip dinamik bir diziyi veya tüm değerlerini başlangıç değerlerine ayarlanan aynı uzunluktaki statik diziyi atadığı dizilerde de kullanılabilir. **`delete a[x]`**, dizinin **x** dizinindeki ögeyi siler ve diğer tüm ögeleri ve dizinin uzunluğunu el değmemiş hâlde bırakır. Bu, özellikle dizide bir boşluk bıraktığı anlamına gelir. Ögeleri kaldırmayı planlıyorsanız, **`mapping`** muhtemelen daha iyi bir seçimdir.

Yapılar için **`delete`**, tüm üyelerin sıfırlanmasıyla bir **yapı** atanmasını sağlar. Başka bir deyişle, bir **`delete`** işlemi sonrası a'nın değeri, a ile atama yapılmadan bildirilen ile aynıdır.

`Delete`, eşlemeler üzerinde hiçbir etkiye sahip değildir (*eşlemelerin anahtarları isteğe bağlı olabileceği ve genellikle bilinmediği için*). Bu nedenle, bir yapıyı silerseniz eşleme olmayan tüm üyeleri sıfırlar ve eşleme olmadıkça üyelere tekrar girer. Ancak, ayrı ayrı tuşlar ve eşlenecekleri değerler silinebilir. **a** bir eşleme ise **`delete a[x]`** **x**'te depolanan değeri siler.

`delete a` işleminin gerçekten atama gibi davrandığını, yani **a**'da yeni bir nesneyi sakladığını anlamak çok önemlidir. Bu durum, **`a`** referans değişken olduğunda da görülebilir. Bu noktada önceden bahsettiği değeri değil, sadece kendini sıfırlar.

```
Pragma solidity >=0.4.0 <0.6.0;

 124ublic124t DeleteExample {
    uint data;
    uint[] dataArray;

    function f()  124ublic {
        uint x = data;
        delete x; // x'i 0'a atar, verileri etkilemez
        delete data; // veriyi 0'a ayarlar, X'i etkilemez
        uint[] storage y = dataArray;
        delete dataArray;
        Assert(y.length == 0);
    }
}
```

Buraya kadar temel yapı hakkında az çok fikir sahibi olduğunuzu düşünüyorum. Akıllı sözleşmeler kodlarken birimler ve global olarak mevcut değişkenler üzerine de yoğunlaşacaksınız. Bu noktada Ethereum ağının birimleri ve değişkenleri üzerine de birkaç satır yazmam gerekiyor.

Ether Birimleri: Bir sabit sayı, Ether'in alt sınıfını ifade etmek için bir **`wei`**, **`finney`**, **`szabo`** veya **`Ether`** eki alabilir; bir postfix olmadan kullanılan Ether sayılarının ön tanımlı olarak **Wei** olduğu varsayılır.

```
Assert(1 wei == 1);
assert(1 szabo == 1e12);
assert(1 finney == 1e15);
assert(1 ether == 1e18);
```

Alt sınıf eki aslında, onluk bir gücün çarpanı işlevini görür.

Zaman Birimleri: Gerçek sayılardan sonraki **`seconds`**, **`minutes`**, **`hours`**, **`days`** ve **`weeks`** gibi son ekler, saniyelerin temel kabul edildiği bir düzende zaman birimlerini belirlemek için kullanılabilir.

Tabii bu, birtakım dönüşümlerle mümkün olur. Bu dönüşümleri yeri geldiğinde hesap makinesiyle yapmayı deneyebilirsiniz. Şahsen birkaç kez bu duruma düşmüştüm. Sonradan elimin altında bulunması için bir çizelge hazırladım. Gerçekten çok pratik oluyor. Şimdi sizlerle o çizelgeyi paylaşmak istiyorum.

```
1 == 1 saniye
1 dakika == 60 saniye
1 saat == 60 dakika
1 gün == 24 saat
1 hafta == 7 gün
```

Bu birimleri kullanarak takvim hesaplamaları yaparken dikkat edin, çünkü **her yıl 365 güne eşit değildir**. Yine aynı sebeple, her gün tam olarak 24 saat de değildir. Artık saniyelerin tahmin edilememesi nedeniyle, kesin bir takvim kütüphanesinin haricî bir kâhin tarafından güncellenmesi gerekir.

Bu son ekler değişkenlere uygulanamaz. Mesela, bir fonksiyon parametresini gün cinsinden yorumlamak istiyorsanız, aşağıdaki şekilde yapabilirsiniz.

```
function f(uint start, uint daysAfter)  ublic {
    if (now >= start + daysAfter * 1 days) {
      // ...
    }
}
```

Blok ve İşlem Özellikleri: Akıllı sözleşmeler geliştirirken blok işlemleri üzerinde döngüler kuracağınız zamanlar da olacak. Bu gibi durumlarda kullanabileceğimiz birçok işlem bulunuyor.

`blockhash(uint blockNumber) returns (bytes32)`: Verilen bloğun hash değeri, geçerli bloklar hariç yalnızca en yeni 256 adet blok için çalışır.

`block.coinbase (payable address)`: Mevcut blok madenci adresi.

`block.difficulty (uint)`: Geçerli blok zorluk değeri.

`block.gaslimit (uint)`: Geçerli blok bilgisi.

`block.number (uint)`: Geçerli blok numarası.

`block.timestamp (uint)`: UNIX döneminden bu yana saniye olarak tanımlanan, geçerli blok zaman damgası.

`gasleft () returns (uint256)`: Kalan gaz miktarı.

`msg.data (byte calldata)`: Tamamlanmış call verisi.

`msg.sender (payable address)`: Mesajın göndereni (*mevcut çağrı*).

`msg.sig (bytes4)`: Calldata'nın ilk dört baytı (*yani fonksiyon tanımlayıcısı*).

`msg.value (uint)`: Mesaj ile gönderilen wei sayısı.

`now (uint)`: Geçerli blok zaman damgası (*block.timestamp için diğer ad*).

`tx.gasprice (uint)`: İşlemin gaz fiyatı.

`tx.origin (payable address)`: İşlemin göndereni (*tam çağrı zinciri*).

ABI Kodlama ve Kod Çözme Fonksiyonları:

`abi.decode(bytes memory encodedData, (...)) returns (...)`: ABI, verilen verilerin kodunu çözerken türleri parantez içinde ikinci argüman olarak verir. Mesela,

```
(uint a, uint[2] memory b, bytes memory c) = abi.decode(data, (uint,
uint[2], bytes))
```

`abi.encode(...) returns (bytes memory)`: ABI verilen argümanları kodlar.

