پردیس علوم
دانشکده ریاضی، آمار و علوم کامپیوتر

مروری بر بلاکچین و طراحی NFT

نگارنده: حامد مروی
استادان راهنما: دکتر هادی فراهانی، دکتر مرتضی محمد نوری

پایان‌نامه برای دریافت درجه کارشناسی
در رشته علوم کامپیوتر

زمستان ۱۴۰۱

چکیده

در این پژوهش پس از معرفی سیستم‌های توزیع‌شده به مرور مفاهیمی ابتدایی برای درک بلاکچین می‌پردازیم. بلاکچین یک دفترکل غیرمتمرکز و توزیع شده برای ذخیره اطلاعات در ساختمان داده‌ای شبیه به لیست پیوندی است. هر عضو این لیست پیوندی را یک بلاک می‌گوییم که به هش بلاک قبلی متصل است. در ادامه به بیان عملکرد شبکه بلاکچین و برخی مزایا و معایب آن می‌پردازیم. سپس مروری بر مفاهیم مورد نیاز در رمزنگاری و بیان برخی کاربردهای آن در بلاکچین خواهیم پرداخت. در ادامه به معرفی مفهوم اجماع و الگوریتم معروف PoW می‌پردازیم. سپس بلاکچین بیتکوین و پس از آن بلاکچین اتریوم که بستری برای قراردادهای هوشمند فراهم می‌کند را بررسی می‌کنیم و به‌عنوان نمونه قرارداد هوشمندی برای طراحی یک لاتاری با استفاده از مفهوم اراکل ارایه می‌دهیم. در پایان مفهوم توکن را توضیح می‌دهیم و پس از معرفی استانداردهای طراحی توکن، یک توکن با استاندارد ERC20 **و همچنین یک توکن NFT با استاندارد** ERC721 **طراحی می‌کنیم.**

سپاس گزاری

خداوند را سپاس‌گزارم که به من این توفیق را عطا کرد تا بتوانم قدمی در راه علم و دانش بردارم و به گردآوری اطلاعات مربوط به این تحقیق بپردازم.
از استاد راهنمای عزیزم، دکتر فراهانی سپاس‌گزارم که من را با بلاکچین آشنا کرده و الفبای آن را به من آموختند و با تقدیر و سپاس از دکتر نوری.
از خانواده عزیزم سپاس‌گزارم که همواره من را در راه رسیدن به اهدافم همیاری کرده‌اند.

پیش‌گفتار

مطالعه و تحقیق روی فناوری بلاکچین از اوایل دهه نود آغاز شد. از آن به‌عنوان ساختمان داده‌ای جهت ذخیره اسناد و تراکنش‌های مالی استفاده می‌شود. بلاکچین پایه‌ای را برای اعتماد دیجیتال فراهم کرده که کاربران می‌توانند بدون نیاز به اعتماد به یک واسط مرکزی با هم تبادل اطلاعات داشته باشند. در سال ۱۹۹۱ استوارت هابر[۱] مقاله‌ای باعنوان "چگونه یک سند دیجیتال را زمان‌بندی کنیم" منتشر کرد. این مقاله تغییر ناپذیری رکوردهای دیجیتال را با استفاده از سرویس مهر زمانی[۲] که از توابع هش و امضای دیجیتال برای تایید معتبر بودن سند استفاده می‌شود، بررسی می‌کند. در این سیستم اسناد مانند یک لیست پیوندی به هم متصل شده اند تا یک توالی زمانی برای تایید هر سند ایجاد کنند. این زنجیره ساده‌ترین نسخه بلاکچین است. سپس در سال ۱۹۹۸ نیک سابو[۳] شروع به‌کار برروی یک ارز دیجیتالِ غیرمتمرکز به‌نام **BitGold** کرد. او مقاله‌ای منتشر کرد که در آن یک زنجیره از هشِ اسناد با مهر زمانی بود که مساله‌ای به نام "دوبار خرج کردن" را تشخیص می‌داد اما بدون استفاده از سیستمی متمرکز نمی‌توانست مانع از رخداد آن شود. بیت‌گلد عملی نشد زیرا نیک سابو معتقد بود باید سیستمی غیرمتمرکز درست شود که مشکل دوبار خرج کردن حل گردد. همچنین در بیت‌گلد مفهوم ارزش سکه‌ها (یا کوین‌ها) براساس هزینه منبع محاسباتی برای استخراج آن معرفی شد. مشکل این موضوع این است که اگر ارزش مستقیما با هزینه محاسباتی مرتبط باشد، ممکن است ارزش یک سکه از سکه‌ی مشابه دیگر بالاتر باشد.

سرانجام بلاکچین و رمزارز بیتکوین توسط شخص ناشناسی با نام مستعار ساتوشی ناکاموتو[۴] در سال ۲۰۰۸ معرفی شد. ساتوشی از مطالعات قبلی در زمینه‌های دفتر کل[۵] توزیع‌شده، سیستم‌های غیرمتمرکز و توابع رمزنگاری و هش کردن، بیتکوین را پیاده‌سازی کرد. هدف این کار ساختن یک دفتر معاملات برای رمزارز بیتکوین بود. معرفی بلاکچین بیتکوین آن را به اولین ارز دیجیتالی تبدیل کرد که مساله دوبار خرج کردن را بدون نیاز به شخص سومِ قابل اعتماد (یک واسط) رفع کرد. ارزش سکه‌های بیتکوین توسط بازار آزاد قیمت‌گذاری می‌شوند، در غیر این‌صورت بیتکوین‌هایی که در سال ۲۰۱۰ استخراج (ماین) شدند، ارزششان از بیتکوین‌های امروزه کمتر می‌بود زیرا هزینه محاسباتی کمتری نیاز داشتند. بلاکچین بیتکوین یک شبکه همتا-به-همتا است که تراکنش‌ها را با زمان اجرایشان با هم ترکیب می‌کند و در یک بلاک قرار می‌دهد. برای این‌کار نیازی به تایید یک نهاد مرکزی نیست و توسط الگوریتم اجماع[۶] که تحت کنترل هیچ فردی نیست انجام می‌شود.

در سال ۲۰۱۳ ویتالیک بوترین مقاله‌ای با این ایده مطرح کرد که بیت‌کوین به یک زبان برنامه‌نویسی جهت ساخت اپلیکیشن‌های غیرمتمرکز نیاز دارد. این ایده مورد موافقت سایر اعضای انجمن بیتکوین واقع نشد پس خودش یک شبکه محاسباتی توزیع‌شده بر مبنای بلاکچین با نام اتریوم را ارایه نمود. در این محیط برنامه‌نویسان می‌توانند برنامه‌هایی تحت عنوان قرارداد هوشمند ایجاد

---

[۱] Stuart Haber
[۲] time-stamp
[۳] Nick Szabo
[۴] Satoshi Nakamoto
[۵] ledger
[۶] consensus algorithm

کنند که یکی از کاربردهای آن ایجاد یک تراکنش مالی درصورت برآورده شدن دنباله‌ای از شروط است. اتریوم این اجازه را می‌دهد تا برنامه‌نویسان، اپلیکیشن‌های غیرمتمرکز یا DApp[۷] را روی بستر بلاکچین بنویسند و منتشر کنند.
در این گزارش ابتدا به بررسی مفاهیم ابتدایی مورد نیاز برای درک بلاکچین می‌پردازیم و سپس مروری مختصر برروی دو بلاکچین بیتکوین و اتریوم خواهیم داشت و در پایان با معرفی استانداردهای ساخت توکن و NFT یک توکن با استاندارد ERC721 ارایه می‌کنیم.

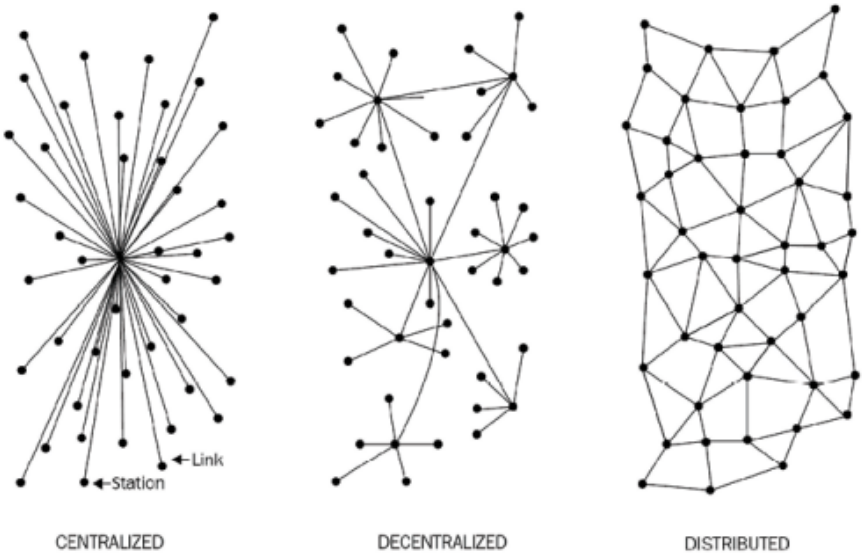
[۷] Decentralized Application

فهرست مطالب

فصل ۱

سیستم‌های غیرمتمرکز و توزیع‌شده

از آن‌جا که بلاکچین سیستمی توزیع‌شده و غیرمتمرکز است، در این فصل به توضیح این دو مفهوم می‌پردازیم.
سسیستم‌ها به سه گروه متمرکز[۱] ، غیرمتمرکز[۲] و توزیع‌شده[۳]. دسته بندی می‌شوند که در شکل زیر قابل مشاهده است.



در سیستم‌های متمرکز یک نهاد مرکزی نظارت سیستم را برعهده دارد و تصمیم‌گیرنده نهایی است. تمام کاربران سیستم متمرکز بر این نهاد وابسته هستند. اکثر سرویس‌های آنلاین از این

centralized[۱]
decentralized[۲]
distributed[۳]

مدل استفاده می‌کنند.

سیستم‌های غیر متمرکز: غیرمتمرکز بودن یعنی به‌جای یک نهادِ مرکزی، اختیارات را بین چند واحد مختلف تقسیم کنیم. این تغییر برای سازمان‌ها فوایدی مانند افزایش کارایی، تسریع تصمیم‌گیری و کاهش حجم بار روی مدیر مرکزی می‌شود. همان‌طور که در شکل بالا می‌توان دید در سیستم غیرمتمرکز چند گروه داریم و هر گروه رهبر خودش را دارد. این کار باعث می‌شود یک نهاد واحد، تمام قدرت را در اختیار نداشته باشد.

بلاکچین سیستمی است که نیاز به هیچ واسطی ندارد و می‌تواند چند رهبر داشته باشد که توسط مکانیزم اجماع انتخاب می‌شوند. این مدل اجازه می‌دهد هر فردی سعی کند به‌عنوان تصمیم‌گیرنده اصلی انتخاب شود. مکانیزم اجماع این فرایند انتخاب رهبر را امکان‌پذیر می‌کند. معروف تعریف روش برای این کار اثبات کار[۴] نام دارد که در فصل‌های بعد به آن می‌پردازیم.

سیستم‌های توزیع‌شده : درک سیستم‌های توزیع شده [۵] برای درک مفهوم بلاکچین ضروری است زیرا بلاکچین اساسا یک سیستم توزیع شده است. به‌طور دقیق‌تر، یک سیستم توزیع شدهِ غیر متمرکز [۶] است.

سیستم توزیع شده یا شبکه توزیع شده یک الگو محاسباتی است که در آن دو یا چند گره[۷] (یا سیستم) از طریق یک شبکه با هم در ارتباط اند و برای انجام یک هدف مشترک تلاش می‌کنند و در انتها به یک جواب نهایی می‌رسند.

در تعریف بالا دو نکته مشهود است. اول این‌که ما چند گره (کامپیوتر) داریم که به هم متصل هستند و دوم این‌که این گره‌ها از دید کاربر به‌صورت یک دستگاه دیده می‌شوند. به‌عنوان مثال موتور جست‌وجوگرِ گوگل نوعی شبکه توزیع شده است که چند سیستم متفاوت به دنبال پیدا کردن جوابی برای کاربر هستند و در انتها کاربر یک جواب می‌بیند.

در تعریف بالا به گره اشاره شد. گره یک دستگاهِ یکتا در این سیستم توزیع شده است. تمام گره‌ها قابلیت فرستادن و گرفتن پیام از هم‌دیگر را دارند. پیامِ یک گره می‌تواند درست باشد، اشکال داشته باشد یا از قصد پیامِ اشتباه بفرستد. گره‌ای که اشکال داشته باشد را گره بیزانسی[۸] می‌نامیم.

---

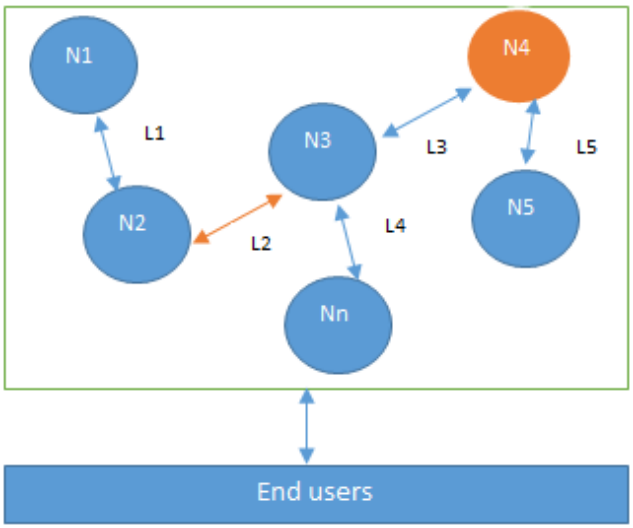
[۴] Proof of Work - Pow

[۵] distributed system

[۶] decentralized

[۷] node

[۸] Byzantine node



در شکل بالا مثالی از یک شبکه توزیع شده را می‌بینیم و در این شبکه گره N4 یک گره بیزانسی است و همچنین رابطِ L2 مشکل دارد یا سرعتش کم است. با افزایش روز افزون حجم داده‌ها و عدم توانایی پاسخِ سیستم‌های پردازشی معمولی پاسخ‌های مختلفی مطرح شد. استفاده از سیستم‌های پردازشی به مشخصات سخت‌افزاری قوی‌تر نیز جایگزین مناسبی نبود زیرا هزینه‌های تامین و نگه‌داری از این منبع گران‌قیمت بود. در راستای حل این مشکل از سیستم‌های توزیع شده که مجموعه‌ای از همان سیستم‌های معمولی است استفاده شد.
طراحی و پیاده‌سازی سیستم‌های توزیغ شده پیچیده است اما قضیه CAP [۹] معیار مناسبی را برای این منظور ارائه داده.

۱.۱ قضیه CAP

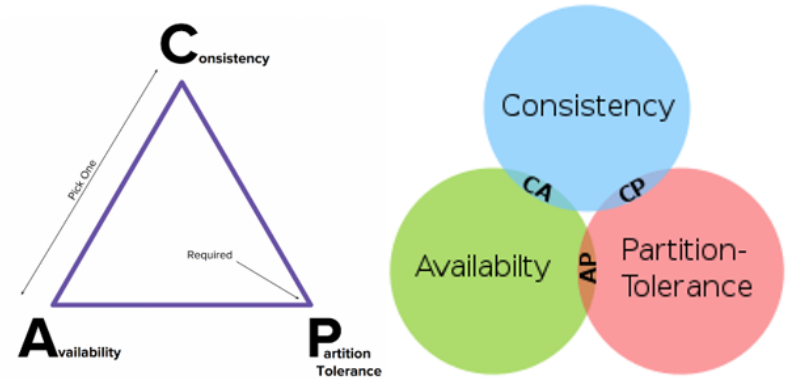
چالش اصلی در شبکه‌های توزیع شده هماهنگی بین گره‌ها و همچنین میزان تحمل و رفع خطای شبکه است. یعنی حتی اگر تعدادی از گره‌ها یا لینک‌های بین گره‌ها (تا حد معینی) مشکل داشته باشند، سیستم باید همچنان بتواند به نتیجه مورد نظر برسد.
در قضیه سی‌ای‌پی ثابت شده که یک سیستم توزیع شده نمی‌تواند همزمان سه ویژگی زیر را داشته باشد:

- سازگاری[۱۰]: سازگاری ویژگی‌ای است که اطمینان می‌دهد همه گره‌ها در یک سیستم توزیع‌شده در هر زمان یک کپی واحد و یکسان از داده‌ها را دارند. برای این‌که این اتفاق بیفتد، باید داده‌هایی که روی یک گره قرار می‌گیرند، به سرعت روی تمام گره‌های دیگر در سیستم نیز اجرا شوند.

[۹] CAP theorem
[۱۰] consistency

- در دسترس بودن[۱۱]: یک گره سالم موجود در سیستم نتیجه درخواست را به کاربر برگرداند. به‌عبارت دیگر، داده‌ها در هر گره در دسترس هستند و گره‌ها به درخواست‌ها پاسخ می‌دهند و هر گره پاسخ یکسانی را برمیگرداند.
- تحمل تقسیم کار[۱۲]: این ویژگی اطمینان می‌دهد اگر گروهی از گره‌ها به‌دلیل خرابی شبکه قادر به برقراری ارتباط با گره‌های دیگر نباشند، شبکه می‌تواند همچنان به درستی به کار خود ادامه دهد.

در سیستم‌های توزیع شده داشتن ویژگی تقسیم کار اجباری است و از بین دو ویژگی دیگر باید یکی را انتخاب کرد.



[۱۱] availability
[۱۲] partition tolerance

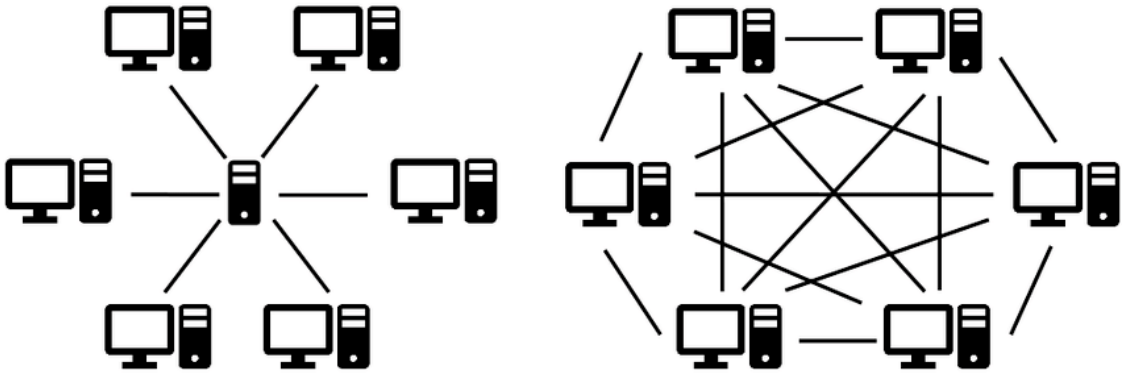
فصل ۲

بلاکچین

در این فصل ابتدا مفاهیم ابتدایی مورد نیاز فراگیری مفهوم بلاکچین را توضیح می‌دهیم و سپس مفهوم بلاکچین را توضیح می‌دهیم.

۱.۲ همتا به همتا

در یک سیستم همتابه‌همتا[۱] هیچ مدیر مرکزی‌ای وجود ندارد و تمام سیستم‌ها می‌توانند بدون واسطه با هم در ارتباط باشند. مقابل این شبکه مدلِ سرویس‌گیرنده - سرویس دهنده [۲] است که یک سیستم مرکزی کارها را بین سیستم‌های دیگر توزیع می‌کند و همچنین رابطی برای ارتباط بین سیستم‌ها است. این ویژگی این امکان را می‌دهد تا هر دو گره بتوانند مستقیما با هم تراکنش داشته باشند و نیازی به وجود رابطِ سوم مانند بانک نیست.



Client-Server network

P2P network

[۱] peer to peer
[۲] client - server model

۲.۲ اجماع

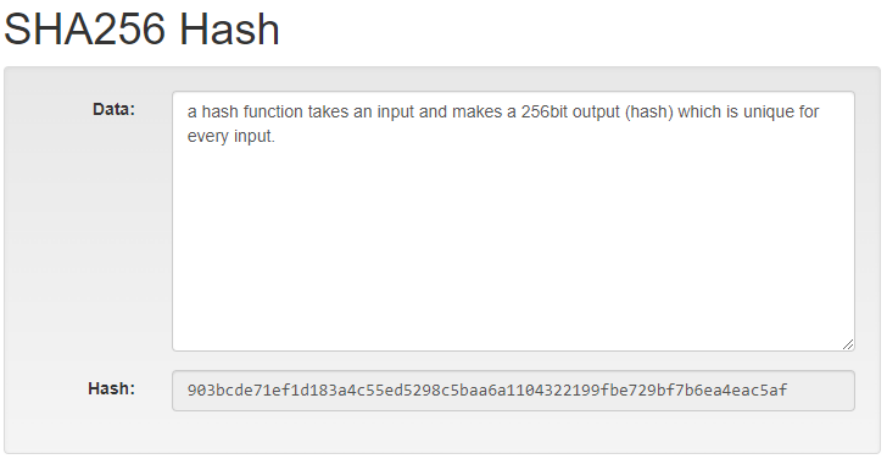
مشکلی که در شبکه‌های توزیع‌شده وجود دارد، دستیابی به قابلیت اطمینان کلی به سیستم درصورت وجود تعدادی فرایند معیوب است. یعنی اگر تمام سیستم‌های شبکه با هم هماهنگ نبودند، باز هم بتوان به خروجی قابلِ اطمینانی دست یافت. برای این هدف الگوریتم‌هایی با نام الگوریتم اجماع[۳] وجود دارند.

۳.۲ دفتر کل توزیع شده

دفتر کل توزیع شده[۴] اجتماعی از داده‌های دیجیتالی مختلف است که در چندین سیستم مختلف، در جاهای مختلف ذخیره شده است. دقت کنید که همه این سیستم‌ها کپی‌ای از یک داده دارند و همه یک چیز را ذخیره کرده‌اند. در این سیستم‌ها هیچ مدیر مرکزی‌ای وجود ندارد. یک دفتر کل توزیع شده نیاز به شبکه کامپیوتر همتابه‌همتا و الگوریتم‌های اجماع دارد. یکی از فرم‌های دفتر کل توزیع شده تکنولوژی بلاکچین است.

۴.۲ مفهوم هش

تابع هش[۵] الگوریتمی است که داده‌هایی را به‌عنوان ورودی می‌گیرد و رشته‌ای شامل ۰ و ۱ را برمی‌گرداند. خروجی این تابع همیشه از اندازه‌ای مشخص است.
یکی از این الگوریتم‌ها SHA256 نام دارد که داده را می‌گیرد و رشته‌ای ۲۵۶ بیتی از ۰ و ۱ برمی‌گرداند. در مثال زیر رشته ۲۵۶ بیتی در مبنای ۱۶ نمایش داده شده:



---

consensus algorithm[۳]
distributed ledger[۴]
hash[۵]

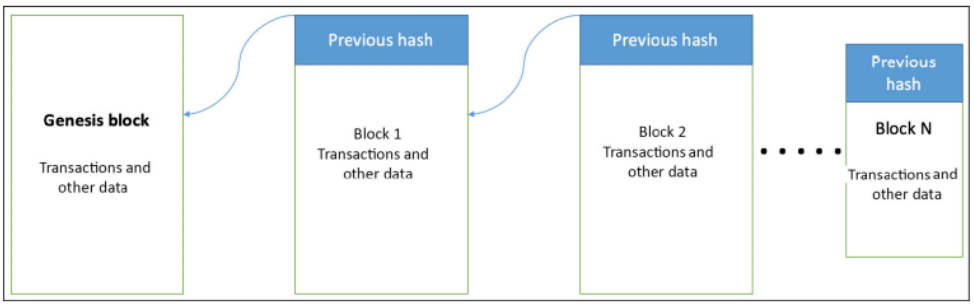
به‌طور مختصر مفاهیم بالا را برای ارایه تعریفی از بلاکچین توضیح دادیم. در فصل‌های بعد برخی موارد را به‌طور مفصل‌تر بررسی خواهیم کرد.

۵.۲ تعریف بلاکچین

برای بلاکچین دو تعریف ارایه می‌دهیم:
تعریف لیمن[۶] : بلاکچین سیستمی در حال رشد، ایمن و مشترک بین کاربران و برای ذخیره اطلاعات است به‌طوری که هر کاربر یک کپی از داده واحدی را در اختیار دارند و داده‌های ذخیره شده در سیستم تنها در حالتی بروز رسانی می‌شوند که تمام گره‌هایی که در تراکنش جاری نقش دارند با این امر موافقت کنند.
تعریف تکنیکال: بلاکچین یک دفتر کل توزیع شده همتا ـ به ـ همتا ، رمزنگاری شده [۷] و غیر قابل تغییر است که فقط بزرگ‌تر می‌شود و اضافه کردن داده به آن تنها از طریق توافق جمعی [۸] امکان پذیر است.

همان‌طور که از اسمش پیداست، بک سیستم بلاکچین از چند بلاک تشکیل شده که این بلاک‌ها توسط ساختمان داده لیست پیوندی[۹] به هم متصل شده‌اند و به مرور به سیستم اضافه می‌شوند. هر بلاک می‌تواند شامل هر داده‌ای مانند تراکنش‌های بانکی، اسناد مالکیت یا پیام‌های شخصی باشد.

همان‌طور که در شکل زیر مشاهده می‌شود یک بلاک از دو بخش اصلیِ داده‌های بلاک و هشِ بلاک قبلی تشکیل شده است.



مفاهیمی که در یک بلاک می‌بینیم:

- آدرس[۱۰]: آدرس‌ها شناسه‌های منحصر به‌فردی هستند که در تراکنش‌هایی که در شبکه بلاکچین اتفاق میفتند دو طرف قرارداد یعنی فرستنده و گیرنده را مشخص میکنند.

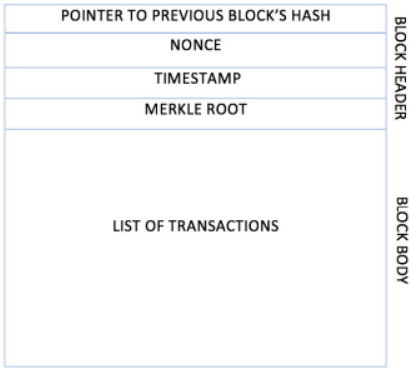
---

[۶] layman
[۷] cryptographically secure
[۸] updatable via consensus
[۹] linked list
[۱۰] address

• تراکنش[۱۱]: تراکنش یک واحد اصلی از بلاکچین است. تراکنش نمایانگر انتقال مبلغی از آدرسِ فرستنده به آدرسِ گیرنده است.

• هش: هر بلاک دارای هش است. هش مقداری است که توسط یک تابع هش[۱۲] ساخته شده و ورودی‌های این تابع داده‌های موجود در هر بلاک مانند نانس، برچسب زمانی، هشِ بلاک قبلی و تراکنش‌های موجود در بلاک است. پس تمام داده‌های موجود در بلاک ورودی‌های این تابع هش هستند.

• بلاک: یک بلاک از چند تراکنش و همچنین عناصر دیگری مانند هشِ بلاک قبل، برچسب زمانی[۱۳] ، ریشه درخت مرکل[۱۴] و نانس[۱۵] است. بلاک از یک سرتیترِ و تعدادی تراکنش که به‌ترتیبی سازمان دهی شده پشت هم قرار دارند تشکیل می‌شود. هر بلاک را می‌توان به دوقسمت بدنه، که شامل تراکنش‌های موجود در بلاک است و سرتیتر تقسیم کرد.



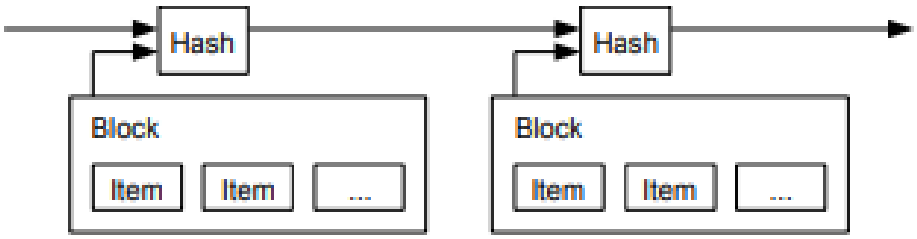
• ریشه درخت مرکل: درخت مرکل[۱۶] یا درخت هش[۱۷] ساختار داده‌ای است که در شبکه‌های بلاکچین استفاده می‌شود. درخت مرکل ساختار داده‌ای شبیه یک درخت وارونه است که می‌تواند حجم زیادی از داده را به‌صورت ایمن پردازش و خلاصه‌سازی کند. به‌عنوان مثال در شکل زیر می‌خواهیم هشت رشته را رمزنگاری کنیم. این رشته‌ها در برگ‌های درخت قرار داده می‌شوند و سپس آن‌ها را دوبه‌دو با یک تابع هش رمزنگاری می‌کنیم تا در نهایت به یک مقدار در ریشه برسیم که در ریشه درختِ مرکل قرار می‌گیرد.

---

[۱۱] transaction
[۱۲] hash function
[۱۳] time stamp
[۱۴] merkle root
[۱۵] nonce
[۱۶] merkle tree
[۱۷] hash tree

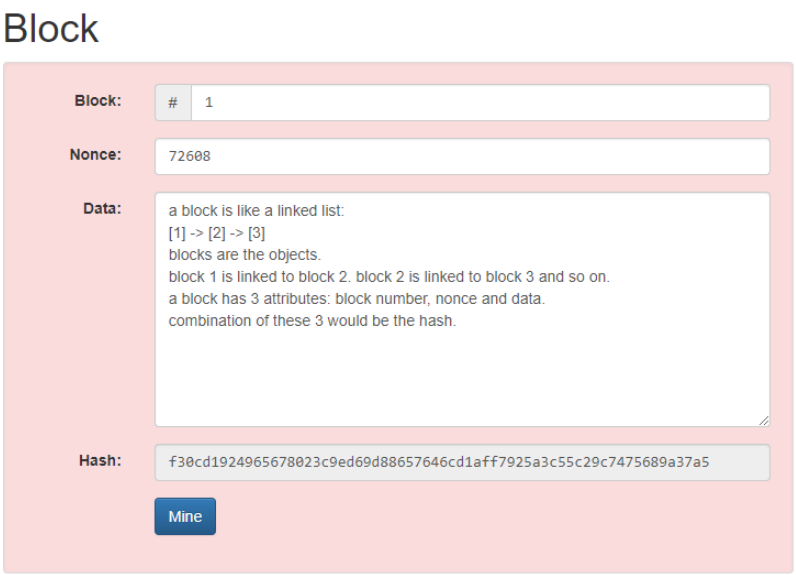
در بلاک‌چین از درخت مرکل برای رمزنگاری تراکنش‌های موجود در یک بلاک استفاده می‌کنیم. یعنی برگ‌های درخت تراکنش‌های موجود در بلاک هستند و ریشه درخت مرکل شامل همه تراکنش‌های در آن بلاک است. به کمک این ساختمان داده اگر بخواهیم تراکنش‌های یک بلاک را تایید کنیم، نیاز نیست این کار را تک تک به ازای هر تراکنش انجام دهیم و می‌توانیم صرفا ریشه درخت مرکل را تایید کنیم. ریشه درخت مرکل در بخش سرتیترِ بلاک قرار دارد.

همان‌طور که گفتیم بلاکچین از چند بلاک تشکیل شده که با لیست پیوندی به هم متصل اند. یکی از ویژگی‌های هر بلاک هشِ مربوط به آن است که خروجی تابع هش‌ای است که ورودی‌هایش داده‌های موجود در آن بلاک مانند عددی به نام نانس، شماره بلاک، تراکنش‌ها، هشِ بلاک قبلی و چیزهای دیگر است. نکته مهم در اینجا این است که برای ساختنِ هش بلاکِ فعلی از هشِ بلاک قبل استفاده می‌شود و دقت شود که فقط در بلاک اولیه[۱۸] از هشِ بلاکِ قبلی استفاده نمی‌شود.

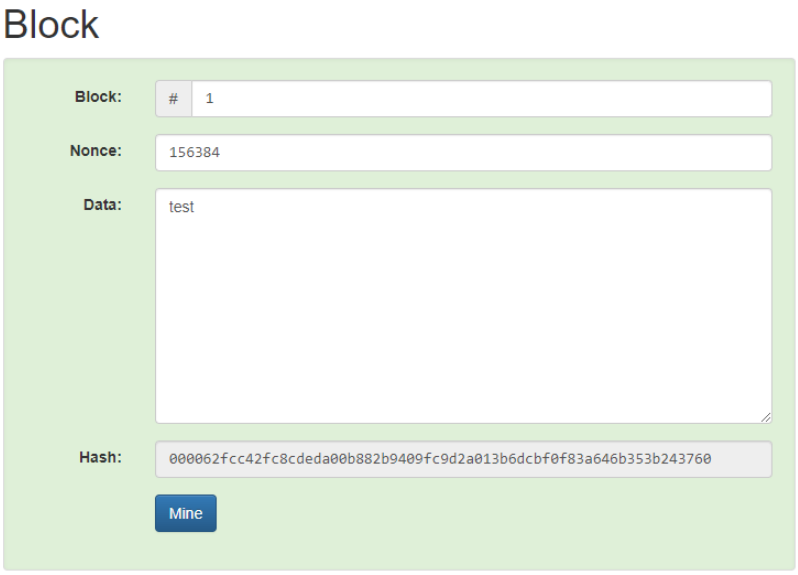


[۱۸] genesis block

- نانس:



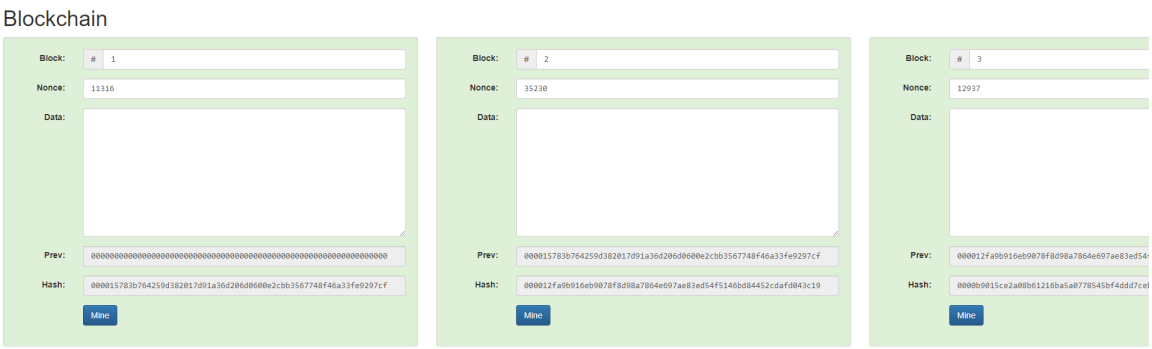
در مثال بالا برای ساخته شدنِ هش سه وردی داریم: شماره بلاک، داده‌ی بلاک و عددِ نانس. دوتای اَولی از قبل تعیین شدَه اند اما عدد نانس مجهول است. وقتی ماینرها می‌خواند بلاک را ماین کنند عملا باید عددِ نانسی را پیدا کنند که وقتی داده‌های بلاک (چیزهایی که از قبل تعیین شده و ثابت اند) و عددِ نانس را به تابعِ هش بدهیم، هش بلاک دارای شرطی باشد، مثلا عدد هش باید با ۴ تا صفر شروع شود. به‌عنوان مثال برای شماره بلاک ۱ و داده زیر عددِ نانس‌ای که شرط را برقرار کند (در این مثال یعنی هشِ خروجی با ۱۶ تا صفر شروع شَود) به صورتِ زیر است:



در مثالِ زیر ساختارِ یک بلاکچین را نشان می‌دهیم. در این مثال هر بلاک ۵ ویژگی دارد:

- شماره بلاک
- عدد نانس
- داده
- هشِ بلاک قبلی
- هشِ که به کمکِ ۴ تای قبلی ساخته می‌شود

که به کمک هش بلاک قبلی میتوان لیست پیوندیِ دوطرفه داشت و برای بلاکِ اول این مقدار برابر با صفر است. هشِ بلاک فعلی به هشِ بلاک قبل ربط دارد پس اگر بخواهیم دادِه یکی از بلاک‌های قبلی را عوض کنیم، به صورت زنجیره وار هشِ تمام بلاک‌های بعد از آن عوض می‌شود و به کمک این ساختار می‌توان گفت داده‌های موجود در یک بلاکچین تغییر ناپذیراند، زیرا وقتی داده یکی از بلاک‌های قبل را عوض کنیم، باید گره‌های دیگر را راضی کنیم که با این تغییر ما همگام باشند وگرنه گره ما گره‌ای مشکل‌زا حساب می‌شود و تغییری که داده‌ایم در شبکه حساب نمی‌شود.