`abi.encodePacked(...) returns (bytes memory)`:

Verilen bağımsız değişkenlerin paketlenmiş kodlamasını gerçekleştirir. Paketlenmiş kodlamanın belirsiz olabileceğini unutmayın!

`abi.encodeWithSelector(bytes4 selector, ...) returns (bytes memory)`: ABI, ikinci değerden başlayarak verilen argümanları kodlar ve verilen dört bayt seçiciyi hazırlar.

`abi.encodeWithSignature(string memory signature, ...) returns (bytes memory)`: **`abi.encodeWithSelector`** ile eş değer bir kullanıma sahiptir.

Örnek vermek gerekirse;

```
(abi.encodeWithSelector(bytes4(keccak256(bytes(signature))), ...)
```

Bu kodlama fonksiyonları, haricî bir fonksiyon çağırmadan haricî fonksiyon çağrıları için veri oluşturmak amacıyla kullanılabilir.
Dahası, `keccak256` (`abi.encodePacked (a, b)`), yapılandırılmış verinin hash değerini hesaplamanın bir yoludur.

Adres Tür Ögeleri: Akıllı sözleşmeler geliştirirken adresler belki de en çok kullanacağınız ögeler arasında olacak. Adres türlerine önceki bölümlerde kısa bir giriş yapmıştım. Şimdi kullanabileceğiniz etiketlere kısaca değinelim ki örnek kodlara geçtiğimizde adreslerin kullanımını daha net anlayın.

`<adres> .balance (uint256)`: Wei biriminde adres bakiyesi.

`<address payable>.transfer(uint256 amount)`: Verilen miktarda Wei'yi adrese gönderir, başarısızlık durumunda gönderilen miktar geri döner.

`<address payable>.send(uint256 amount) returns (bool)`: Verilen miktarda Wei'yi adrese gönderir, başarısızlık durumunda `false` verir.

`<address>.call(bytes memory) returns (bool, bytes memory)`: Verilen yük ile düşük seviye `call` düzenler, başarı durumunu ve iade verilerini geri gönderir.

`<address>.delegatecall(bytes memory) returns (bool, bytes memory)`: Verilen yük ile düşük seviye `delegatecall` düzenler, başarı durumu ve iade verilerini iletir.

`<address>.staticcall(bytes memory) returns (bool, bytes memory)`: Verilen yük ile düşük seviyeli `staticcall` düzenler, başarı durumu ve iade verilerini iletir, mevcut tüm gazları yönlendirir.

Adres etiketleriyle çalışırken ve başka sözleşmelerle etkileşime girerken mümkün olduğunca `.call()` kullanmaktan kaçınmalısınız. Çünkü `.call()` yönergesi tür kontrolünü, fonksiyon varlığı kontrolünü ve argüman paketlemesini kontrol etmeden işlem yaptığı için uygulamanızda beklenmeyen sıkıntılara yol açabilir.

Benzer şekilde `.send` kullanımında da dikkatli olmak gerekir. Çağrı yığını derinliği 1024 sınırına ulaşmış ise aktarım başarısız olur veya alıcı gazın bitmesi durumunda da başarısız olur. Bu nedenle, güvenli Ether transferlerini yapmak için, her zaman gönderinin geri dönüş değerini kontrol etmeniz gerekir. Daha garantili bir iş yapmak adına `.transfer()` yönergesini kullanmanızı öneririm.

Sözleşme yönergeleri: Sözleşmelerde kullanılan tanımlamalar uygulamamızın işleyişi için oldukça önemlidir. Özellikle şartların oluşmadığı zamanlarda sözleşmeye konu olan tutarların iadesi söz konusuysa bu yönergeleri kullanmak durumdayız.

`this (geçerli sözleşmenin türü)`: Açıkça adres'e çevrilebilir olan mevcut sözleşme.

`selfdestruct(address payable alıcısı)`: Mevcut sözleşmeyi iptal etmek ve/veya fonlarını verilen adrese göndermek.

Ayrıca, mevcut sözleşmenin tüm fonksiyonları doğrudan geçerli fonksiyon dâhil olmak üzere çağrılabilir.

Sözleşme tip bilgisi: Sözleşmeleri tiplere ayırarak çalışmak, zaman zaman pratik olabilir. Ancak sözleşmeye tip atarken Ethereum ağının kullandığınız yordama destek verip vermediğine dikkat etmelisiniz.

`Type(X)`, X türü hakkında bilgi almak için kullanılabilir. Şu anda, bu özellik için sınırlı destek var, ancak gelecekte genişletilebilir. C tipi bir sözleşme için aşağıdaki özellikler mevcuttur:

`type(C).creationCode`: Sözleşmenin oluşturma kodunu içeren bellek bayt dizisidir. Bu, satır içi derlemede, özellikle `create2` operasyon kodu kullanılarak özel oluşturma yordamları için kullanılabilir. Bu özelliğe, sözleşmenin kendisi üzerinden veya sözleşmeden türetilmiş herhangi bir başka sözleşme içerisinden erişilemez. Çünkü, bu tür işlemler bayt kodunun çağrı sitesinin bayt koduna dâhil edilmesine neden olarak döngüsel bir yapı oluşturur. Bu tip dairesel referanslar sözleşmeler içerisinde mümkün değildir.

`type(C).runtimeCode`: Sözleşmenin çalışma zamanı kodunu içeren bellek bayt dizisidir. Bu, genellikle C tipinin kurucusu tarafından dağıtılan bir koddur. C, satır içi derleme kullanan bir kurucuya sahipse bu gerçekte dağıtılan bayt kodundan farklı olabilir. Ayrıca, kitaplıkların düzenli aramalara karşı korunmak için konuşlandırma sırasında çalışma zamanı bayt kodunu değiştirdiğini unutmayın. Bu özellik için `.creationCode` ile aynı kısıtlamalar geçerlidir.

Son Söz: Bu bölümde, örnek olarak paylaşacağım akıllı sözleşmeleri daha iyi anlayabilmeniz için **Solidity** diline kısa bir giriş yaptım.

Ancak bir dili birkaç sayfaya sığdırmamız mümkün değil. Kitabın asıl odağının Solidity dili olmadığını da hesaba katarsak bu kadarla yetinmemiz gerekecek.