۶.۲ عملکرد شبکه بلاکچین

در بلاکچین کاربرها و سیستم‌های مختلف را گره می‌نامیم. هر گره یا ماینر[۱۹] است که بلاک‌های جدید را درست می‌کند و رمز ارز (سکه یا پولِ مجازی) ضرب می‌کند (مانند ضرب کردن سکه یا پول در دنیای واقعی) یا امضا کننده بلاک استَ که یک تراکنش را به‌صورت دیجیتال امضا و تایید می‌کند. یکی از تصمیمات مهم در هر شبکه بلاکچین این است که چه گره‌ای بلاکِ جدید را به شبکه اضافه می‌کند. این تصمیم به کمک ''مکانیزم اجماع'' گرفته می‌شود. مکانیزم اَجماع را در ادامه توضیح خواهیم داد. در زیر توضیح می‌دهیم که در یک شبکه بلاکچین تراکنش‌ها چگونه تایید می‌شوند و بلاک‌های جدید به شبکه اضافه می‌شوند.

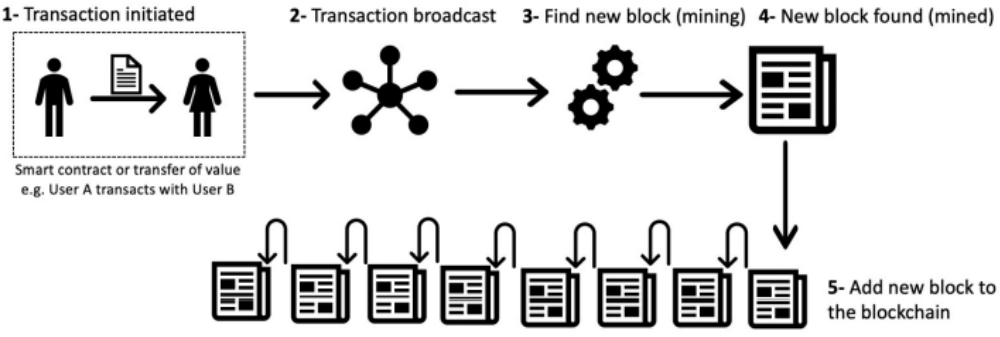
۱) شروع تراکنش: گره‌ای با ساختن یک تراکنش و سپس به‌صورت دیجیتالی امضا کردنش تراکنش را آغاز می‌کند. یک تراکنش می‌تواند نمایانگر چند عمل در بلاکچین باشد. یک تراکنش معمولا ساختمان داده‌ای است که یا نشانگر انتقال مقداری رمزارز بین دو کاربر در شبکه هستند یا قراردادِ هوشمندی اند که می‌تواند هر عمل مورد نیازی را اجرا کند. یک تراکنش می‌تواند بین دو یا چندَ کاربر اجرا شود.

۲) تراکنش تایید و منتشر می‌شود: تراکنش ابتدا توسط گره‌های دیگر تایید می‌شود و سپس به‌عنوان یک تراکنش معتبر در شبکه منتشر می‌شود.

۳) عملیات پیدا کردن بلاک جدید: وقتی تراکنش توسط گره‌هایی به نام ماینر ها تایید می‌شود، عملیات ماین کردن[۲۰] آغاز می‌شود. این عملیات گاها ''پیدا کردنِ بلاک جدید'' نیز نامیده می‌شود. در این حین ماینرها تلاش می‌کنند تا زودتر از گره‌های دیگر بلاکی که درست کرده‌اند را نهایی کنند تا در شبکه ثبت شود.

۴) بلاکِ جدید پیدا می‌شود: وقتی یک ماینر پازل محاسباتی را حل کند (در بالاتر دیدیم که باید عدد نانسَ‌ای را پیدا کند که شرایطِ تعیین شده برای عددِ هش برقرار شود) بلاکِ جدید پیدا شده تلقی می‌شود. در این شرایط تراکنش نهایی می‌شود. معمولا در شبکه‌های بلاکچینَ مانند بیتکوین به ماینری که پازل را حل کند مقداری رمزارز به عنوانِ پاداش داده می‌شود.

۵) اضافه کردن بلاک جدید به شبکه: بلاکی که تازه پیدا شده تایید می‌شود و تراکنش یا قراردادِ هوشمند اجرا می‌شود و به گره‌های دیگر اطلاع رسانی می‌شود. سپس این بلاک به شبکه اضافه می‌شود و برای همیشه در شبکه باقی می‌ماند.



---

[۱۹] miner

[۲۰] mining

در این فصل درباره مفاهیم مقدماتی در بلاکچین صحبت کردیم. در فصل‌های بعد به توضیحات بیشتری درباره برخی از این مفاهیم می‌پردازیم.

۷.۲ مزایا و معایب شبکه بلاکچین

مزایای مورد توجه شبکه بلاک‌چین به‌شرح زیر است:

- غیر متمرکز بودن: این موضوع یکی از مفاهیمِ اصلی شبکه بلاکچین است. برای تایید و اجرای یک قرارداد نیاز به شخص سوم و میانجی‌ای (مانند بانک) نیست و به‌جای آن از یک مکانیزم اجماع برای توافق درباره اعتبار قرار استفاده می‌شود.
- شفافیت و اعتماد: همه به اطلاعاتی که روی شبکه بلاکچین قرار دارد دسترسی دارند.
- در دسترس بودن: چون سیستم روی هزاران گره در یک شبکه همتابه‌همتا توزیع شده و هر گره کپی‌ای از شبکه را روی خود دارد، دسترسی به اطلاعات بسیار آسان می‌شود. اگر یکی از گره‌ها شبکه را ترک کند یا غیر فعال شود، شبکه همچنان به کارِ خود ادامه می‌دهد.
- امنیت بالا: همه تراکنش‌های روی شبکه رمزگذاری شده اند. همچنین هر تراکنشی که توسط گره‌ای منتشر می‌شود، توسط قوانینی توسط گره‌های دیگر چک می‌شود.
- تغییر ناپذیری: وقتی داده‌ای روی شبکه بلاکچین نوشته شود، پاک کردن یا عوض کردنش تقریبا غیر ممکن است. امکانِ عوض کردن داده وجود دارد اما بسیار مشکل است. درباره عوض کردن داده‌ای که روی شَبکه بلاکچین نوشته شده بیشتر در بخش حمله ۵۱ درصد توضیح می‌دهیم.
- معاملات سریع‌تر: در دنیای مالی بلاکچین میتواند نقش بزرگی در تسویه معاملات داشته باشد. در بلاکچین تأییدِ قرارداد زمان بر نیست زیرا تمام داده‌ها در یک دفتر کل توزیع شده که دو طرف قراردادِ به آن دسترسی دارند وجود دارد.
- صرف جوبی در هزینه: از آنجا که هیچ شخص سومی در معامله وجود ندارد، لازم نیست هزینه‌هایی که برای انجامِ معامله به شخص سوم داده می‌شد را پرداخت کرد.
- بستری برای قراردادهای هوشمند[۲۱]: در شبکه‌های بلاکچین جدید مانند اتریوم[۲۲] و مولتی‌چین[۲۳] این قابلیت برای کاربران وجود دارد که برنامه‌های کامپیوتری دلخواه خود را روی این شبکه‌ها اجرا کنند. به این برنامه‌ها قراردادِ هوشمند میگوییم. اما در شبکه‌های قدیمی‌تر مانند بیتکوین این قابلیت وجود ندارد. در بخش‌های بعدی بیشتر درباره قراردادهای هوشمند و زبان‌های برنامه‌نویسیِ مخصوص به شبکه‌های بلاکچین صحبت می‌کنیم.

مانند هر فناوری دیگری، برخی چالش‌ها باید در نظر گرفته شوند تا بتوانیم به سیستمی قوی‌تر و مفیدتر برسیم. فناوری بلاکچین نیز از این قاعده مستثنی نیست و تلاش زیادی در

---

[۲۱] smart contract
[۲۲] Ethereum
[۲۳] MultiChain

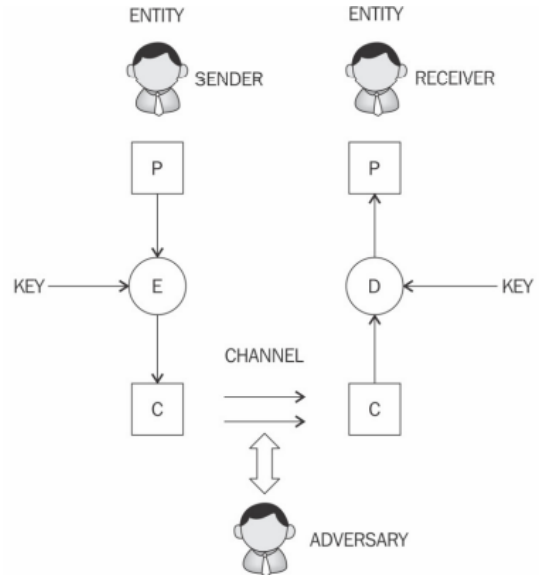
دنیاهای صنعت و دانشگاه برای غلبه بر این چالش‌ها انجام می‌شود. مشکلات اصلی عبارت از زیر اند:

- مقیاس پذیری: در حال حاضر، شبکه‌های بلاکچین، برخلاف شبکه‌های مالی فعلی، آنقدر مقیاس‌پذیر نیستند. دلیل اصلی این مشکل اجماعِ است زیرا برَای تایید یک تراکنش، تمام گره‌های در شبکه باید تراکنش را تایید کنند. به‌عنوان مثال در بیتکوین درصورت افزایش تعداد گره‌ها مشکلی نیست اما با افزایش تعدادِ تراکنش‌ها مشکل ایجاد می‌شود. دستیابی به مقیاس پذیری معمولاً مستلزم قربانی کردن غیرمتمرکز بودن، امنیت یا درجه‌ای از هر دو است.
- عدم استفاده توسط عموم: بلاکچین فناوری نوظهوری است و با اینکه این دیدگاه درحالِ تغییر است، اما راه زیادی تا استفاده روزانه توسط کاربران دارد. چالش اصلی در اینَ مسیر راحت‌تر کردن استفاده از این تکنولوژی است که باعث می‌شود کاربران بیشتری از آن استفاده کنند. همچنینی حل چالشِ مقیاس‌ذیری نیز در این مسیر کمک کننده است.
- قانون‌مند بودن : به‌خاطر غیر متمرکَز بودن، هیچ نهاد مرکزی وجود ندارد که همچنین باعث مشکل عدم استفاده توسط عموم است زیرا در روش‌های مالی سنتی، به‌خاطر وجود یک نهاد نظارتی، کاربران تا حدی احساس اطمینان دارند زیرا درصورت بروز مشکل کسی هست که پاسخ‌گو باشد. در بلاکچین چنین نهادِ پاسخ‌گویی وجود ندارد.
- فناوری نسبتا نابالغ: در مقایسه با فناوری‌های دیگر که چندین دهه مورد مطالعه قرار گرفته‌اند، بلاکچین همچنان فناوری نوظهوری است که برای به بلوغ رسیدن به تحقیق و پژوهش زیادی نیاز دارد.
- حریم خصوصی و محرمانه بودن: حریم خصوصی یکی از نگرانی‌های مربوط به بلاکچین‌های عمومی مانند بیتکوین است زیرا هرکسی می‌تواند هر تراکنشی که در شبکه اجرا می‌شود را ببیند. این شفافیت جزو یکی از مزایای بلاکچین بود اما در بسیاری از صنایع مطلوب نیست.

فصل ۳

رمزنگاری

رمزنگاری[۱] علم ایمن سازی اطلاعات با وجود مهاجمان است؛ با این فرض که مهاجمان منابع نامحدودی را در اختیار دارند.
سایفر[۲] الگوریتمی است که برای رمزنگاری و رمزگشایی داده استفاده می‌شود که اگر مهاجی توانست داده ارسال شده توسط فرستنده را که توسط سایفر رمزنگاری شده به‌دست آورد، به داده‌ای بی‌معنی برسد و بدون کلیدی خاص نتواند داده را رمزگشایی کند. شکل زیر نمونه‌ای از رمزگزاری و رمزگشایی را نشان می‌دهد:



در تصویر بالا وP E و C و D به‌ترتیب نشان‌گر متنِ عادی، رمزگزاری کردن، متن رمزگزاری شده

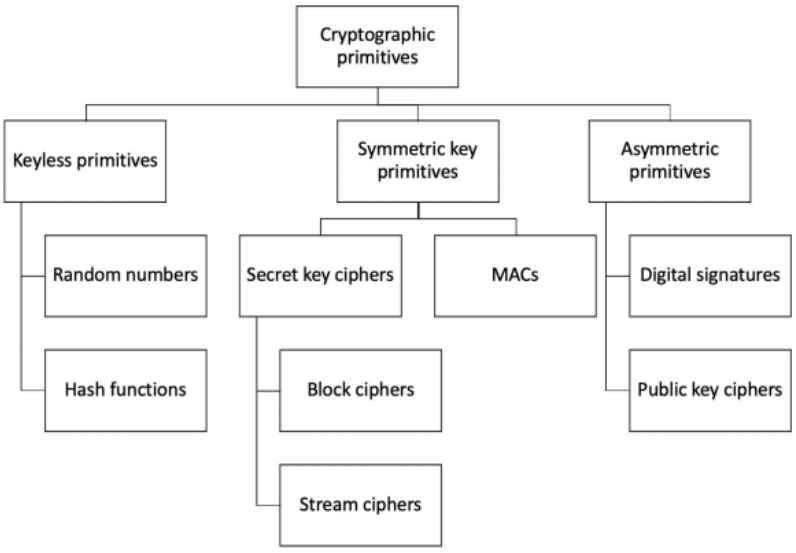
---
[۱] cryptography
[۲] cipher

و رمزگشایی هستند. در این مدل از مفاهیمی مانند نهاد[۳] ، فرستنده[۴] ، گیرنده[۵] ، مهاجم[۶] کلید[۷] و کانالِ ارسال داده[۸] است که در زیر توضیح می‌دهیم.

- نهاد: یک شخص یا سیستم که پیامی را می‌فرستد، می‌گیرد یا عملیاتی را روی داده انجام می‌دهد.
- فرستنده: نهادی است که داده را مخابره می‌کند
- گیرنده: نهادی که داده فرستاده شده را دریافت می‌کند
- مهاجم: نهادی که سعی می‌کند سرویس امنیتی را دور بزند
- کلید: داده‌ای است که به کمکش می‌توان داده خام را رمزگزاری کرد و همچنین داده رمزگزاری شده را رمزگشایی کرد
- کانال ارسال داده: به‌کمکش می‌توان بین دو نهاد ارتباط برقرار کرد

در رمزگزاری متقارن کلید رمزگزاری (که در اختیار فرستنده است) و کلید رمزگشایی (که در اختیار گیرنده است) یکسان هستند.

همان‌طور که در شکل زیر می‌بینید رمزنگاری به سه شاخه اصلی تقسیم می‌شود: رمزنگاری بدون کلید[۹] ، رمزنگاری متقارن[۱۰] و رمزنگاری نامتقارن[۱۱] تقسیم می‌شود.



---

[۳] entity
[۴] sender
[۵] receiver
[۶] adversary
[۷] key
[۸] channel
[۹] keyless primitive
[۱۰] symmetric key primitive
[۱۱] asymmetric key primitive

در ادامه برخی از این مفاهیم را توضیح می‌دهیم.

۱.۳ رمزنگاری بدون کلید

در این بخش به دو مفهوم تولید اعداد تصادفی و توابع هش می‌پردازیم.