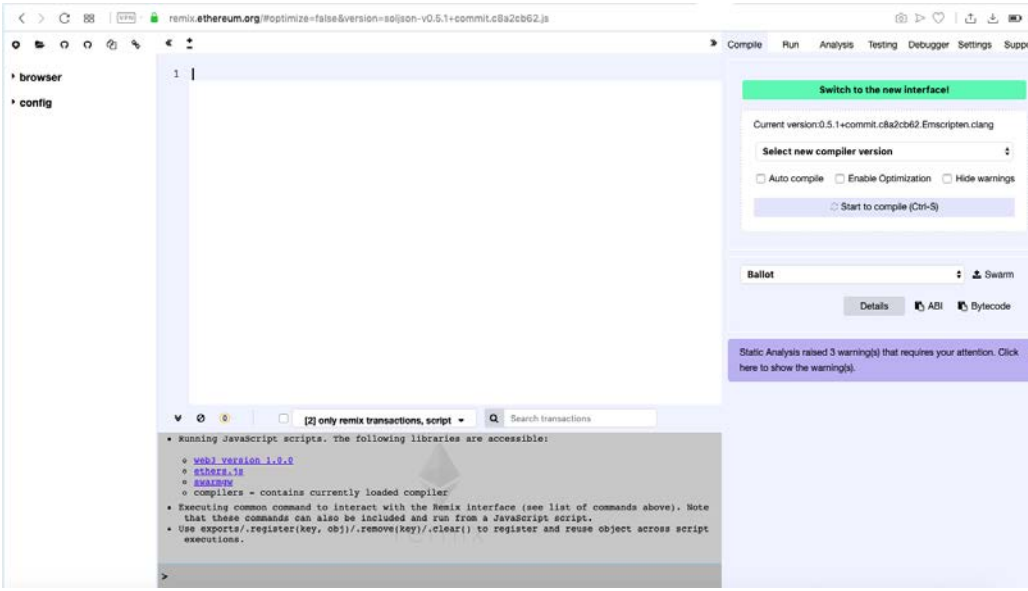
Bu bölümü bitirince, en azından Solidity dilinin temel işlev operatörlerini, geliştirme ortamlarının kurulumunu ve basit bir sözleşmenin nasıl kodlanacağını deneyimlemiş oldunuz.

Daha fazlası için bir sonraki bölümde paylaşacağım kapsamlı örnekleri inceleyerek kendinizi geliştirebilirsiniz.

İLK AKILLI SÖZLEŞMEMİZ

Bu bölümde artık sözleşme yazmaya başlıyoruz. Ben Remix'i kullanacağım. Siz de Remix ya da Solc kullanabilirsiniz. Remix'e, *remix.ethereum.org* adresinden ulaşabileceğinizi hatırlatmak istiyorum.

Remix'i açtığınızda, sizi **Şekil 1.106**'daki gibi bir ekran karşılayacak.

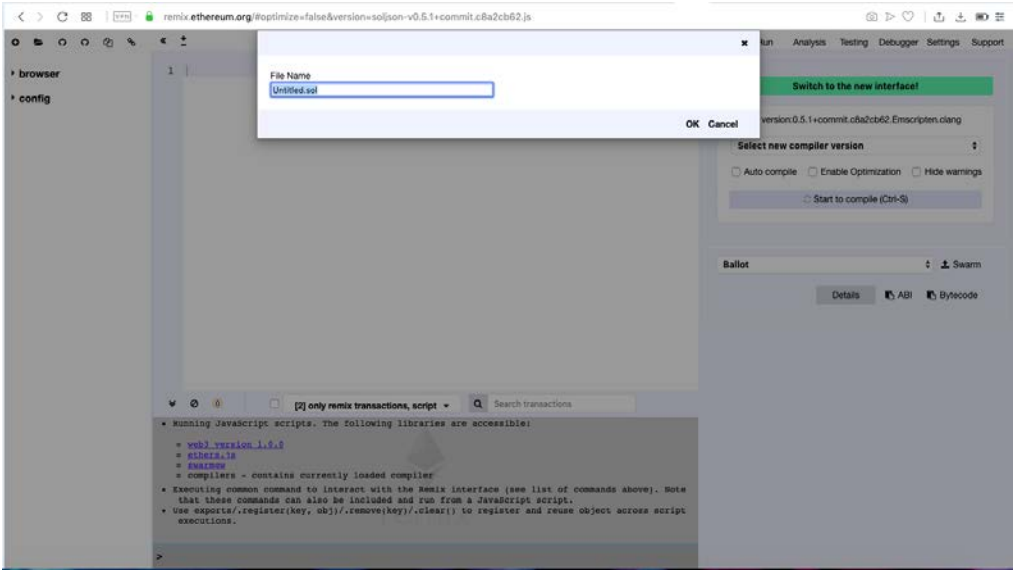


Şekil 1.106: Remix IDE'nin açılışı

Bu ekranın sol üst köşesinde yer alan menü başlıklarından "**+**" işaretiyle betimlenmiş olan "**yeni**" düğmesine basarak projemizi başlatıyoruz.

Bu düğmeye bastığımızda bizi isim kutucuğu karşılayacak. Hemen ilk uygulamamızın adını verelim. Dosyamı **AçıkArtırma1** olarak adlandırıyorum. Siz dilediğiniz ismi verebilirsiniz. Uzantının **.sol** olmasına dikkat edin.

Solidity kaynak kod dosyalarının uzantısı **.sol** olarak belirlenmiş. Dolayısıyla Remix'e .sol uzantılı dosyaları yükleyebileceğiniz gibi, yazdığınız kodları da .sol uzantılı olarak indirebilirsiniz.



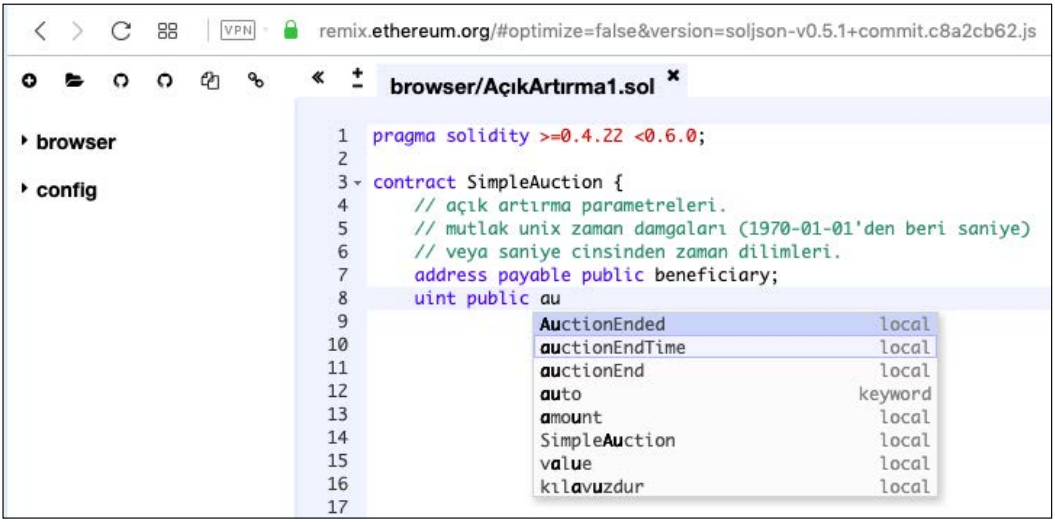
Şekil 1.107: Sol üst köşesinde yer alan menü başlıklarından "+" işaretiyle betimlenmiş olan "yeni" düğmesine basarak projemizi başlatıyoruz.

İlk yazdığımız kod satırımız `pragma` olacak. Pragmanın ne işe yaradığını önceki bölümlerde açıklamıştık. Burada derleyici çekirdeğimizi **0.4.22** ve **0.6.0** sürümleri arasından herhangi bir sürüm olarak belirleyeceğim. Siz bunu sabitleyebileceğiniz gibi, farklı koşullar da tanımlayabilirsiniz.

Detayları hatırlamıyorsanız `Pragma` kullanımı anlattığımız **HIZLANDIRILMIŞ SOLIDITY** bölümüne göz atabilirsiniz.

Remix ile kod yazmak oldukça pratiktir. Normalde grafiksel ortamda Visual Studio, Xcode ve Brackets; terminal ortamında ise Vim ve Nano kullanmaya alışığım. Ancak Remix ile kod yazmak da alışık olduğum ortamları aratmıyor.

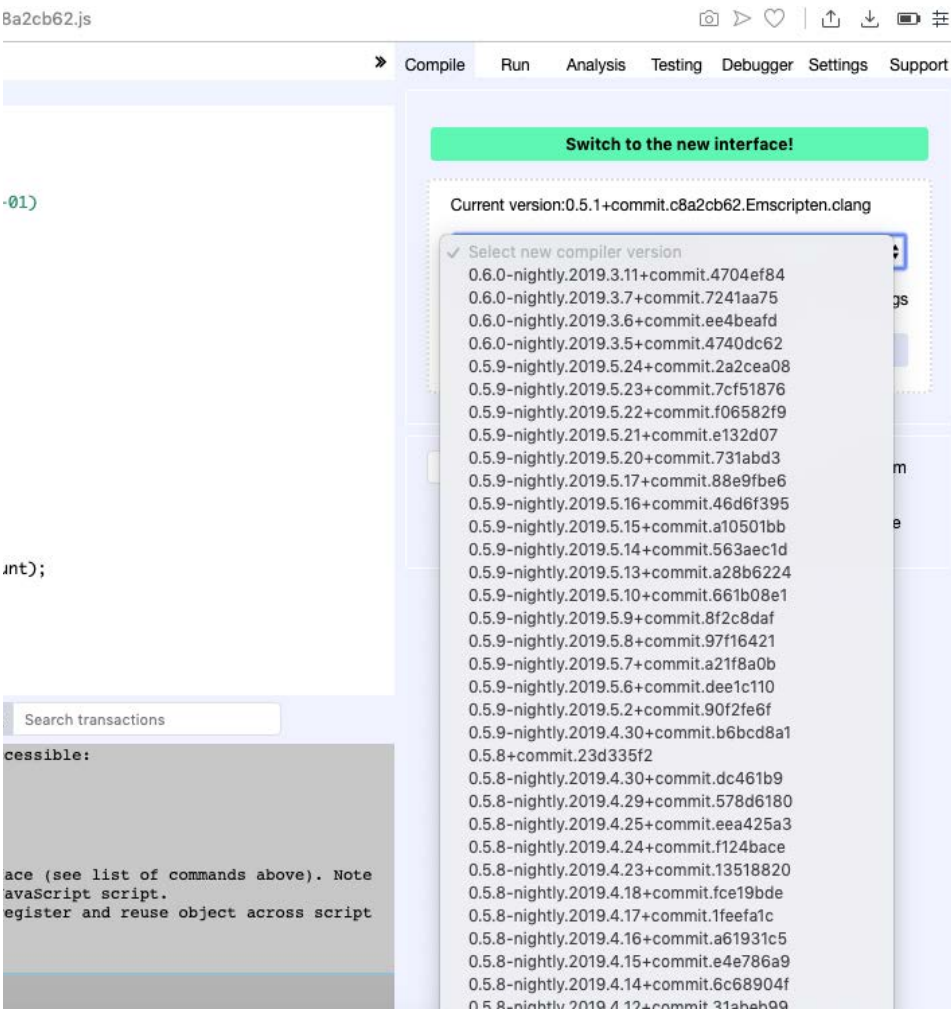
Kod renklendirme ya da otomatik tamamlama gibi temel IDE özelliklerinin hepsini barındırıyor. Üstelik, debug özelliği oldukça gelişmiş. "**Compile**" başlığı altında derleme ayarları ve olası uyarılar çok iyi konumlandırılmış.



Şekil 1.108: Remix birçok IDE'de bulunan gelişmiş özelliğe sahip.

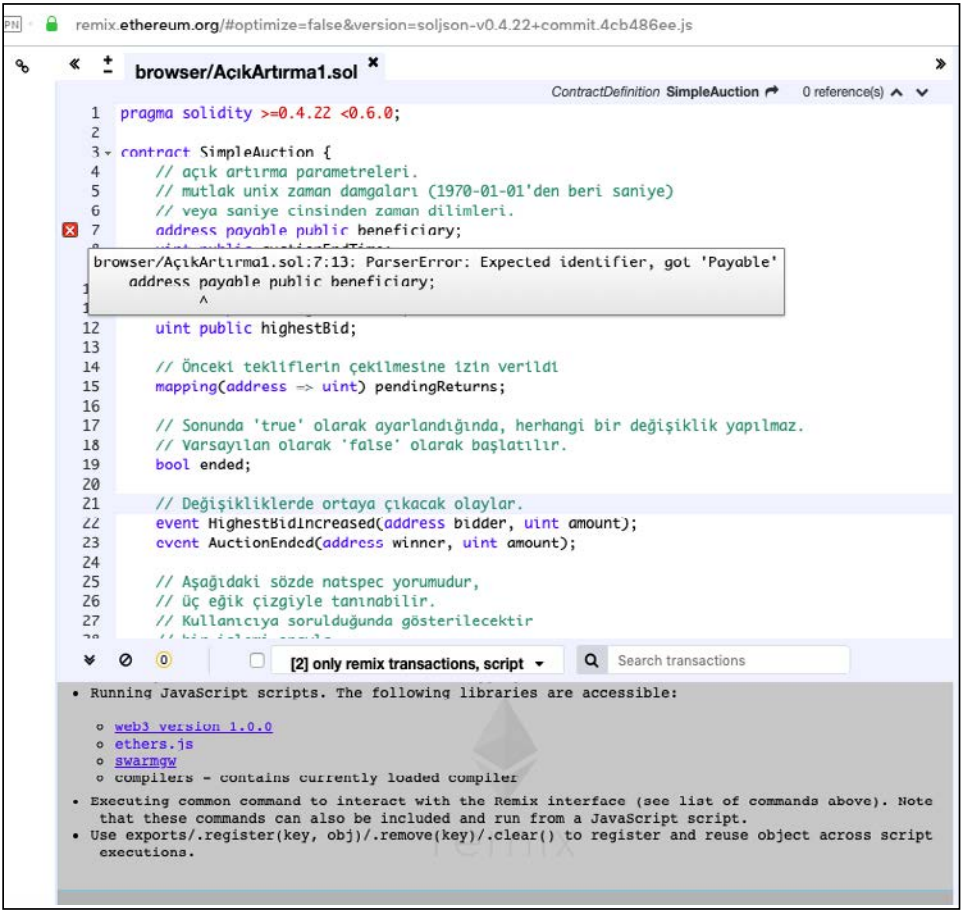
Remix, derleme, çalıştırma ve debug gibi özellikleri sağ üst köşede sekmeler hâlinde kullanıma sunuyor. Bu, kodunuzdaki olası sıkıntıları tespit etmenizde ve gidermenizde ergonomik bir kullanım kolaylığı sunuyor.