۱.۱.۳ اعداد تصادفی

تولید اعداد تصادفی برای غیر قابل پیش‌بینی بودن الگوریتم‌ها در دنیای بلاکچین اهمیت دارد تا مهاجمان نتوانند نتیجه الگوریتم‌ها را حدس بزنند. البته تولید اعداد کاملا تصادفی همچنان کاری غیر ممکن است اما می‌توان اعدادی با درجه بالای تصادفی بودن تولید کرد.
دو دسته‌بندی در تولید اعداد تصادفی وجود دارد:

- تولید اعداد تصادفی [۱۲]:
یک RNG دستگاه یا تابع تولیدکننده اعداد تصدفی است. به‌عنوان مثال به‌کمک یک تاس می‌توان از بین اعداد یک تا شش، یکی را به‌صورت تصادفی انتخاب کرد و به هیچ صورت نمی‌توان نتیجه خروجی را حدس زد. در کامپیوترها محاسبات برمبنای منطق و با صفر و یک است پس با کامپیوتر نمی‌توان یک اعداد کاملا تصادفی ساخت. از این رو توابع شبیه ساز اعداد تصادفی در کامپیوترها استفاده می‌شود. به این توابع، توابع تولید اعداد شبه تصادفی می‌گوییم.

- تولید اعداد شبه تصادفی [۱۳]:
توابع تولید اعداد شبه تصادفی، توابع قطعی[۱۴] ای هستند که از یک عدد تصادفی اولیه[۱۵] استفاده می‌کند تا اعداد شبه تصادفی‌ای را به‌کمک یک الگوریتم قطعی تولید کند. از این الگوریتم‌ها برای تولید کلید رمزگزاری و رمزگشایی استفاده می‌شود. مثالی از این الگوریتم‌ها BBS[۱۶] است.

۲.۱.۳ توابع هش

از توابع هش برای رمزگزاری کردن اطلاعات استفاده نمی‌شود، بلکه داده ورودی را به رشته‌ای از طول ثابت تبدیل می‌کند. خانواده‌های مختلفی از توابع هش مانند MD، SHA۱، SHA۲ و RIPEMD، SHA۳ وجود دارند. علاوه بر استفاده توابع هش در شبکه بلاکچین و شبکه‌های همتا-به-همتا، از توابع هش در امضا دیجیتال و تایید احراز هویت نیز استفاده می‌شود.
ویژگی‌های تابع هش:

---

[۱۲] Random Number Generators - RNGs
[۱۳] Pseudorandom Number Generators - PRNGs
[۱۴] deterministic
[۱۵] seed
[۱۶] Blum-Blum-Shub

• **تبدیل پیام‌های دلخواه به رشته با طول ثابت: همان‌طور که قبلا گفته شد توابع هش باید بتوانند رشته‌ای از هر طولی را به‌عنوان ورودی بگیرند و به‌عنوان خروجی رشته‌ای از طولی ثابت بدهند. در بیشتر توابع اندازه رشته خروجی بین ۱۲۸ تا ۵۱۲ بیت است.**

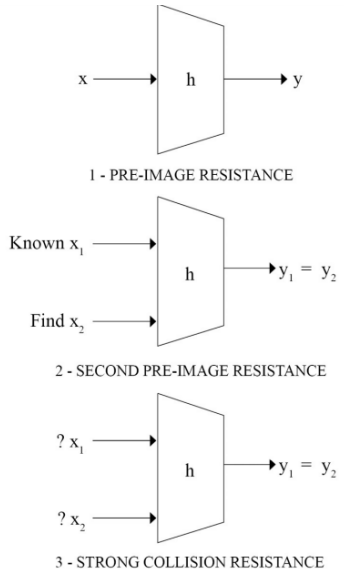
• **محاسبه آسان: محاسبه خروجی توابع هش، جدا از اندازه ورودی، آسان و سریع است.**

• **مقاومت در برابر تصویر اولیه و پردازش نشده:[۱۷] یکی از ویژگی‌های مطلوب توابع هش است که یعنی اگر یک خروجی داده شود، نباید بتوان ورودی را حدس زد. مثلا در معادله زیر $y = h(x)$ که h تابع هش، x ورودی تابع و y خروجیِ هش شده اند، اگر y را داشته باشیم نباید بتوان x را حدس زد. همچنین به x پیش تصویرِ [۱۸] y می‌گوییم.**

• **مقاومت در برابر تصویر پردازش نشده دوم:[۱۹] اگر ورودی $x۱$ را داشته باشیم به‌صورتی که $y = h(x۱)$ برقرار باشد، نباید بتوان $x۲$ ای را حدس زد کرد که $y۲ = h(x۲)$ باشد و همچنین $y۱ = y۲$ برقرار باشد.**

البته دقت شود به‌خاطر این‌که طول خروجی توابع هش ثابت است پس قطعا $x۲$ ای وجود دارد که $h(x1) = h(x2)$ باشد اما تابع هش باید به‌صورتی باشد که نتوان چنین $x۲$ ای را حدس زد.

• **مقاومت برخورد:[۲۰] پیدا کردن دو ورودی مثل $x1$ و $x2$ که $h(x۲) = h(x۱)$ باید مشکل باشد این سه ویژگی را می‌توان در تصویر زیر مشاهده کرد:**



در الگوریتم اثبات کار (PoW)[۲۱] که در مکانیزم اجماع به‌کار گرفته می‌شود، دو بار از الگوریتم SHA-۲۵۶ بری اثبات هزینه محاسباتی توسط ماینرها استفاده می‌شود. همچنین از

---

[۱۷] Pre-image resistance
[۱۸] Pre-image
[۱۹] Second pre-image resistance
[۲۰] collision resistance
[۲۱] Proof of Work

الگوریتم RIPEMD-۱۶۰ در تولید آدرس بیتکوین استفاده می‌شود.
در فصل‌های بعد بیشتر درباره الگوریتم اثبات کار و مکانیزم اجماع توضیح می‌دهیم.

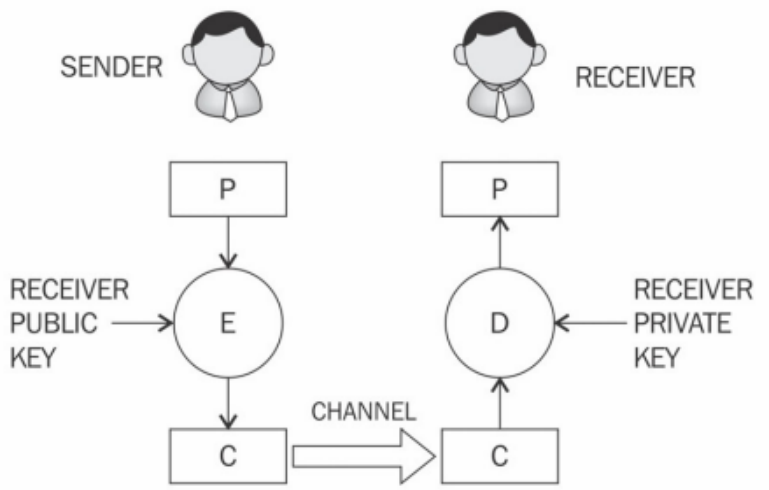
۲.۳ رمزگزاری کلید عمومی

اگر کلیدهایی که به کمکشان داده را رمزگزاری و رمزگشایی می‌کنیم با هم برابر باشند، از رمزنگاری متقارن استفاده کرده‌ایم. کلید رمزنگاری برای رمزگزاری و رمزگشایی لازم است و باید محرمانه باقی بماند برای همین به آن کلیدِ مخفی نیز می‌گویند.
بسته به پروتوکل کلیدهای متفاوتی وجود دارد:
- در سیستم‌های کلید متقارن، فقط به یک کلید نیاز هست که فرستنده و گیرنده آن‌را میدانند. از این کلید برای رمزگزاری داده اولیه و رمزگشایی داده‌ای که توسط فرستنده ارسال شده استفاده می‌شود.
- نوع دیگری از کلیدها کلید عمومی[۲۲] و کلید خصوصی[۲۳] است که باهم برای رمزنگاری کردن کلید عمومی یا رمزنگاری نامتقارن استفاده تولید می‌شوند. از کلید عمومی برای رمزگزاری متن ارسالی استفاده می‌شود درحالی که از کلید خصوصی در رمزگشایی به‌کار برده می‌شود و گیرنده پیام باید آن‌را مخفی نگه دارد.

پس چند نکته زیر را در رمزنگاری نامتقارن داریم:
- کلید عمومی همان کلیدی است که فرستنده از آن برای رمزنگاری کردن پیام ارسالی استفاده می‌کند.
- گیرنده پیام به‌کمک کلید خصوصی که فقط خودش در اختیار دارد پیام را رمزگشایی می‌کند.
- کلید عمومی توسط کلید خصوصی ساخته می‌شود و گیرنده پیام می‌تواند آن‌را به هرکسی که می‌خواهد به او پیام بفرستد بدهد.
- به‌کمک کلید عمومی نمی‌توان کلید خصوصی را حدس زد

در شکل زیر می‌توان روند ارسال پیام به کمک رمزنگاری نامتقارن را مشاهده کرد که مانند مثالی است که در ابتدای این فصل دیدیم.

---
[۲۲] public key
[۲۳] private key



ENCRYPTION DECRYPTION USING PUBLIC / PRIVATE KEY

از رمزنگاری نامتقارن می‌توان در امضای دیجیتالی[۲۴] و احراز هویت نیز استفاده کرد. برخلاف رمزنگاری متقارن، رمزنگاری نامتقارن فرایندی زمان‌گیر است. برای همین از رمزنگاری متقارن برای داده‌های بزرگ استفاده نمی‌شود.

- در سیستم‌هایی که از رمزنگاری متقارن استفاده می‌شود، ابتدا فرستنده باید کلید را به گیرنده از طریق یک راه ارتباطی ایمن ارسال کند. برای این کار از رمزنگاری نامتقارن استفاده می‌شود و سپس ارتباط بین دو گره با روش متقارن ادامه پیدا می‌کند.

---

digital signature[۲۴]

فصل ۴

اجماع

همان‌طور که گفتیم یکی از مهم‌ترین ویژگی‌های شبکه‌های بلاکچین و سایر دفاتر توزیع شده؛ غیرمتمرکز بودن است. درواقع غیرمتمرکز بودن این شبکه‌ها معنی این است که تمامی اطلاعات و داده‌ها، روی سرور اصلی ذخیره و پردازش نشده و بر روی چندین گره گوناگون ذخیره و نگه‌داری می‌شود.
از طرفی، اصلی‌ترین دلیل استفاده از دفاتر کل توزیع‌شده مثل بلاکچین، بالابردن سطح امنیت و اطمینان از عدم نابودی یا خرابی داده‌ها است. در حقیقت اگر اطلاعاتی که بر روی یکی از گره‌ها قرار دارند به دلایل مختلف خراب شوند و از بین بروند، بی‌شمار گره دیگر وجود دارند که این اطلاعات را بر روی خود ذخیره کرده‌اند.
اما در زمان استفاده از بلاکچین یا انواع دیگر دفاتر کل توزیع‌شده، یک موضوع بسیار حیاتی وجود دارد؛ درواقع زمانی‌که قرار باشد داده‌ها مداوم به‌روزرسانی گردند، این به‌روزرسانی الزاماً باید بر روی تمامی گره‌ها یا سرورها انجام شود و برای انجام این‌کار، از مکانیزم یا الگوریتمی به‌نام الگوریتم اجماع به‌کار برده می‌شود.

الگوریتم اجماع روشی است که از طریق آن، تمام افراد حاضر در شبکه بلاک چین به یک توافق مشترک در مورد وضعیت حال حاضر دفترکل توزیع‌شده می‌رسند. بنابراین، الگوریتم‌های اجماع، اعتبار را در شبکه‌ی بلاکچین و اعتماد را بین گره‌ها ایجاد می‌کنند.
با کمک الگوریتم اجماع، به توافق رسیدن بین نودها از طریق یک سیستم رای‌گیری صورت می‌گیرد. به این صورت که اگر یک کاربر تراکنشی (هر نوع داده‌ای را شامل می‌شود) را به شبکه ارسال کند، اطلاعات ارسال شده توسط همه‌ی گره‌ها بررسی می‌شود. اگر با توجه به اطلاعات قبلی، تراکنش فرستاده شده صحت داشته باشد، گره یک تاییدیه مبنی بر صحت آن به شبکه ارسال می‌کند.
مجموعه‌ای از تراکنش‌ها یک بلاک را تشکیل می‌دهند. اگر بیش‌از ۵۱ درصد گره‌ها، اضافه شدن این بلاک را به بلاکچین تایید کنند، نودها بلاک جدید را به سیستم خود اضافه می‌کنند و تراکنش‌های داخل آن موفق و نهایی می‌شود.

یکی از الزامات مکانیزم اجماع این است که باید تحمل خطا[1] داشته باشد. یعنی درصورت وجود تعدادی مشکل در شبکه باید بازمهم بتواند به‌کار خود ادامه دهد. طبیعی است که سیستم می‌تواند تا تعداد مشخصی خطا را تحمل کند زیرا هیچ شبکه نمی‌تواند درصورت خطا داشتن اکثر گره‌هایش به‌درستی کار کند.

۱.۴ انواع الگوریتم‌های اجماع

الگوریتم‌های اجماع را می‌توان براساس توانایی تحمل خطا به دو دسته تحمل خطای تصادف[2] و تحمل خطای بیزانسی[3] تقسیم کرد.
CFT فقط قادر به تحمل فروپاشی‌ها (یا اصطلاحا کرش) است. اما سیستم‌های BFT تحمل کاملی دربرابر واکنش‌ها و رفتارهای بیزانسی[4] دارند.
ابتدا مساله ژنرال‌های بیزانسی را توضیح می‌دهیم.

فرض کنید گروهی از فرماندهان بیزانسی یک شهر را به محاصره خود درآورده‌اند و هر یک از ایشان فرماندهی بخشی از ارتش بیزانس را به عهده دارد. پیش از آغاز روز بعد، فرماندهان باید در مورد حمله به شهر یا عقب‌نشینی تصمیم‌گیری کنند. برخی از ایشان طرفدار حمله به شهر هستند و برخی دیگر تمایلی به حمله ندارند و می‌خواهند عقب‌نشینی کنند. اما آنچه بیش از اقدام به حمله یا اقدام به عقب‌نشینی اهمیت دارد، انجام یک اقدام واحد توسط همه فرماندهان است زیرا در صورت حمله‌ی ناقص یا عقب‌نشینی ناقص، جان تعداد زیادی از سربازان به خطر میافتد. در میان این ارتش فرماندهان خائنی وجود دارد که قصد دارند این هماهنگی را بر هم بزنند. آن‌ها به برخی از فرماندهان میگویند که حمله می‌کنیم و به برخی دیگر میگویند که عقب‌نشینی می‌کنیم و بدین‌صورت در میان کل ارتش اختلاف ایجاد می‌کنند.
از سوی دیگر، به دلیل آنکه فرماندهان ازنظر مسافت با یکدیگر فاصله‌دارند، برای انتقال پیام‌ها از نامه‌رسان استفاده می‌کنند. استفاده از نامه‌رسان‌ها راه‌حل مشکل را پیچیده‌تر می‌کند، زیرا ممکن است این نامه‌رسان‌ها نامه را به‌درستی به فرمانده هان دیگر انتقال ندهند و یا حتی بین راه کشته شوند و پیام به مقصد نرسد.
اگر دقت کنید در مساله بالا می‌توان هر فرمانده را به‌عنوان گره‌ای در سیستم و هر پیام‌رسان را به‌عنوان رابط بین دو گره در نظر گرفت که ممکن است هردو اشتباه کنند.
فقط سیستم‌های غیرمتمرکز مستعد مشکل ژنرال‌های بیزانسی هستند، زیرا فاقد منبع اطلاعاتی قابل اعتماد هستند و هیچ راهی برای تأیید اطلاعاتی که از سایر کاربران شبکه دریافت می‌کنند ندارند. در سیستم‌های متمرکز، یک مرجع برای انتشار اطلاعات دقیق و در عین حال جلوگیری از انتشار اطلاعات اشتباه یا تقلبی در سراسر شبکه مورد اعتماد است.
PBFT[5] یک الگوریتم اجماع است که در سال ۱۹۹۹ منتشر شد و مساله ژنرال‌های بیزانسی

---

[1] fault tolerance
[2] Crash fault-tolerance - CFT
[3] Byzantine fault-tolerance - BFT
[4] Byzantine
[5] Byzantine Practical Fault Tolerance

را حل می‌کند.
از الگوریتم‌های CFT میتوان Paxos و Raft را نام برد.
الگوریتم‌های معروف BFT شامل گواه اثبات کار[۶]، گواه اثبات سهام[۷] و تحمل خطای بیزانس عملی[۸] هستند. PoW به اجاع ناکاموتو هم معروف است که الگوریتم اجماعی بود که ناکاموتو برای بیتکوین به‌کار برد.
همانندسازی[۹] رویکردی استاندارد برای مقاوم‌کردن سیستم در برابر خطا است. همانندسازی باعث این می‌شود که یک کپی از داده در تمام گره‌های شبکه وجود داشته باشد. این کار تحمل خطا و همچنین ویژگی در دسترس بودن شبکه را بهتر می‌کند. این بدان معنی است که حتی اگر برخی گره‌ها معیوب باشند، شبکه کلی در دسترس است زیرا اطلاعات در چند گره ذخیره شده‌اند.
دو نوع همانندسازی داریم:

- همانند سازی فعال[۱۰] که در این روش گره‌ها کپی‌ای از ماشین حالت را در خود ذخیره می‌کنند. همانند سازی منفعل[۱۱] که در این روش فقط یک گره کپی‌ای از ماشین حالت را در اختیار دارد.