Kod yazma işlemi bittikten sonra derleme yapmak için "**compiler**" sürümü seçmeniz gerekiyor. Benzer şekilde birden fazla kaynak kod dosyasıyla çalışıyorsanız "**compiler**" bölümünün altında dosya adı seçmeniz gerekiyor. Yoksa Remix hâlihazırda açık olan kod sayfanızı derlemeye çalışacaktır.



Şekil 1.109: Derleme işlemi öncesinde dilediğiniz bir "compiler" sürümü seçebilirsiniz.

Remix ile çalışırken bazen çakışmalardan dolayı anlık hatalar almanız mümkün. Öyle bir durumda derleyici sürümünü kontrol etmenizi öneririm. Çünkü bazen bir özellik yeni bir sürümde desteklenmemeye başlayabiliyor. Fakat derleyici listesinden eski bir sürümü seçmeniz hâlinde kodunuz çalışacaktır.



Şekil 1.110: Remix ile çalışırken bazen çakışmalardan dolayı anlık hatalar almanız mümkün.

Yalnız, kullandığınız bir özelliğin ya da yönergenin artık yeni sürümde desteklenmiyor olması sizi güvenlik ve olası açıklar konusunda biraz düşündürmeli. Eğer ciddi güvenlik açıklarına sebep olan bir özelliği kullanmaya diretiyorsanız belki alternatif bir yöntem seçmeli ya da farklı bir fonksiyon kurmayı denemelisiniz. Sözleşmenizi kullanacak olanları riske atmamalısınız.

Evet, beklenen an geldi. Şimdi basit bir akıllı sözleşme yazarak okuduklarımızı pekiştirelim. Yazacağımız sözleşme "**merhaba dünya**" geleneğini yaşatmak için hazırladığım bir örnek. Anlayacağınız üzere Solidity ile Ethereum ağı üzerinden tüm dünyaya "**merhaba**" diyeceğiz.

```
pragma solidity ^0.4.22;

contract MerhabaDunya {
    // bu bir akıllı sözleşmedir
}

function birseyYaz() {
    // yapılacak iş
}

return "merhaba dünya";

function birseyYaz() returns (string) {

}
```

Şimdi, işi biraz büyütelim ve işe yarar uygulamalar yapalım. Basit ve gizli olmayan türde açık artırma sözleşmesi kodlayacağız.

Açık artırma yapılarına hepimiz aşinayız. **Ebay** ile hayatımıza giren açık artırma tabanlı ikinci el satış fikri, ülkemizde de zamanında popüler olmuş ve yaygınlaşmıştı. Ancak son zamanlarda kripto para ve akıllı sözleşmeler sayesinde bu iş modeli çok daha ileri bir aşamaya yükseldi. Artık çok daha güvenli bir yapı altında ve çok daha basit bir şekilde artırma usulünde satış düzenlenebiliyor. İş dünyasının vazgeçilmezi ihaleler de bu sayede **e-ihale** formuna kolayca dönüştürülebiliyor. **Artırma modelli online satış**ın genel fikri, herkesin ihale sürecinde açık olarak teklif verebilmesi için bir altyapı ya da platform sağlayabilmektir. Teklif veren kullanıcılar, tekliflerinde vaat ettikleri Ether miktarı ile ihaleyi kazanmaya çalışırlar. En yüksek teklifin verilmesi durumunda, bir önceki en yüksek teklif bozulmuş olur. İhale süresinin bitmesinin ardından, geçerli en yüksek teklifi veren alıcı ve malı satan kullanıcı sözleşmeyi manuel olarak onaylayarak ürün veya servis alışverişini tamamlarlar. Sözleşme bu temel şart gerçekleşmeden aktif hâle gelemez, yürürlüğe girmez.