الگوریتم‌های اجماع را می‌توان به دو گروه تقسیم کرد:

- سنتی- براساس رای دادن
- مبتنی بر قرعه‌کشی- الگوریتم ناکاموتو و پس از ناکاموتو

اجماع مبتنی بر رای‌گیری سنتی در سیستم‌های توزیع شده چندین دهه مورد تحقیق واقع شده‌اند. الگوریتم‌های مختلفی بر این زمینه مانند Paxos و PBFT طراحی شده‌اند. این گروه از الگوریتم‌ها پیش ار بیتکوین وجود داشتند.
الگوریتم‌های ناکاموتو (یا مبتنی بر قرعه‌کشی) برای اولین بار با بیتکوین معرفی شدند. الزامات اصلی این الگوریتم‌ها امنیت و پیشرفت هستند.
امنیت: یعنی معمولا هیچ اتفاق بدی نیفتد. سه ویژگی وجود دارد که باید برقرار باشند:

- توافق: طبق این ویژگی هیچ دو فرایندی نباید درمورد یک مساله جواب‌های متفاوت به‌دست آورند.
- اعتبار: اگر فرایندی جوابی برای سوالی به‌دست آورده، این جواب باید براساس یک فرایند پیشنهاد شده باشد. یعنی این جواب باید نتیجه یک فرایند باشد و از هیچی ارایه نشده باشد.
- تمامیت: هر فرایند فقط یک بار جواب برمیگرداند.

---

[۶] Proof of Work - PoW
[۷] Proof of Stake - Pos
[۸] Practical Byzantine Fault Tolerance - PBFT
[۹] replication
[۱۰] active replication
[۱۱] passive replication

پیشرفت: یعنی سیستم در یک حالت نمی‌ایستد و پیشرفتی حاصل می‌شود.
هر گره صادق باید در نهایت جوابی را انتخاب کند.

PoW

اجماع ناکاموتو[۱۲] یا اثبات کار[۱۳] برای اولین بار در بیتکوین در سال ۲۰۰۸ معرفی شد. مکانیزم اجماع برای کاهش حملات سیبیل[۱۴] طراحی شده. حمله سیبیل، حمله‌ای است که سعی دارد روی بیشتر بخش‌های سیستم کنترل داشته باشد. حمله سیبیل معمولا توسط یک گره انجام می‌شود. این گره تعداد زیادی هویت جعلی می‌سازد و سعی می‌کند به‌کمک این هویت‌ها؛ تاثیر نامطلوبی روی سیستم داشته باشد. ساخت هویت‌های جعلی کار ساده‌ای است اما در بیتکوین به‌خاطر نیاز به اثبات داشتن نیروی محاسباتی کافی، انجام این کار سخت است.
در PoW هر گره باید عددی به نام نانس[۱۵] را پیدا کند که شرایط زیر را برقرار کند:

$$H(N||PrevHash||Tx||Tx||...Tx) < Target$$

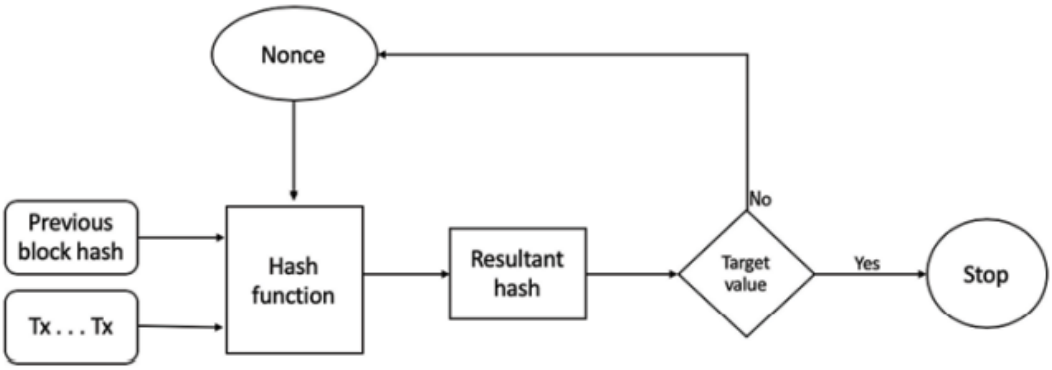
توضیح گزاره بالا به‌صورت زیر است:

- تراکنش‌هایی که تایید شده اند در کل شبکه منتشر می‌شوند. این تراکنش‌ها را با Tx نشان می‌دهیم.
- گره‌ها این تراکنش‌ها را دریافت می‌کنند و در یک بلاک قرار می‌دهند.
- گره‌ها هشِ بلاک قبلی را می‌خوانند که با PrevHash مشخص شده/
- بلاک فعلی درجه سختی‌ای دارد که از قبل تعیین شده و با Target نشان می‌دهیم.
- حالا گره‌ها باید عددی به نام nonce را پیدا کنند که از هش کردن تراکنش‌ها، هش بلاک قبلی و این عدد نانس، شرایط گزاره بالا برقرار شود.
- هر کس که نانس را پیدا کرد، این عدد را به بقیه اطلاع می‌دهد و درستی‌اش بررسی می‌شود. سپس بلاک جدید با این عدد نانس ساخته می‌شود. دقت کنید که یک جواب وجود ندارد ممکن است شرایط با عدد نانس دیگری نیز براورده شود اما اولین عدد نانس‌ای که پیدا شد در بلاک قرار داده می‌شود.

برای این‌که حل پازل بالا هزینه محاسباتی دارد، امکان sybil attack بسیاری پایین است چون هیچ گروهی در دنیای واقعی کامپیوترهای لازم برای فراهم کردن قدرت محاسباتی‌ای که از بقیه قسمت‌های شبکه بلاکچین بیشتر شود را ندارد.
در تصویر زیر می‌توان نمودار اثبات کار را دید:

---

[۱۲] Nakamoto consensus
[۱۳] Proof of Work - Pow
[۱۴] sybil attack
[۱۵] nonce



توضیحات ناکاموتو درباره اثبات کار را می‌توان اینجا مشاهده کرد.

فصل ۵

بیتکوین

برای چندین دهه ارزهای دیجیتال مورد مطالعه قرار گرفته‌اند. اولین پیشنهاد برای ارز دیجیتال به سال ۱۹۸۲ توسط دیوید چام[۱] که در زمینه علوم کامپیوتر و رمزنگاری فعالیت می‌کرد برمی‌گردد. در سال ۱۹۹۵ شرکت او دیجی‌کش که یک شرکت پول الکترونیکی بود، اولین ارز دیجیتال را با ای‌کش[۲] ایجاد کرد.

در سال ۲۰۰۸ بیتکوین در مقاله‌ای به‌نام :Bitcoin A Peer-to-Peer Electronic Cash System توسط شخص ناشناسی با نام مستعار ساتوشی ناکاموتو معرفی شد. ایده کلیدی ارایه شده در این مقاله ارز دیجیتالی بود که کاملا همتا ـ به ـ همتا است و نیاز به هیچ رابطی مانند بانک برای انتقال پول ندارد.

بیتکوین نتیجه سال‌ها تحقیق و مطالعه روی زمینه‌هایی مانند رمزنگاری و سیستم‌های توزیع شده مانند درخت مرکل، توابع هش و امضای دیجیتال است که از همه آن‌ها برای ساخت بیتکوین استفاده شده. بیتکوین مشکلات بسیاری که مربوط به ارز دیجیتال و سیستم‌های توزیع شده بود را حل می‌کند، شامل:

- مساله ژنرال‌های بیزانسی: که مشکل مورد نظر را در فصل رمزنگاری توضیح دادیم. در بیتکوین از الگوریتم PoW استفاده شده که این مساله را حل می‌کند.
- دو بار خرج کردن: یک مشکل احتمالی و بالقوه در طرح و برنامه ارز دیجیتال است که در آن هر رمز دیجیتال یا توکن می‌تواند بیشتر از یک بار خرج شود. بر خلاف پول فیزیکی (اسکناس)، هر رمز دیجیتال شامل یک فایل دیجیتالی است که می‌تواند تکثیر یا جعل شود.

در سیستم‌های متمرکز یک طرف سوم قابل اعتماد مرکزی که می‌تواند مشخص کند که ارزی مصرف شده یا نه که نیاز به اعتماد به این شخص سوم دارد.

در سیستم‌های غیرمتمرکز از الگوریتم اجماع مانند PoW استفاده می‌شود. به‌عنوان مثال تراکنش‌ها در یک بلاک هستند و بلاک‌ها توسط لیست پیوندی به هم متصل اند. هر سرور می‌تواند با حل پازلی محاسباتی (که به انجام این کار ماین کردن می‌گویند) بلاکی را بسازد.

---

[۱] David Chaum

[۲] ecash

در مساله دوبار خرج کردن اگر دو بلاک (با تراکنش‌های متفاوت) ساخته شوند، شبکه تنها یکی را قبول می‌کند و به بلاکچین اضافه می‌کند که این‌کار توسط الگوریتم PoW اتفاق می‌افتد. بلاکی اضافه می‌شود که حداقل ۵۱ درصد گره‌ها آن را قبول کنند و در ازایش این گره‌ها جایزه‌ای دریافت می‌کنند. برای این‌که گره‌ها جایزه‌شان را دریافت کنند، اکثرا بلاک درست را قبول می‌کنند. این سیستم دربرابر حمله ۵۱ درصد[۳] آسیب پذیر است.

- حمله سیبیل: [۴] حمله سیبیل یک نوع حمله سایبری است که در آن یک کاربر می‌تواند با ایجاد چندین حساب جعلی، کنترل شبکه را تا حدی در دست بگیرد و اقدامات خرابکارانه انجام دهد. در این‌جا نیز الگوریتم PoW از این حمله جلوگیری می‌کند. کاربران باید مقدار معینی هزینه محاسباتی محتمل شوند تا پاداشی بگیرند. اگر گره‌های جعلی بخواهند بلاکی به شبکه اضافه کنند، پذیرفته نمی‌شوند زیرا میزان کار (هزینه محاسباتی) خواسته شده را انجام نداده اند. در این‌جا نیز درصورتی که کسی بتواند حمله ۵۱ درصد انجام دهد، سیستم آسیب پذیر است. در حمله ۵۱ درصد باید کسی حداقل ۵۱ درصد گره‌های در شبکه را در اختیار داشته باشد و علاوه بر این، تمام بلاک‌هایی که تا الان ماین شده اند را دوباره ماین کند. به‌خاطر هزینه محاسباتی بالا، بلاکچین‌های بزرگ مانند بیتکوین از این حمله در امان هستند.

بیتکوین از عناصر زیر تشکیل شده است.

- کلیدهای دیجیتال
- آدرس‌ها
- تراکنش‌ها
- بلاکچین
- ماینرها
- شبکه بیتکوین
- کیف‌پول ها[۵]

در شبکه بیتکوین، مالکیت ارزِ بیتکوین و ارسال آن به شخص دیگری با تراکنش‌ها، وابسته به کلیدهای خصوصی، کلیدهای عمومی و آدرس‌ها است. در بیتکوین از رمزنگاری منحنی بیضوی[۶] برای ساخت جفت کلیدهای خصوصی و عمومی استفاده می‌شود.

کلیدهای خصوصی را فقط مالک کیف پول در اختیار دارد و از آن برای امضای تراکنش‌ها، جهت اثبات مالکیت، استفاده می‌شود.

کلیدهای خصوصی اعدادی ۲۵۶ بیتی هستند.

کلیدهای عمومی روی شبکه بلاکچین موجود هستند و تمام کاربران شبکه می‌توانند آن را ببینند. کلیدهای از کلیدِ خصوصی ساخته شده‌اند. وقتی تراکنشی توسط کلید خصوصی امضا

---

۳ ۵۱ percent attack

۴ sybil attack

۵ wallets

۶ Elliptic Curve Cryptography - ECC

می‌شود و روی شبکه منتشر می‌شود، از کلیدهای عمومی استفده می‌شود تا چک شود که تراکنش واقعا توسط گره‌ای که تراکنش را منتشر کرده امضا شده.

۱.۵ تراکنش‌ها

تراکنش‌ها هسته بیتکوین هستند. یک تراکنش می‌تواند به سادگی فرستادن چند بیتکوین به یک آدرس یا پیچیده‌تر باشند. هر تراکنش از حداقل یک ورودی و یک خروجی تشکیل شده. به ورودی می‌توان به‌عنوان سکه‌ها(کوین) ای نگاه کرد که در تراکنش قبلی‌ای درست شده‌اند و حالا در حال خرج شدن هستند، و به خروجی هم می‌توان به عنوان سکه‌هایی که در حال درست شدن هستند نگاه کرد. اگر یک تراکنش سکه‌های جدید ضرب می‌کند، پس هیچ ورودی‌ای ندارد و درنتیجه نیاز به امضا کردن هم ندارد. اگر تراکنشی به کاربر دیگری سکه بفرستد، پس باید توسط کلید خصوصی فرستنده امضا شود. در این حالت باید به تراکنش قبلی هم ارجاع داده شود تا مبدا سکه‌ها مشخص شود.
تراکنش‌ها رمزنگاری نمی‌شوند و برای عموم برروی شبکه بلاکچین قابل مشاهده اند. هر بلاکی نیز از چند تراکنش تشکیل شده و کاربران می‌توانند اطلاعات در یک بلاک را بخوانند.
نگاهی به اتفاقاتی که برای اجرای یک تراکنش می‌افتد بیاندازیم:

۱. ابتدا ارسال کننده پول تراکنشی را به کمک برنامه کیف‌پولش می‌فرستند.
۲. سپس برنامه کیفِ پول تراکنش را با کلید خصوصی امضا می‌کند
۳. تراکنش روی شبکه منتشر می‌شود و بقیه کاربران از آن مطلع می‌شوند..
۴. درست بودن تراکنش توسط ماینرها تایید می‌شود.
۵. تراکنش به‌عنوان نامزد برای اضافه شدن به بلاک قرار گرفته می‌شود.
۶. وقتی تراکنش ماین شد و توسط الگوریتم اجماع قبول شد، بر روی شبکه قرار می‌گیرد.

هنگامی که یک تراکنش توسط کاربری روی شبکه فرستاده می‌شود، در محل خاصی در کلاینتِ نرم افزار بیتکوین قرار می‌گیرد. به این مکان استخر تراکنش[۷] گفته می‌شود.
استخر تراکنش یا استخر حافظه، جایی در خافظه محلی هر گره متصل به شبکه بیتکوین است که لیستی از تراکنش‌هایی که هنوز به بلاک فعلی اضافه نشده‌اند را نگه می‌دارد. ماینرها معمولا تراکنش‌هایی را که جایزه بزرگ‌تری دارند قبول می‌کنند.
هزینه تراکنش[۸] توسط ماینرها تعیین می‌شود. هزینه معمولا بسته به اندازه تراکنش است و با منهاکردن تعداد ورودی از خروجی محاسبه می‌شود.

$$\text{fee} = \text{sum}(\text{inputs}) - \text{sum}(\text{outputs})$$

البته که هزینه تراکنش‌ها همیشه ثابت نیستند و حتی تراکنش‌هایی هستند که هیچ جایزه‌ای به ماینر نمی‌دهند اما صرفا ممکن است زمان زیادی طول بکشد تا در شبکه ثبت شوند.

یک تراکنش از اعضای زیر تشکیل شده:

---
[۷] transaction pool
[۸] transaction fee

• ورژن: عددی ۴ بایتی است و قوانینی را برای پردازش تراکنش توسط گره‌ها مشخص می‌کند. دو نوع ورژن برای تراکنش وجود دارد، ۱ یا ۲.
• تعداد ورودی‌ها : ۱ تا ۹ بایت است که عددی مثبت است و تعداد ورودی‌های تراکنش را مشخص می‌کند.
• لیست ورودی‌ها: هر ورودی از چند قسمت تشکیل شده که شامل هش تراکنش قبلی، اندیس تراکنش قبلی، طول متن تراکنش و متن تراکنش است.
همچنین در تراکنش اولِ هر بلاک بیتکوین‌هایی که در این بلاک خرج شده‌اند مشخص می‌شود.
• تعداد خروجی‌ها: که ۱ تا ۹ بایت است.
• لیست خروجی‌ها: که نشان دهنده گیرنده‌های بیتکوین ارسال شده در تراکنش است.
• قفل زمانی: که به کمکش ارسال کننده تراکنش می‌تواند تعیین کند زودترین زمانی که تراکنش می‌تواند پرداخت شود کی باشد. مثلا می‌توان تراکنش را الان ساخت و ارسال کرد ولی تعیین کرد که تا حداقل یک هفته بعد نمی‌تواند ماین شود و اجرا شود. این عدد ۴ بایت است.

۲.۵ بلاک

همان‌طور که قبلا گفتیم بلاکچین یک دفتر کل توزیع شده از تراکنش‌ها است. بلاکچین بیتکوین هم یک دفتر کل توزیع شده است که تمام تراکنش‌هایی که در شبکه بیتکوین اتفاق افتاده را با ترتیب، با ثبت زمان تراکنش و به‌صورت غیر قابل تغییر ذخیره کرده. ماینرها تراکنش‌ها را تایید می‌کنند و پس از اجماع روی بلاک فعلی قرار می‌دهند. هر بلاک توسط یک هش شناخته می‌شود و با لیست پیوندی به هشِ بلاک قبلی متصل است.
یک بلاک در بیتکوین شامل چهار قسمت زیر است:

• اندازه بلاک: ۴ بایت است و اندازه بلاک را نشان می‌دهد.
• سرتیتر بلاک: ۸۰ بایت است و چند قسمت دارد که در بخش بعد توضیح می‌دهیم.
• شمارنده تراکنش: اندازه این قسمت متفاوت است و شامل تعداد تمام تراکنش‌هایی که در بلاک فعلی است را نشان می‌دهد. اندازه‌اش بین ۱ تا ۹ بایت است.
• تراکنش‌ها: اندازه اش متفاوت است و تراکنش‌های موجود در بلاک را نگه می‌دارد.