Bu mantığı koda dökecek olursak;

```
pragma solidity >=0.4.22 <0.6.0;

contract SimpleAuction {
    // açık artırma parametreleri.
    // mutlak unix zaman damgaları (1970-01-01'den beri saniye)
    // veya saniye cinsinden zaman dilimleri.
    address payable public beneficiary;
    uint public auctionEndTime;

    // Açık artırmanın şu anki durumu.
    address public highestBidder;
    uint public highestBid;

    // Önceki tekliflerin çekilmesine izin verildi
    mapping(address => uint) pendingReturns;

    // Sonunda 'true' olarak ayarlandığında, herhangi bir değişiklik yapılmaz.
    // Varsayılan olarak 'false' olarak başlatılır.
    bool ended;

    // Değişikliklerde ortaya çıkacak olaylar.
    event HighestBidIncreased(address bidder, uint amount);
    event AuctionEnded(address winner, uint amount);

    // Aşağıdaki sözde natspec yorumudur,
    // üç eğik çizgiyle tanınabilir.
    // Kullanıcıya sorulduğunda gösterilecektir
    // bir işlemi onayla.

    /// _biddingTime ile basit bir açık artırma oluşturun
    constructor(
        uint _biddingTime,
        address payable _beneficiary
    ) public {
        beneficiary = _beneficiary;
        auctionEndTime = now + _biddingTime;
    }

     /// Bu işlemle birlikte gönderilen değer ile
     /// açık artırma için teklif verin.
```

```
 /// Değer, yalnızca açık artırma
 /// kazanılmadığında iade edilir.
function bid() public payable {
    // Bağımsız değişken gerekli değildir,
    // tüm bilgiler zaten işlemin bir parçasıdır.
    // Fonksiyonun Ether kodunu alabilmesi
    // için anahtar kelime ödemesi gerekir.

    // Eğer teklif varsa, aramayı geri al
    // dönem bitti.
    require(
        now <= auctionEndTime,
        "Açık artırma sona erdi."
    );

    // Teklif yüksek değilse,
    // para iade edilir.
    require(
        msg.value > highestBid,
        "Zaten daha yüksek bir teklif var."
    );

    if (highestBid != 0) {
        // Basitçe latestBidder.send (latestBid)
        // kullanarak parayı geri gönderme
        // bir güvenlik riskidir
        // çünkü sözleşmede bir açık yaratabilir.
        // Alıcılara withdraw izni vermek
        // her zaman daha güvenlidir
        // paralarını kendileri geri çeker.
        pendingReturns[highestBidder] += highestBid;
    }
    highestBidder = msg.sender;
    highestBid = msg.value;
    emit HighestBidIncreased(msg.sender, msg.value);
}
/// Geçersiz olan bir teklifi geri çekiyoruz.
function withdraw() public returns (bool) {
    uint amount = pendingReturns[msg.sender];
    if (amount > 0) {
        // Bunu sıfıra ayarlamak önemlidir,
        // çünkü alıcı 'send' sonucu dönmeden önce
```

```
            // çağrının bir parçası olarak bu işlevi tekrar 'call' edebilir.
            pendingReturns[msg.sender] = 0;

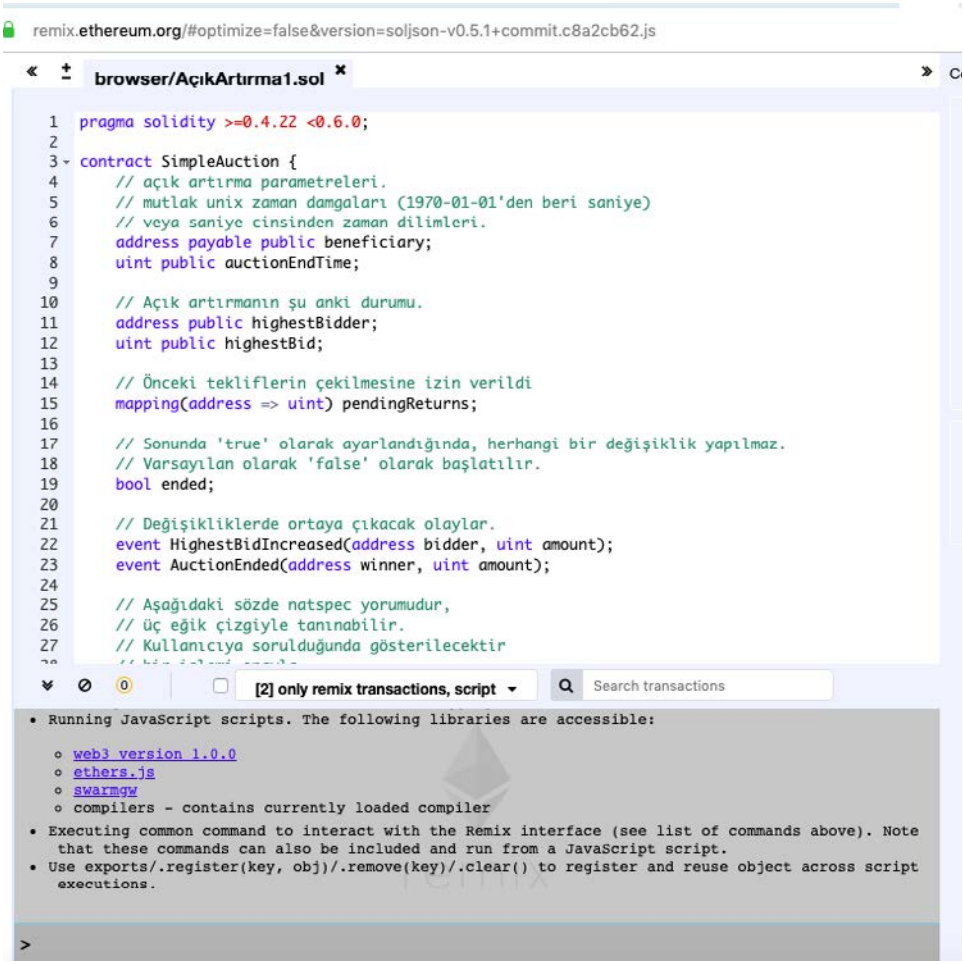
            if (!msg.sender.send(amount)) {
                // Burada 'call' kullanmaya gerek yok,
                // sadece borçlu olduğunuz miktarı sıfırlayın yeter.
                pendingReturns[msg.sender] = amount;
                return false;
            }
        }
        return true;
    }

    /// Açık artırmayı sonlandırın ve en yüksek teklifi gönderin
    function auctionEnd() public {
        // Etkileşen fonksiyonları yapılandırmak için iyi bir kılavuzdur
        // diğer sözleşmelerle (yani, işlevleri çağırır veya Ether gönderir)
        // üç aşamada:
        // 1. kontrol koşulları
        // 2. eylemleri gerçekleştirme (potansiyel olarak değişen koşullar)
        // 3. diğer sözleşmelerle etkileşime girmek
        // Bu aşamalar karışıksa, diğer sözleşme mevcut sözleşmeye geri dönebilir
        // ve durumu değiştirebilir ya da etkilerin (eter ödemesi)
        // birden fazla kez yapılmasına neden olabilir.
        // Dahili olarak adlandırılan fonksiyonlar harici sözleşmelerle
        // etkileşimi içeriyorsa, harici sözleşmelerle etkileşimin de dikkate
        // alınması gerekir.

        // 1. Koşullar
        require(now >= auctionEndTime, "Açık artırma henüz
bitmedi.");
        require(!ended, "auctionEnd fonksiyonu çağrıldı.");

        // 2. Etkiler
        ended = true;
        emit AuctionEnded(highestBidder, highestBid);

        // 3. Etkileşim
        beneficiary.transfer(highestBid);
    }
}
```



Şekil 1.111: Kodlamanızı tamamladığınızda ilk sözleşmenizi tamamlamış olacaksınız.

Gördüğünüz gibi aslen bir açık artırma yapısı kurgulamak **Solidity** ile oldukça kolaydır. Ancak bazen açık artırma yerine kapalı usulde teklif verilmesi esasıyla artırma düzenlenebilir. Hatta negatif artırma yöntemi de kurgulanabilir. Kapalı ihale mantığında karşımıza çıkabilecek en büyük sorun, teklif veren için sözleşmenin bağlayıcı ve gizli hâle nasıl getirileceğidir. Teklif verenin, açık artırmayı kazandıktan sonra para göndermemesini önlemenin tek yolu, parayı peşin olarak ödemesini sağlamaktır ki bunun en kolay yolu bir **akıllı sözleşme** hazırlamaktan geçer.

Yalnız unutmamanız gereken önemli bir nokta var; Ethereum'da para transferleri adres bazında anonim olabilirken meblağ bazında **gizlilik yoktur**. Yani herkes gönderilen ödemelerin meblağlarını görecektir.

Şimdi gelin önceki sözleşmemizde birkaç değişiklik yapalım.

```
pragma solidity >0.4.23 <0.6.0;

contract BlindAuction {
    struct Bid {
        bytes32 blindedBid;
        uint deposit;
    }

    address payable public beneficiary;
    uint public biddingEnd;
    uint public revealEnd;
    bool public ended;

    mapping(address => Bid[]) public bids;

    address public highestBidder;
    uint public highestBid;

    // Önceki teklif sahiplerinin paralarını çekmesine izin verildi
    mapping(address => uint) pendingReturns;

    event AuctionEnded(address winner, uint highestBid);

    /// Değiştiriciler fonksiyonlara yapılan girdileri
    /// doğrulamak için uygun bir yoldur.
    /// 'onlyBefore' yöntemi aşağıdaki teklife uygulanır:
    /// Yeni işlev gövdesi, değiştiricinin gövdesidir;
    /// burada '_' işlevi, eski işlev gövdesi tarafından değiştirilir
    modifier onlyBefore(uint _time) { require(now < _time); _; }
    modifier onlyAfter(uint _time) { require(now > _time); _; }

    constructor(
        uint _biddingTime,
        uint _revealTime,
        address payable _beneficiary
    ) public {
```

```
        beneficiary = _beneficiary;
        biddingEnd = now + _biddingTime;
        revealEnd = biddingEnd + _revealTime;
    }
```

/// '_BlindedBid' = keccak256(abi.encodePacked(value, fake,
/// secret)) ile kör bir teklif verin. Gönderilen Ether, yalnızca /// teklif ifşa aşamasında doğru bir şekilde bildirilirse iade /// edilir. Teklifle birlikte gönderilen Ether en azından "value" /// ve "fake" ise, teklif geçerlidir. "fake"
/// değerini "true" olarak ayarlamak ve kesin tutarı göndermek, /// gerçek teklifi gizlemenin bir yoludur ancak yine de gerekli /// para yatırma işleminin yapılmasının yoludur. Aynı adres
/// birden fazla teklif verebilir.

```
    function bid(bytes32 _blindedBid)
        public
        payable
        onlyBefore(biddingEnd)
    {
        bids[msg.sender].push(Bid({
            blindedBid: _blindedBid,
            deposit: msg.value
        }));
    }
```

/// Kör tekliflerinizi ortaya çıkarın.
/// Doğru şekilde körlenmiş tüm geçersiz teklifler ve tamamen
/// yüksek olanlar hariç tüm teklifler için geri ödeme alırsınız.

```
    function reveal(
        uint[] memory _values,
        bool[] memory _fake,
        bytes32[] memory _secret
    )
        public
        onlyAfter(biddingEnd)
        onlyBefore(revealEnd)
    {
        uint length = bids[msg.sender].length;
        require(_values.length == length);
        require(_fake.length == length);
        require(_secret.length == length);

        uint refund;
        for (uint i = 0; i < length; i++) {
```

```
            Bid storage bidToCheck = bids[msg.sender][i];
            (uint value, bool fake, bytes32 secret) =
                    (_values[i], _fake[i], _secret[i]);
            if (bidToCheck.blindedBid != keccak256(abi.
encodePacked(value, fake, secret))) {
                // Teklif aslında açıklanmadı.
                // Depozito iadesi yapmayın.
                continue;
            }
            refund += bidToCheck.deposit;
            if (!fake && bidToCheck.deposit >= value) {
                if (placeBid(msg.sender, value))
                    refund -= value;
            }
            // Gönderenin aynı teminatı yeniden talep
            // etmesini imkansız hale getirin.
     bidToCheck.blindedBid = bytes32(0);
        }
        msg.sender.transfer(refund);
    }

    // Bu "internal" bir işlevdir.
    // Yani yalnızca sözleşmenin kendisinden
    // (veya türetilmiş sözleşmelerden) çağrılabileceği
    // anlamına gelir.

    function placeBid(address bidder, uint value) internal
            returns (bool success)
    {
        if (value <= highestBid) {
            return false;
        }
        if (highestBidder != address(0)) {
            // Bir önceki en yüksek teklifi veren kişiye geri ödeme.
            pendingReturns[highestBidder] += highestBid;
        }
        highestBid = value;
        highestBidder = bidder;
        return true;
    }
    /// Geçersiz olan bir teklifi geri çekin.
    function withdraw() public {
```

```
        uint amount = pendingReturns[msg.sender];
        if (amount > 0) {
            pendingReturns[msg.sender] = 0;

            msg.sender.transfer(amount);
        }
    }

    /// Açık artırmayı sonlandırın ve
    /// en yüksek teklifi satıcıya gönderin.
function auctionEnd()
        public
        onlyAfter(revealEnd)
    {
        require(!ended);
        emit AuctionEnded(highestBidder, highestBid);
        ended = true;
        beneficiary.transfer(highestBid);
    }
}
```

Açık veya kapalı usulde artırma düzenlemekle satıcı için **en iyi fiyat** koşulunu yaratmış oluyoruz. Üstelik ödemenin tahsil edilmesini garanti altına alarak en yüksek fiyatı veren alıcı adayından teklifi kadar Ether miktarını blokaj altına alıyoruz.