گفتیم سرتیتر بلاک شمال چند قسمت مختلف است که در زیر توضیح می‌دهیم:

• ورژن: عددی ۴ بایتی است که قوانینی که باید برای تایید بلاک رعایت شود را تعیین می‌کند.
• هش سرتیتر بلاک قبلی: عددی ۳۲ بایتی است که با دوبار هش کردنِ هش سرتیتر بلاک قبلی توسط الگوریتم SHA۲۵۶ به‌دست میاید.
• ریشه درخت مرکل: ریشه درخت مرکل تراکنش‌های در بلاک را به‌دست میاوریم و با دوبار هش کردن آن توسط الگوریتم SHA۲۵۶ آن‌را به‌دست میاوریم. این عددی ۳۲ بایتی است
• مهر زمانی: عددی ۴ بایتی که زمان حدودی ساخته شدن بلاک را نشان می‌دهد.
• درجه دشواری بلاک: عددی ۴ بایتی است. گفتیم ماینرها باید برای ماین کردن تراکنش/بلاک

باید پازلی را حل کنند. میزان سختی این پازل را درجه دشواری می‌گوییم.
• نانس:[9] این عددی است که ماینرها باید پیدا کنند تا پازل حل شود و پیدا کردن این عدد به میزان دشواری بستگی دارد. این عدد ۴ بایت است.

۳.۵ ماین کردن

ماین کردن[10] فرایندی است که در آن بلاک‌ها جدید به بلاکچین اضافه می‌شوند. بلاک‌ها شامل تراکنش‌هایی هستند که توسط فرایند ماین کردن تایید شده اند. بعد از این‌که بلاک ماین شد و تایید شد، به شبکه اضافه می‌شود و این باعث می‌شود که بلاکچین همیشه در حال رشد باشد. این فرایند به‌خاطر الزامات الگوریتم PoW هزینه‌بر است و ماینرها باید عدد نانس‌ای را پیدا کنند که پازل الگوریتم را جل کند. وجود این پازل به این دلیل است که اطمینان حاصل کسب کند که قبل از ساخته شدن بلاک جدید مطمین شد ماینرها هزینه مورد نیاز را صرف کرده اند. با حل پازل PoW ماینرها سکه (کوین) جدید ضرب می‌کنند که این فرایند هزینه زیادی از جمله هزینه محاسباتی و برق مصرف می‌کند. همچنین این فرایند سیستم را در برابر کلاه‌برداری و حمله دوبار خرج کردن ایمن می‌کند.

تقریبا هر ده دقیقه یک بلاک جدید ماین می‌شود تا فرکانس تولید بیتکوین را کنترل کند. این فرکانس توسط شبکه بیتکوین کنترل می‌شود تا عرضه مالی را کنترل کند. در روز حدود ۱۴۴ بلاک ماین می‌شود و حدود ۱۷۲۸ بیتکوین جدید ضرب می‌شود. از آن‌جا که عرضه بیتکوین محدود است، تا سال ۲۱۴۰ تمام ۲۱ میلیون بیتکوین ماین می‌شوند اما ماینرها همچنان می‌توانند با کارمزد اجرای تراکنش‌ها درآمد کسب کنند.

زمانی که یک ماینر (گره) به شبکه بیتکوین متصل می‌شود، وظایف مختلفی را انجام می‌دهد:

- هماهنگ شدن با شبکه : زمانی‌که یک گره به شبکه بیتکوین اضافه می‌شود، با درخواست از گره‌های دیگر بلاکچین را دانلود می‌کند. البته بسته به شرایط و نوع گره لزوما همه بلاکچین را دانلود نمی‌کند.
- اعتبارسنجی تراکنش : تراکنش‌هایی که روی شبکه منتشر می‌شوند توسط ماینرها با چک کردن درست بودن امضا و خروجی‌های قرارداد اعتبارسنجی می‌شوند.
- اعتبارسنجی بلاک : ماینرها می‌توانند بلاک‌ها را اعتبارسنجی کنند. این کار شامل تایید هر تراکنش در بلاک و همچنین تایید مقدار نانس بلاک است.
- ساختن بلاک جدید : ماینرها بعد از تایید کردن تراکنش‌هایی که رو شبکه منتشر شده‌اند، می‌توانند با آن‌ها بلاک جدیدی بسازند.
- اجرای PoW : این وظیفه، هسته اصلی فرایند ماین کردن که که ماینرها با حل پازلی محاسباتی، بلاک معتبر را پیدا می‌کنند. در بخش‌های قبل گفتیم که در قسمت سرتیتر بلاک[11] عددی ۳۲ بیتی به نام نانس وجود دارد که ماینرها باید آن‌را مدام عوض کنند و عدد جدیدی به‌جایش امتحان کنند تا هش به‌دست آمده از بلاک (که وابسته به همین عدد نانس

---
[9] nonce
[10] mining
[11] block header

هم هست) شرایط پازل را برقرار کند.

- دریافت جایزه : زمانی‌که یکی از گره‌ها پازل را حل کند، عدد نانس را به بقیه گزارش می‌دهد و گره‌های دیگر درستی جوابش را تایید می‌کنند و بلاک جدید را میسازند. ممکن است به‌خاطر پیدا شدن جواب دیگری توسط یک گره دیگر در تقریبا همان زمان یکی از این بلاک‌ها قبول نشوند اما وقتی بلاکی قبول شد، به ماینر ۲۵.۶ بیتکوین جایزه داده می‌شود و علاوه بر آن اگر در اجرا و تایید تراکنشی در آن بلاک تاثیر داشت، جایزه آن را هم دریافت می‌کند.

گفتیم اگر ماینر بتواند بلاک جدید را با حل پازل PoW پیدا کند، جایزه‌ای دریافت می‌کند. همچنین ماینرها برای اجرا و تایید تراکنش‌ها نیز هزینه‌ای دریافت می‌کنند. تقریبا هر ۱۰ دقیقه یک بلاک جدید ساخته می‌شود. و همچنین جایزه‌ای که به ماینر برای کشف بلاک داده می‌شود پس از ساخته شدن هر ۲۱۰۰۰۰ بلاک نصف می‌شود. این زمان تقریبا برابر با ۴ سال است. زمانی‌که بیتکوین در سال ۲۰۰۹ آغاز شد، جایزه ساخت بلاک ۵۰ بیتکوین بود و به‌مرور نصف شد و درحال حاضر ۲۵.۶ بیتکوین برای هر بلاک است. این مکانیزم باعث جلوگیری از تورم و کنترل عرضه بیتکوین می‌شود.

برای اینکه ماینرها جایزه‌شان را دریافت کنند، باید نشان دهند که پازل محاسباتی را حل کرده‌اند. به این کار PoW یا اثبات کار گفته می‌شود.

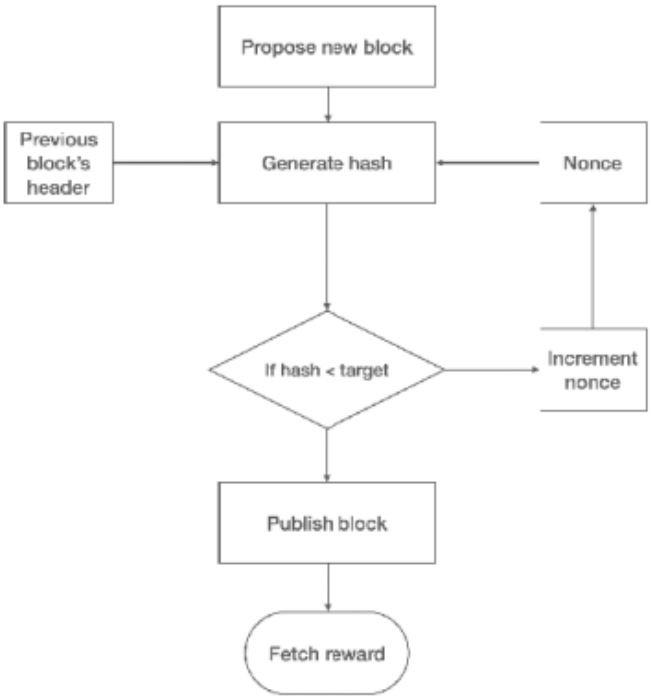
در PoW اثبات می‌شود که ماینر منابع محاسباتی کافی برای ساخت بلاک جدید را خرج کرده. در زیر می‌توان پازلی که باید حل کرد را دید:

$$H(N||PrevHash||Tx||Tx||...Tx) < Target$$

که در اینجا N عدد نانس است، PrevHash هش بلاک قبلی است، Tx ها تراکنش‌های موجود در بلاک هستند ، Target عددی است که سختی مساله را مشخص می‌کند و H تابع هش است که PrevHash N، و تراکنش‌ها را با هم هش می‌کند و یک عدد به‌دست میاید. پازل به این صورت است که ماینر باید عدد نانس‌ای را به‌دست آورد که جواب تابع هش از Target کوچک‌تر باشد. هرچه Target کوچک‌تر باشد، سختی پازل بیشتر است و هر چند وقت یکبار به سختی این پازل اضافه می‌شود.

پس الگوریتم ماین کردن به‌صورت زیر است:

- سرتیتر بلاک قبلی از شبکه بیتکوین بازخوانی می‌شود.
- تعدادی از تراکنش‌هایی که روی شبکه منتشر شده و تایید شده اند جمع آوری می‌شوند.
- عدد نانسی انتخاب می‌شود و با هش بلاک قبل و تراکنش‌ها دوبار به‌کمک الگوریتم SHA۲۵۶ هش می‌شود.
- بررسی می‌شود که آیا خروجی هش از میزان ساختی پازل یا همان مقدار target کوچک‌تر هست یا نه. اگر شرایط برقرار بود بلاک جدید به همراه نانس به شبکه گزارش می‌شود تا درستی‌اش تایید شود و این بلاک به بلاکچین اضافه می‌شود.
- اگر نانس شرایط را برقرار نکرد، دوباره نانس دیگری انتخاب می‌شود و الگوریتم تکرار می‌شود.



شکل ۱.۵: فرایند ماین کردن

فصل ۶

قراردادهای هوشمند و بلاکچین اتریوم

۱.۶ قرارداد هوشمند

در سال ۱۹۹۶ نیک سزابو[۱] ایده قرارداد هوشمند[۲] را معرفی کرد. تعریف وی از قرارداد هوشمند به‌شرح زیر است:

> قرارداد هوشمند یک پروتوکول تراکنش الکترونیکی است که مفاد قرارداد را اجرا می‌کند. اهداف اصلی‌اش راضی کردن شرایط قرارداد، کمینه کردن کردن استثناها، چه قصدی و چه از روی اشتباه و همچنین کمینه کردن نیار به اعتماد به واسطه‌ها است. اهداف مالی متشکر هم شامل کم کردن کلاه برداری، هزینه‌های نظارت و اجرا و هزینه‌های دیگر تراکنش است.

قرارداد هوشمند به‌طور محدودی در بلاکچین بیتکوین در سال ۲۰۰۹ پیاده‌سازی شد. بیتکوین زبان برنامه‌نویسی محدودی به‌نام **script** را پشتیبانی می‌کند که اجازه می‌دهد کاربران به هم‌دیگر بیتکوین بفرستند. اما این زبان تورینگ کامل نیست و اجازه پیاده‌سازی برنامه‌های دلخواه را نمی‌دهد. اما معمولا زبان‌های برنامه‌نویسی قراردادهای هوشمند توانایی اجرای برنامه‌های بیشتری را دارند.

حال تعریفی تکنیکال از قرارداد هوشمند ارایه می‌دهیم:

یک قرارداد هوشمند برنامه کامپیوتری امن و غیرقابل توقف است که نشان‌دهنده توافق بین دو یا چند گروه است که به‌طور خودکار اجرا پذیر است.

از تعریف بالا می‌فهمیم که یک قرارداد هوشمند برنامه‌ای کامپیوتری است که به زبانی نوشته شده که دستگاه هدف آن را می‌فهمد. همچنین توافقاتی را بین چند گروه شامل می‌شود. نکته دیگر این است که قراردادهای هوشمند مطابق با دستورات در کد، خودبه‌خود اجرا می‌شوند، مثلا بعد از این‌که که شرایط خاصی برقرار شده باشد. همچنین تمام شرایط قرارداد به‌همان صورتی که انتظار می‌روند اجرا می‌شوند.

قراردادهای هوشمند روی بلاکچین اجرا می‌شوند.

---

[۱] Nick Szabo

[۲] smart contract

یکی از شبکه‌های بلاکچین که مورد بحث ما است اتریوم[۳] نام دارد و زبان برنامه‌نوسی مخصوص به آن سالیدیتی[۴] است.
یک قرارداد هوشمند ویژگی‌های زیر را دارد:

- به‌طور خودکار روی بلاکچین اجرا می‌شود و نیاز به مداخله ندارد.
- تمام شرایطِ قرارداد اجرا می‌شوند و نمیتوان شرایطی که پذیرفته شده را کنار زد.
- امن است به این معنی که نمی‌توان آن را دستکاری کرد.
- قطعی است و این ویژگی این اطمینان حاصل را ایجاد می‌کند که قرارداد هوشمند همیشه یک خروجی می‌دهد.
- از نظر معنایی درست است. یعنی برنامه هم برای کاربر و هم کامپیوتر معنا دار است.
- غیر قابل توقف است. این ویژگی یعنی مهاجمان یا شرایط غیرقابل پیش‌بینی نمی‌توانند اجرای قرارداد را متوقف کنند یا آن را تغییر دهند. زمانی‌که اجرای قرارداد شروع شود، به‌طور قطعی[۵] اجرا می‌شود و اجرایش در زمانی متناهی تمم می‌شود.

قابل ذکر است که یک قرارداد هوشمند باید چهار ویژگی اول را داشته باشد و اما در شرایطی می‌تواند از دو ویژگی آخر صرف نظر کند.

۲.۶ اتریوم

ویتالیک بوترین[۶] بلاکچین اتریوم را نوامبر ۲۰۱۳ پیشنهاد کرد. ایده اصلی او طراحی زبانی تورینگ کامل بود که قراردادهای هوشمند را برای بلاکچین و برنامه‌های غیرمتمرکز[۷] اجرا کند. این مفهوم مخالف بیتکوین است که زبان **script** در بلاکچین بیتکوین تنها می‌تواند عملیات ضروری را انجام دهد و زبانی تورینگ کامل نیست.
اتریوم شبکه‌ای همتا-به-همتا است که در آن گره‌های مختلف جهت فعال نگه داشتن بلاکچین و مشارکت در مکانیزم اجماع شرکت می‌نند. این شبکه‌ها به سه دسته تقسیم می‌شوند:

- شبکه اصلی : بسیاری از پروژه‌های بلاکچین ارز دیجیتال دارای یک شبکه اصلی و اختصاصی هستند که به آن مین‌نت[۸] می‌گویند. شبکه آزمایشی هم وجود دارد که برای این است که قبل از اجرای یک برنامه روی شبکه اصلی، آن را روی شبکه آزمایشی امتحان کرد. میتوان در سایت `https://etherscan.io` که مرورگر برای شبکه بلاکچین اتریوم است، جزییات هر بلاک، تراکنش‌ها، بلوک‌ها، آدرس کیف پول‌های ارز دیجیتال، قراردادهای هوشمند و سایر داده‌های در شبکه را بررسی کرد. واضح است که تنها یک

---
[۳] ethereum
[۴] solidity
[۵] deterministic
[۶] Vitalik Buterin
[۷] Decentralized Applications - DApps
[۸] mainnet

شبکه اصلی وجود دارد.

- شبکه‌های آزمایشی : شبکه‌های آزمایشی[۹] اتریوم در واقع یک کپی از بلاکچین اتریوم هستند که تقریبا همه ویژگی‌های شبکه اصلی را دارند. تفاوت آن با شبکه اصلی در این است که ارز اتر در شبکه‌های آزمایشی بی ارزش است. همچنین زمانی‌که برنامه‌ای را روی Mainnet اجرا کنیم باید هزینه‌ای به نام gas پرداخت کنیم. با اجرای آزمایشی برنامه‌ها روی یکی از Testnet ها نیازی به پرداخت این هزینه نیست. شبکه‌های آزمایشی فعلی اتریوم Sepolia و Goerli نام دارند.
- شبکه‌های اختصاصی : همان‌طور که از اسمش معلوم است شبکه‌های اختصاصی[۱۰] شبکه‌هایی هستند که هر کس می‌تواند برای خودش بسازد و این شبکه‌ها معمولا محلی و روی دستگاه خود فرد هستند و از آنها جهت تست کردن برنامه‌ها استفاده می‌شود.

در بلاکچین اتریوم عناصر مختلفی داریم که شامل زیر هستند:

- کلیدها و آدرس‌ها
- حساب‌ها[۱۱]
- تراکنش‌ها و پیام‌ها
- رمزارز اتر[۱۲] یا توکن‌ها[۱۳]
- ماشین مجازی اتریوم[۱۴]
- قراردادهای هوشمند

کلیدها و آدرس‌ها

از کلیدها و آدرس‌ها در اتریوم برای نشان دادن مالکیت و همچنین ارسال رمز ارزِ اتر استفاده می‌شود. کلیدها از دو جفت خصوصی و عمومی تشکیل می‌شوند. کلید خصوصی به‌صورت تصادفی ساخته می‌شود و مالک آن باید آن را محرمانه نگه دارد، اما کلید عمومی از روی کلید خصوصی ساخته می‌شود. آدرس‌ها از کلیدهای عمومی ساخته می‌شوند و کدهایی ۲۰ بایتی هستند که نشان‌دهنده حساب‌ها می‌باشند.
مراحل ساخت کلید به شرح زیر می‌باشد:

- ابتدا کلید خصوصی طبق قوانین خم بیضوی[۱۵] ساخته می‌شود که عددی مثبت، تصادفی و ۲۵۶ بیتی است.
- سپس به کمک الگوریتم ECDSA کلید عمومی از کلید خصوصی ساخته می‌شود.
- کلید عمومی توسط الگوریتم Keccak هش می‌شود و از ۱۶۰ بیت سمت راست آدرسِ ساخته می‌شود.

---

[۹] testmets
[۱۰] private nets
[۱۱] accounts
[۱۲] Ether
[۱۳] Token
[۱۴] Ethereum Virtual Machine - EVM
[۱۵] elliptic curve

مثالی از کلیدها و آدرس در اتریوم مانند زیر است:

- کلید خصوصی:

b51928c22782e97cca95c490eb958b06fab7a70b9512c38c36974f47b954ffc4

- کلید عمومی:

3aa5b8eefd12bdc2d26f1ae348e5f383480877bda6f9e1a47f6a4afb35cf998ab847
f1e3948b1173622dafc6b4ac198c97b18fe1d79f90c9093ab2ff9ad99260

- آدرس:

0x77b4b5699827c5c49f73bd16fd5ce3d828c36f32

حساب‌ها

یکی دیگر از عناصر مهم در اتریوم حساب است که برای تعامل داشتن با بلاکچین مورد نیاز است. یک حساب یا نشان دهنده کاربر است و یا یک قرارداد هوشمند.
یک حساب با دو جفت کلید عمومی و خصوصی تعریف می‌شود. کاربران از حساب‌ها استفاده می‌کنند تا با بلاکچین از طریق تراکنش‌ها تعامل داشته باشند. قبل از این‌که گره‌ای یک تراکنش را به شبکه ارسال کند، باید ابتدا آن را توسط اکانتی امضا کند. اتریوم یک ماشین حالت براساس تراکنش[۱۶] است، و حالت از نتیجه تعامل بین حساب‌ها و تراکنش‌ها ساخته، یا به روز رسانی می‌شود. همه حساب‌ها یک حالت دارند که از ترکیب همه این حالت‌ها به حالتِ فعلی شبکه اتریم می‌رسیم. با ساخته شدن هر بلاک جدید، حالت شبکه اتریوم به روز رسانی می‌شود.
دو نوع حساب وجود دارد:

- حساب‌های با مالکیت خارجی[۱۷]
- حساب‌های قرارداد[۱۸]

EOA ها شبیه حساب‌هایی در بیتکوین هستند که توسط یک کلید خصوصی کنترل می‌شوند.
CA ها اکانت‌هایی اند که یک کدِ برنامه نویسی و یک کلید خصوصی دارند.
ویژگی‌های هر اکانت را می‌توان در زیر مشاهده کرد:
EOAs

- دارای حالت هستند. (مانند حالت در یک ماشین حالت)
- مرتبط به یک کاربر انسانی هستند برای همین به آن‌ها حسابِ کاربری نیز می‌گوییم.
- هر EOA میزانی پول دارد که به آن balance می‌گوییم.
- توانایی ارسال تراکنش دارند.
- هیچ کدی مرتبط با آن‌ها نیست.
- با کلیدهای خصوصی کنترل می‌شوند.
- EOA ها نمی‌توانند اطلاعاتی را از روی بلاکچین بخوانند.
- EOA ها می‌توانند تراکنش‌ای را صدا بزنند.

---

[۱۶] transaction-driven state machine
[۱۷] Ecternally Owned Accounts - EOAs
[۱۸] Contract Accounts - CAs

CAs

- دارای حالت هستند.
- به‌طور خاص مرتبط با یک کاربر در شبکه بلاکچین نیستند.
- CA ها کیف پولی شامل مقداری رمز ارز اتر دارند.
- کدِ مخصوص به خود را دارند که در حافظه بلاکچین ذخیره می‌شود. همچنین آن‌ها دسترَسی به حافظه دارند.
- می‌توانند در یک تراکنش صدا زده و اجرا شوند. با توجه به تورینگ کامل بودنِ بلاکچین اتریوم، کدِ مربوط به CA می‌تواند هر درجه از پیچیدگی را داشته باشد. این کَد توسط EVM به‌کَمک گره‌ای روی شبکه اتریوم اجرا می‌شود. در ادامه بیشتر درباره EVM بحث خواهیم کرد.
- CA ها نمیتوانند یک تراکنش را شروع کنند.
- CA ها می‌توانند از روی بلاکچین اطلاعاتی را بخوانند.
- هنگامی که CA ها اجرا می‌شوند، یک آدرس به آن اختصاص داده می‌شود که برای این است که بتوان محل آن روی بلاکچین را تشخیص داد.

تراکنش‌ها و پیام‌ها

تراکنش در یک اتریوم داده‌ای است که با کلید خصوصی امضا شده و شامل دستوراتی است که زمانی که اجرایشان کامل شود، یا باعث یک پیام فراخوانی (خواندن اطلاعاتی از روی بلاکچین) یا ساخته شدن یک قرارداد می‌شوند. تراکنش‌ها براساس خروجی‌شان به دو دسته تقسیم می‌شوند:

پیام فراخوانی:[۱۹] در این تراکنش صرفا یک فراخوانی انجام می‌شود و داده‌ای خوانده می‌شود. بعد از اجرای این تراکنش هیچ تغییری در حالت بلاکچین ایجاد نمی‌شود.

- ساخت یک قرارداد:[۲۰] این تراکنش‌ها باعث ساخته شدن یک قراردادِ جدید (CA) می‌شوند. پس اگر تراکنش با موفقیت اجرا شود، تغییری در حالتِ بلاکچین اَیجاد می‌شود.

هردوی این تراکنش‌ا از چند قسمت اصلی تشکیل شده اند که در زیر توضیح می‌دهیم:

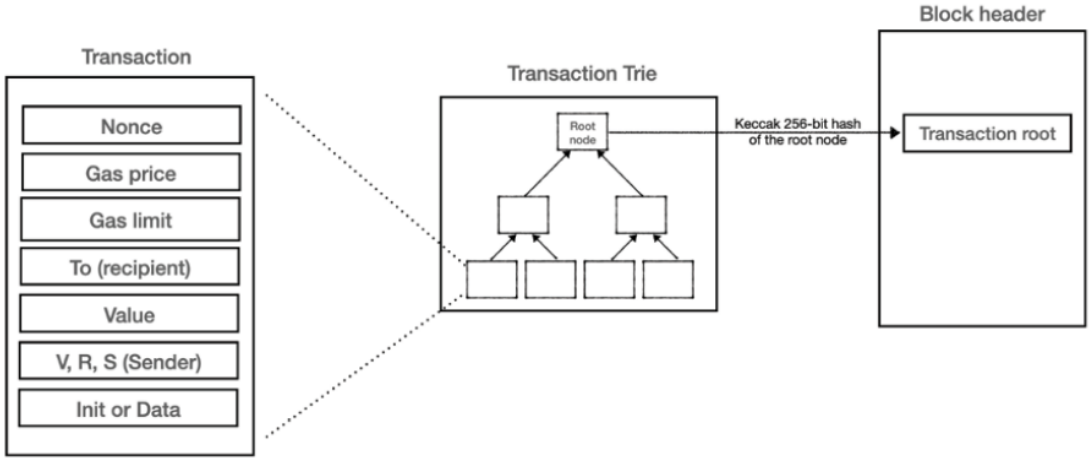
- nonce: نانس عددی است که وقتی یک تراکنش توسط کاربر فرستاده می‌شود یک واحد افزایش میابد. این عدد باید برابر با تعداد تراکنش‌هایی باشد که فرستاده شده. این عدد را نباید باد عدد نانس‌ای که ماینرها دنبالش هستند اشتباه گرفت.
- gas : این عدد میزان wei[۲۱] ای که مورد نیاز است تا قرارداد اجرا شود را نشان می‌دهد. این مقدار باید به‌خاطر هزینه محاسباتی که برای اجرای قرارداد صرف می‌شود پرداخت شود.
- gas limit: به حداکثر مقدار گاز اشاره دارد که کاربر مایل به مصرف آن برای انجام یک تراکنش خاص است.
- To: همان‌طور که از اسمش معلوم است، To مقداری است که اشاره به آدرس گیرنده

---

[۱۹] message call transaction

[۲۰] contract creation transaction

[۲۱] $10^{18} wei = 1 ether$



تراکنش دارد. این مقدار ۲۰ بایت است.

- **value**: اگر در قرارداد مقداری ارز جابه‌جا می‌شود، این مقدار در قسمت **value** قرار داده می‌شود.
- **signature**: این قسمت شامل سه مقدار **R V**، و **S** است که نشان دهنده امضای دیجیتال (R , S) و همچنین اطلاعات لازم برای بازیابی کلید عمومی (V) است. فرستنده تراکنش را می‌توان به‌کمک این مقادیر به‌دست آورد. برای اطلاعات بیشتر باید درباره Elliptic Curve Cryptpgraphy و منحنی secp256k1 مطالعه کنید.
- **Init** از این مقدار در تراکنش‌هایی که یک قرارداد جدید درست می‌کنند استفاده می‌شود و شامل ارایه‌ای با طول نامحدود است که نشان می‌دهد **EVM** باید چه کدی را اجرا کند. از **Init** تنها یک بار استفاده می‌شود و پس از درست شدن قرارداد از بین می‌رود.
- **Data**: اگر تراکنش از نوع فراخوانی است، به‌جای **Init** از **Data** استفاده می‌شود که شامل داده ورودیِ پیام فراخوانی است. همچنین اندازه این مقدار نامتناهی است.

این ساختار را می‌توان در شکل دید که تراکنش از قسمت‌هایی که گفتیم تشکیل شده و از ترکیب آن‌ها به یک خانه در درخت تراکنش[۲۲] می‌رسیم. درنهایت ریشه درخت از هشِ تراکنش‌ها به‌کمک الگوریتم **Keccak** به‌دست می‌آید و در سرتیترِ بلاکی که شامل این تراکنش‌ها است قرار می‌گیرد.

رمز ارز اتر و توکن

به‌عنوان انگیزه برای ماینرها، اتریوم ارزِ خودش به‌نام اتر را دارد. پس ارزِ مخصوصِ بلاکچین اتریوم، اتر نام دارد. همچنین مفهوم دیگری به‌نام توکن[۲۳] داریم که ارزی است که توسط شخص سومی (مانند یکی از کاربران شبکه اتریوم) ساخته شده و از بلاکچین اتریوم استفاده می‌کند. هر کسی می‌تواند توکن مخصوص خودش را روی بلاکچین اتریوم بسازد.

---

[۲۲] transaction trie
[۲۳] token

همانند بیتکوین، اتریوم هم به ماینرها در ازای قدرت محاسباتی‌ای که در اختیار شبکه می‌گذارند داده می‌شود. واحدهایی که بیشتر در اتریوم استفاده می‌شوند Eth، Gwei و Wei هستند.

ماشین مجازی اتریوم

ماشین مجازی اتریوم دستگاهی براساس پشته[۲۴] است که بایت‌کد دستورات را اجرا می‌کند و حالت ماشین را عوض می‌کند. اندازه هر حرف در EVM ۲۵۶ بیت است و پشته هم ۱۰۲۴ حرف فضا دارد. ماشین مجازی اتریوم تورینگ کامل است اما با میزانی گازی که نیاز دارد تا هر دستور را اجرا کند محدود می‌شود. این یعنی حلقه‌های بی‌نهایت اتفاق نمی‌افتند. EVM توانایی رسیدگی به استثنا[۲۵] را نیز دارد، مانند نبود گاز کافی یا دستوراتِ نامعلوم که در این مواقع ماشین متوقف می‌شود و خطا برمی‌گرداند.

EVM یک سیستم بسته است و کدی که روی EVM اجرا می‌شود به منبع خارجی دسترسی ندارد. این موضوع باعث افزایش امنیت و قطعی بودن[۲۶] سیستم می‌شود.

ماشین مجازی نرم افزاری است که در سطح بالاتر از سیستم عامل هایی مانند ویندوز قرار دارند. ماشین مجازی میتواند قدرت محاسباتی یک کامپیوتر فیزیکی را بصورت مجازی بیشتر کند. بکمک یک ماشین مجازی میتوان از قدرت پردازش سیستم‌های عضو شبکه برای بلاکچین استفاده کرد. یعنی ماشین مجازی اتریوم کامپیوتری است که قدرت پردازش خود را از نقاط مختلف جهان میگیرد و میتواند این قدرت پردازش را در اختیار پروژه های مختلف مانند قراردادهای هوشمند قرار دهد. گره‌های بلاکچین اتریوم میتوانند از نقاط مختلف جهان به ماشین مجازی دسترسی داشته باشند و قدرت پردازشی خود را در اختیار این شبکه بگذارند.

ماشین مجازی اتریوم سه حافظه اصلی دارد:

- Memory: این حافظه برای اجرای برنامه لازم است. زمانی که اجرای برنامه تمام شد، اطلاعات داخل memory پاک می‌شود. حافظه memory بینهایت است اما براساس مقدار گاز برای هر برنامه محدود می‌شود.
- Storage: این حافظه برای همیشه روی بلاکچین باقی می‌ماند و زمانی‌که برنامه تغییراتی روی این حافظه ایجاد می‌کند، دیگر نمی‌توان آن را تغییر داد.
- Stack: EVM محاسبات را طبق ترتیبی که در پشته‌اش قرار گرفته انجام می‌دهد. پشته می‌تواند ۱۰۲۴ عنصر که هر کدام ۲۵۶ بیت هستند را ذخیره کند.

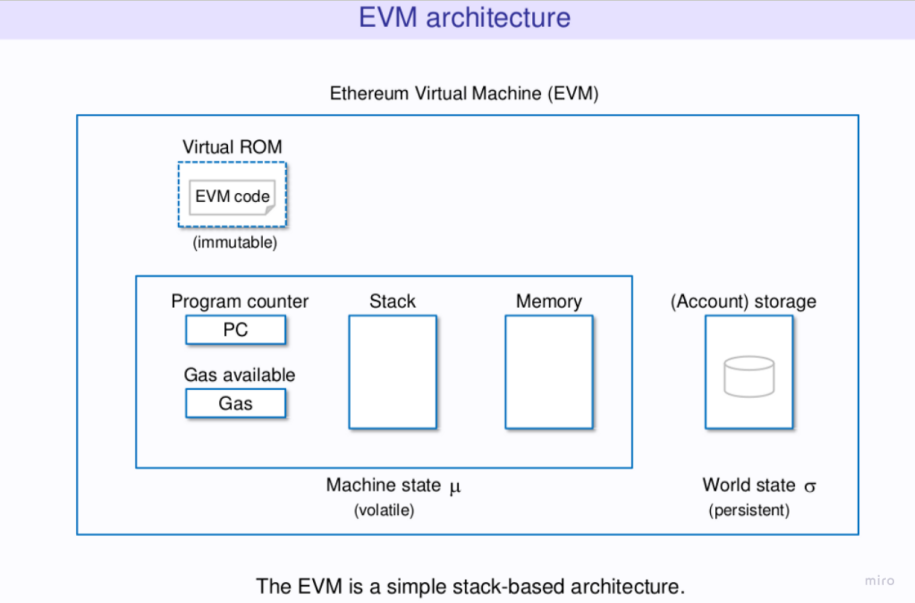
در شکل می‌توانید ساختار ساختار ماشین مجازی اتریوم را دید. stack و memory حافظه های موقت هستند اما storage دایمی هست.

قراردادهای هوشمند

در بخش قبل درمورد قراردادهای هوشمند صحبت کردیم. در اتریوم می‌توان قراردادهای هوشمند را پیاده‌سازی کرد و زبان برنامه‌نویسی بلاکچین اتریوم سالیدیتی[۲۷] نام دارد.