Buraya kadar her şey güzel ama ya satış işlemi? Hadi gelin ona da bakalım.

```
pragma solidity >=0.4.22 <0.6.0;

contract Purchase {
    uint public value;
    address payable public seller;
    address payable public buyer;
    enum State { Created, Locked, Inactive }
    State public state;

    // 'msg.value' değerinin çift olduğundan emin olun.
    // Bölüm tek bir sayı ise kesilecektir.
    // Çarpma ile tek sayı olup olmadığını kontrol edin
    constructor() public payable {
        seller = msg.sender;
```

```
        value = msg.value / 2;
        require((2 * value) == msg.value, "Değer eşit olmalı.");
    }

    modifier condition(bool _condition) {
        require(_condition);
        _;
    }

    modifier onlyBuyer() {
        require(
            msg.sender == buyer,
            "Sadece alıcı bunu çağırabilir."
        );
        _;
    }

    modifier onlySeller() {
        require(
            msg.sender == seller,
            "Sadece satıcı bunu çağırabilir."
        );
        _;
    }

    modifier inState(State _state) {
        require(
            state == _state,
            "Geçersiz durum."
        );
        _;
    }

    event Aborted();
    event PurchaseConfirmed();
    event ItemReceived();

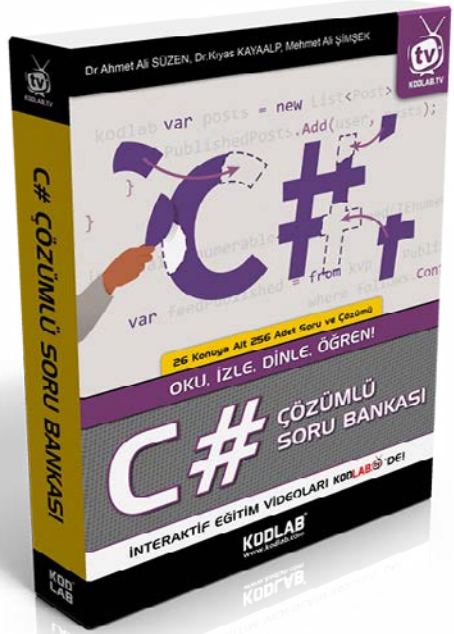
    /// Satın almayı iptal et ve Ether'i geri al.
    /// Sözleşme kilitlenmeden önce
    /// satıcı tarafından çağrılabilir.
    function abort()
```

```
        public
        onlySeller
        inState(State.Created)
    {
        emit Aborted();
        state = State.Inactive;
        seller.transfer(address(this).balance);
    }

    /// Satın almayı alıcı olarak onaylayın.
    /// Transfer işlemi '2 * value' Ether içermelidir.
    /// Onaylanana kadar Ether blokaja alınacak
    /// diyoruz.
    function confirmPurchase()
        public
        inState(State.Created)
        condition(msg.value == (2 * value))
        payable
    {
        emit PurchaseConfirmed();
        buyer = msg.sender;
        state = State.Locked;
    }

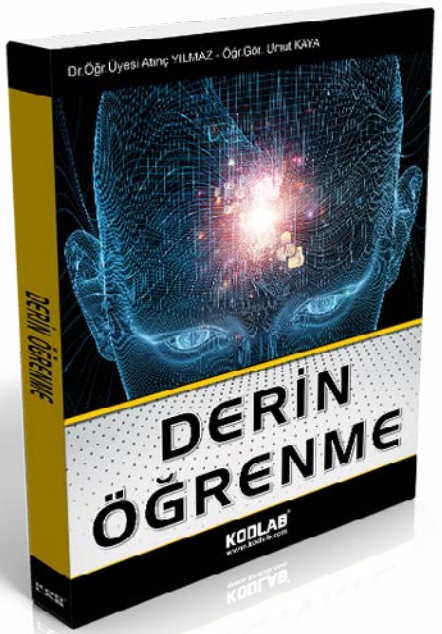
    /// Siz (alıcı) ögeyi aldığınızı onaylayın.
    /// Bu blokajdaki Ether'i serbest bırakacak.
    function confirmReceived()
        public
        onlyBuyer
        inState(State.Locked)
    {
        emit ItemReceived();
        // Önce durumu değiştirmemiz önemli. Çünkü,
        // aksi takdirde, aşağıda 'send' kullanılarak çağrılan
        // sözleşmeler buradan tekrar çağrılabilir.
        state = State.Inactive;
        buyer.transfer(value);
        seller.transfer(address(this).balance);
    }
}
```

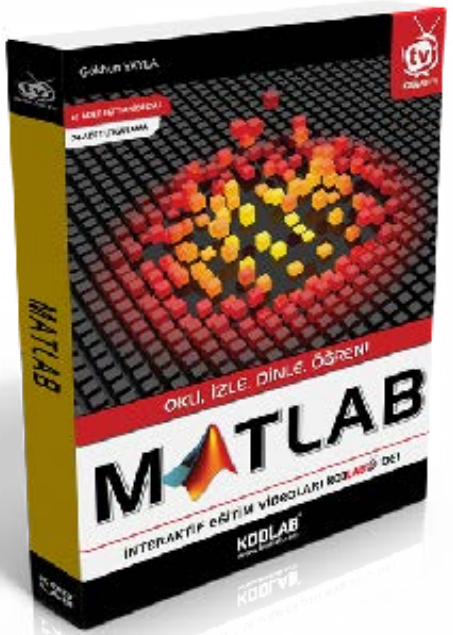

Emrah YÜKSEL
SIFIRDAN İLERİ SEVİYEYE
PHP
WEB PROGRAMLAMA
HTML + CSS + PDO + MySql
OKU. İZLE. DİNLE. ÖĞREN!
SIFIRDAN İLERİ SEVİYEYE
PHP
WEB PROGRAMLAMA
İNTERAKTİF EĞİTİM VİDEOLARI KODLAB'DE!
KODLAB
www.kodlab.com


Dr Ahmet Ali SÜZEN, Dr.Kıyas KAYAALP, Mehmet Ali ŞİMŞEK
C# ÇÖZÜMLÜ SORU BANKASI
26 Konuya Ait 256 Adet Soru ve Çözümü
OKU. İZLE. DİNLE. ÖĞREN!
C# ÇÖZÜMLÜ SORU BANKASI
İNTERAKTİF EĞİTİM VİDEOLARI KODLAB'DE!
KODLAB
www.kodlab.com


İlker ERTUĞRUL
GNU LINUX
OKU. İZLE. DİNLE. ÖĞREN!
GNU LINUX
İNTERAKTİF EĞİTİM VİDEOLARI KODLAB'DE!
KODLAB


Gökhan SU
HERKES İÇİN ARDUINO
OKU. İZLE. DİNLE. ÖĞREN!
HERKES İÇİN
ARDUINO
İNTERAKTİF EĞİTİM VİDEOLARI KODLAB'DE!
KODLAB


Dr.Öğr.Üyesi Atınç YILMAZ - Öğr.Gör. Ümut KAYA
DERİN ÖĞRENME
DERİN ÖĞRENME
KODLAB


OKU, İZLE, DİNLE, ÖĞREN!
MATLAB
İNTERAKTİF EĞİTİM VİDEOLARI
KODLAB
MATLAB