---

[۲۴] stack
[۲۵] exception handling
[۲۶] determinisitc
[۲۷] solidity



در زیر می‌توان نمونه‌ای از یک قرارداد هوشمند به زبان سالیدیتی را مشاهده کرد که در آن لیستی از افراد داریم که برای هر کس نام و عدد متنظارش را ذخیره می‌کنیم.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
contract SimpleStorage {
    uint256 favoriteNumber;
    struct People {
        uint256 favoriteNumber;
        string name;
    }
    People[] public people;
    mapping(string => uint256) public nameToFavoriteNumber;
    function store(uint256 _favoriteNumber) public {
        favoriteNumber = _favoriteNumber;
    }
    function retrieve() public view returns (uint256){
        return favoriteNumber;
    }
    function addPerson(string memory _name, uint256
       _favoriteNumber) public {
        people.push(People(_favoriteNumber, _name));
        nameToFavoriteNumber[_name] = _favoriteNumber;
    }
}
```

در برنامه بالا ابتدا ورژن زبان سالیدیتی را با **pragma** مشخص کنیم. سپس بینید به قرارداد یک اسم می‌دهیم. در این برنامه ساده لیستی از افراد داریم که در آن اسم و شماره هر فرد ذخیره

می‌شود.
در این برنامه دو نوع تابع می‌بینیم یکی تابع **retrieve** که از نوع **call** است و صرفا چیزی را از بلاکچین می‌خواند ولی تغییری در حالت شبکه ایجاد نمی‌کند. اجرای این نوع توابع هزینه گاز ندارد. اما دو تابع دیگر حالت بلاکچین را تغییر می‌دهند پس اگر از این قرارداد بخواهیم یکی از دو تابع **addPerson** یا **store** را صدا بزنیم، باید هزینه‌ای هم پرداخت کنیم.

۱.۲.۶ قرارداد لاتاری

در ادامه یک قرارداد لاتاری را به زبان سالیدیتی توضیح می‌دهیم.
اجرای لاتاری سه بخش کلی دارد:

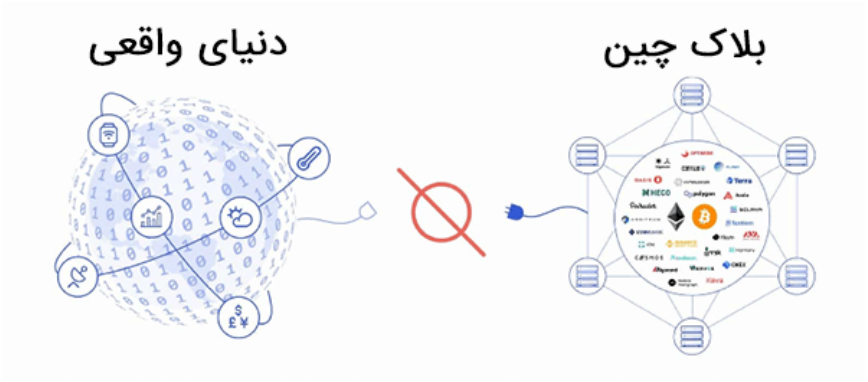
- مالک قرارداد بازه زمانی‌ای را برای شرکت در لاتاری تعیین می‌کند.
- بازیکنان با پرداخت اتر وارد بازی می‌شوند. باید میزان اتر پرداختی حتما بیشتر از ۵۰ دلار باشد.
- پس از تمام شدن زمان شرکت در لاتاری، قرارداد یک نفر را به‌طور تصادفی به‌عنوان برنده انتخاب می‌کند و پول‌هایی که در حساب قرارداد است را به آدرس آن شخص منتقل می‌کند.

کلیت این قرارداد به‌صورت زیر است:

```
1  contract Lottery{
2      function enter() public payable{} //user to enter
3      function getEntranceFee() public{
4  //to enter the lottery there is a minimum amount of 50 USD to
       pay
5       }
6     function startLottery() public{} //contract owner to start
7     function endLottery() public{} //contract owner to end
8  }
```

در این برنامه ۳ تابع اصلی داریم. دو تابع **startLotter** و و **endLottery** بازه شروع و اتمام ورود به لاتاری را مشخص می‌کنند و باید به‌صورتی تعریف شده باشند که تنها مالک قرارداد بتواند آن ها را صدا کند. همچنین بعد از صدا کردن **endLottery** باید برنده مشخص شود و پول‌هایی که در حساب قرارداد هست به حساب برنده ریخته شود.
تابع **enter** جهت وارد شدن کاربران به لاتاری است. برای ورود یک کاربر باید دو شرط برآورده شود. یکی این‌که کاربر در بازه زمانی‌ای که لاتاری باز است می‌تواند وارد شود و دومی این‌که هنگام صدا زدن تابع باید مقداری اتریوم که معادلش به دلار آمریکا از ۵۰ دلار بیشتر باشد پرداخت کند.
از تابع **getEntranceFee** برای بررسی شرط دوم استفاده می‌شود.
هدف این قرارداد آشنایی با مفاهیم **Oracle** و اعداد تصادفی در بلاکچین است.

ابتدا گریزی به مفهوم oracle می‌زنیم و دوباره به برنامه برمی‌گردیم.
قراردادهای هوشمند در بلاکچین توانایی تعامل با داده‌ها و سیستم‌هایی که خارج از محیط بلاکچین هستند را ندارند. منابع خارج از بلاکچین به‌صورت آف‌چین یا بیرون زنجیره‌ای به‌حساب می‌آیند. قطع ارتباط با دنیای خارج از بلاکچین تصمیمی عمدی است که موجب اجماع، جلوگیری از حملات و کاهش هزینه‌های مازاد می‌شود. اراکل خودش یک بلاکچین است که بلاکچین‌های دیگر را به سیستم‌های خارجی متصل می‌کند و در نتیجه قراردادهای هوشمند می‌تواند براساس داده‌های دنیای واقعی ورودی بگیرند.



اراکل‌های متمرکز که از یک موجودیت برای تحویل داده به قراردادهای هوشمند استفاده می‌کنند، هدف استفاده از بلاکچین که غیرمتمرکز بودن است را از بین می‌برد. درصورت آفلاین شدن این اراکل، قرارداد هوشمند به داده مورد نیاز خود دسترسی ندارد. همچنین ممکن است در صورت فساد، این اراکل داده‌ای نادرست تحویل دهد.

برای حل این مشکل باید از اراکل غیرمتمرکز[۲۸] استفاده کرد. DON چندین گره اراکل مستقل را با چندین منبع داده مطمین ترکیب می‌کند و از هر کدام از این اراکل‌ها جوابی می‌گیرد. سپس این جواب‌ها را به‌صورتی با هم ترکیب کرده (به‌عنوان مثال اگر داده عددی بود میانگین می‌گیرد)

[۲۸] Decentralized Oracle Network - DON

و به‌عنوان جواب نهایی به قراردادِ هوشمندی که درخواست داده کرده بود تحویل می‌دهد. اراکل‌ای که ما در مساله از آن استفاده می‌کنیم ChainLink نام دارد.
توجه به این نکته ضروری است که یک اراکل خود منبع داده نیست، بلکه لایه‌ای است که منابع داده خارجی را جستجو، تأیید و سپس آن اطلاعات را انتقال می‌دهد. در زیر می‌توانید اراکل ChainLink و سه مرحله انتقال داده از منابع داده‌ای خارج از بلاکچین تا رسیدن به قرارداد را دید. در نهایت پس از تایید اطلاعات، میانگینی از جواب‌های چند اراکل به‌عنوان جواب نهایی برگردانده می‌شود.

حالا می‌توانیم به قرارداد لاتاری برگردیم. در لاتاری برای انجام دو کار نیاز به اراکل داریم. همان‌طور که گفتیم در تابع enter باید چک شود کاربری که می‌خوهد وارد لاتاری شود حداقل ۵۰ دلار پرداخت کرده باشد. کاربران با اتر پرداخت می‌کنند پس باید بررسی کرد که چه‌مقدار اتر برابر با ۵۰ دلار است. این‌کار را در تابع getEntraceFee انجام می‌دهیم. باید ارزش دلار بر اساس اتریوم را از اراکل ChainLink بگیریم و سپس ۵۰ دلار را به اتر معادلش تبدیل کنیم. نحوه گرفتن نرخ دلار به اتر در سایت chain.link وجود دارد.
مورد دوم استفاده اراکل هنگام انتخاب برنده است. زمانی‌که تابع endLottery را صدا می‌زنیم، باید برنده لاتاری انتخاب شود. برای انجام این‌کار عددی تصادفی تولید می‌کنیم و به‌کمک آن برنده را مشخص می‌کنیم.
از آن‌جا که بلاکچین یک سیستم قطعی[۲۹] است، ساخت عدد تصادفی در آن غیرممکن است. یکی از راه‌های ساخت عدد تصادفی این است که عددی را بر پایه یکی از ویژگی‌های سیستم انتخاب کنیم (مثلا ساعت سیستم) اما در این‌صورت عددمان واقعا تصادفی نیست و امکان حدس زدن آن عدد و تقلب در لاتاری وجود دارد. راهی دیگر ساختن اعداد شبه‌تصادفی[۳۰] است. در این روش معمولا چند متغیر Global انتخاب شده و هش می‌شوند. مثلا می‌توانیم از متغیرهای در برنامه زیر استفاده کرد و آن‌ها را با الگوریتم keccack256 هش کرد:

[۲۹] deterministic
[۳۰] pseudorandom numbers

```
function endLottery() public onlyOwner{ //admin to end
        uint256(
            keccack256(
                abi.encodePacked(
                    nonce, // nonce is preditable (aka,
                        transaction number)
                    msg.sender, // msg.sender is predictable
                    block.difficulty, // can actually be
                        manipulated by the miners!
                    block.timestamp // timestamp is predictable
                )
            )
        ) % players.length;
    }
```

این روش هم مانند روش قبلی ایمن نیست. **nonce** در این‌جا شماره تراکنش است که عددی قابل پیش‌بینی است. **msg.sender** هم کسی است که تابع را صدا می‌زند که در این مثال مالک قرارداد است پس مهاجمان می‌توانند آن‌را هم پیش‌بینی کنند. **block.difiiculty** درچه سختی ماین کردن بلاکی که تراکنش در آن قرار داد است که ماینرها می‌توانند آن را دستکاری کنند. **block.timestamp** هم قابل پیش‌بینی است. یکی از دلایل اصلی قابل پیش‌بینی بودن این متغیرها این است که هرکسی می‌تواند اطلاعاتی که روی بلاکچین نوشته شده را بخواند و چیزی پنهان نیست.

برای حل این مشکل از برنامه‌ای به‌نام **ChainlinkVRF** استفاده می‌کنیم. در این برنامه از **ChainLink** درخواست عدد تصادفی می‌کنیم. سپس **ChainLink** از چند منبع داده عدد تصادفی درخواست می‌کند و جواب‌هایی پس می‌گیرد. ابتدا با الگوریتمی بررسی می‌کند که این اعداد واقعا تصادفی هستند و پس از تایید، این اعداد را ترکیب کرده و به‌عنوان خروجی نهایی به قرارداد هوشمند برمی‌گرداند.

ادامه کد ساده است. صرف باید توابع **startLottery** و **enter** را کامل کنیم.

```
// SPDX-License-Identifier: MIIT
pragma solidity ^0.8.0;
import "@chainlink/contracts/src/v0.8/interfaces/
    AggregatorV3Interface.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import '@chainlink/contracts/src/v0.8/VRFConsumerBase.sol';

contract Lottery is VRFConsumerBase,  Ownable{

    //users are players:
    address payable[] public players;
    address payable public recentWinner;
    uint256 public randomness;

    //in this we will save how much wei would be 50 usd:
    uint256 public usdEntryFee;

    AggregatorV3Interface internal ethUsdPriceFeed;

    enum LOTTERY_STATE{
        OPEN,
        CLOSED,
        CALCULATING_WINNER
    }

    LOTTERY_STATE public lottery_state;

    uint256 public fee;
    bytes32 public keyhash;
    constructor (
        address _priceFeedAddress,
        address _vrfCoordinator,
        address _link,
        uint256 _fee,
        bytes32 _keyhash
    )public   VRFConsumerBase(_vrfCoordinator, _link){
        usdEntryFee = 50 * (10**18); //50 minimum entry.
        ethUsdPriceFeed = AggregatorV3Interface(
            _priceFeedAddress);
        lottery_state = LOTTERY_STATE.CLOSED;
        fee = _fee;
        keyhash = _keyhash;
    }

    function enter() public payable{ //for user to enter
        require(lottery_state == LOTTERY_STATE.OPEN);
        require(msg.value >= getEntranceFee(),"Not enough ETH");
        players.push(payable(msg.sender));
    }
```

```
48
49      //returns how much wei is 50 usd:
50      function getEntranceFee() public view returns(uint256){
51          (,int256 price,,,) = ethUsdPriceFeed.latestRoundData();
                //conversion rate
52          uint256 adjustedPrice = uint256(price) *10**10; //*
                usdEntryFee;
53          uint256 costToEnter = (usdEntryFee * 10**18)/
                adjustedPrice;
54          return costToEnter;
55      }
56
57      function startLottery() public onlyOwner{ //admin to start
58          require(lottery_state == LOTTERY_STATE.CLOSED);
59          lottery_state = LOTTERY_STATE.OPEN;
60      }
61
62      function endLottery() public onlyOwner{ //admin to end
63          lottery_state = LOTTERY_STATE.CALCULATING_WINNER;
64
65          //in the first transaction we will end the lottery
66          // request a random number
67
68          bytes32 requestId = requestRandomness(keyhash , fee);
69
70          //in a second transaction, after a cainlink node
71          //has created a provably random number, it will call
72          //a second transaction itself and its name has to be
73          //fulfillrandomness
74
75      }
76
77      function fulfillRandomness(bytes32 _requestId, uint
            _randomness) internal override {
78          require(lottery_state == LOTTERY_STATE.
                CALCULATING_WINNER,
79           "you arent there yet"
80           );
81          require(_randomness > 0, "random not found");
82
83          //picking a random winner:
84          // _randomness is the random number we get
85          uint256 indexOfWinner = _randomness % players.length;
86          recentWinner = players[indexOfWinner];
87
88          //now we should pay the winner
89          recentWinner.transfer(address(this).balance);
90
91          //reseting the lottery:
```

```
92          players = new address payable[](0);
93          lottery_state = LOTTERY_STATE.CLOSED;
94          randomness = _randomness; //keeping te latest random
                number
95      }
96  }
```

فصل ۷

طراحی توکن در بلاکچین

طراحی توکن در بستر بلاکچین به‌معنی فرایند نمایش یک دارایی دیجیتالی در بلاکچین است. از آن می‌توان برای نمایش کالاها، اموال، مالکیت اثری هنری، ارز یا هر چیز ارزش‌مند دیگری استفاده کرد.
خاطر نشان می‌گردد که کوین و توکن تفاوت‌هایی دارند. کوین رمزارزِ پیش‌فرضِ بلاکچین‌ای است که روی آن اجرا می‌شود. مثال‌های معمول از این کوین‌ها شامل اتر که برای بلاکچین اتریوم و بیتکوین که برای بلاکچین بیتکوین است می‌باشند. این رمزارزها خود نیز نوعی توکن هستند. اما توکن نشان دهنده یک دارایی است که روی بلاکچین اجرا می‌شود. به‌عنوان مثال اتریوم نه‌تنها توکنِ اصلی خودش یعنی اتر را دارد، بلکه هزاران توکن دیگر روی شبکه اتریوم ساخته شده‌اند. به‌کمک قراردادهای هوشمند می‌توان این توکن‌ها را ساخت. دو نوع توکن قابل تعویض[۱] و غیرقابل تعویض[۲] داریم.

دو توکن را قابل تعویض گوییم اگر از یک نوع باشند، به‌عنوان مثال اسکناس‌هایی که روزانه استفاده می‌کنیم قابل تعویض است، به این‌صورت که یک اسکناس هزار تومانی با یک اسکناس هزار تومانی دیگر هیچ تفاوتی ندارد. توکن‌های قابل تعویض نیز این‌گونه‌اند و قابل ویژگی‌های زیر هستند :

- غیر قابل تشخیص : توکن‌های از یک نوع را نمی‌توان از هم تشخیص داد یعنی باهم یکسان هستند.
- قابل تعویض: یک توکن با توکن دیگری که ارزش یکسان دارد قابل تعویض است.
- قابل تقسیم: توکن‌ها را می‌توان به قسمت‌های کوچک‌تر تقسیم کرد. در مثال بالا می‌توان اسکناس هزار تومانی را به دو اسکناس پانصد تومانی تقسیم کرد.

به توکن‌های غیرقابل تعویض NFT[۳] می‌گوییم. در تعریف بالا دیدیم که تعویض‌ذیری این قابلیت را می‌دهد که بتوانیم دو توکن با ارزش یکسان را با هم عوض کنیم، اما NFT ها این قابلیت را

---

[۱] Fungible
[۲] non-fungible
[۳] Non-Fungible Token

ندارند. به‌عنوان مثال یک نقاشیِ هنری یک دارایی غیرقابل تعویض است زیرا منحصر به فرد بوده و نمی‌توان آن را با نقاشی دیگری جایگزین کرد، حتی اگر هر دو قیمت یکسان داشته باشند. پس هر NFT منحصر به فرد است و می‌توان آن را از بقیه تشخیص داد. ویژگی‌های NFT ها به شرح زیر اند:

- منحصر به فرد بودن: NFT ها توکن‌های منحصر به فردی هستند و با توکن‌های دیگر یا آن‌هایی که از جنس یکسان اند تفاوت دارند.
- غیر قابل تعویض: از آن‌جا که منحصر به فرد هستند پس ویژگی‌های خاصی را نمایش می‌دهند. این توکن‌ها با توکن‌های دیگر یا حتی از همان جنس قابل تعویض نیستند.
- تقسیم ناپذیر: این توکن‌ها را فقط می‌توان به‌صورت یکجا و کامل دراختیار داشت و نمی‌توان یک NFT را به چند NFT دیگر تقسیم کرد.

در ادامه استانداردهایی را برای طراحی توکن ارایه می‌دهیم.

۱.۷ استاندارد طراحی توکن

با ظهور بسترهایی برای اجرای قراردادهای هوشمند مانند اتریوم، ساختن توکن بسیار ساده شده است. توکن یک رمزارز است که می‌تواند با چند خط کد روی اتریوم مانند مثال زیر ساخته شود:

```
1  pragma solidity ^0.5.0;
2  contract token {
3      mapping (address => uint) public coinBalanceOf;
4      event CoinTransfer(address sender, address receiver, uint
           amount);
5      /* Initializes contract with initial supply tokens to the
           creator of contract */
6      function tokenx(uint supply) public {
7          supply = 1000;
8          coinBalanceOf[msg.sender] = supply;
9      }
10     /* Very simple trade function */
11     function sendCoin(address receiver, uint amount) public
           returns(boolean){
12         if (coinBalanceOf[msg.sender] < amount) return false;
13         coinBalanceOf[msg.sender] -= amount;
14         coinBalanceOf[receiver] += amount;
15         emit CoinTransfer(msg.sender, receiver, amount);
16         return true;
17     }
18 }
```

این کد به درستی کار می‌کند و می‌تواند توکن‌هایی را درست کند. اما مشکل این است که بدون هیچ مکانیزم استانداردسازی، هرکسی می‌تواند با روش خودش توکنی ایجاد کند که باعث ایجد مشکلاتی در قابلیت استفاده و تعاملِ توکن‌ها با هم می‌شود.

یکی از دلایل اصلی وجود استاندارد طراحی توکن این است که انتقال و تبادل رمزارز ها در کیف پول‌های مجازی و صرافی‌های مجازی یکسان باشد. اگر توکن‌ها با یک استاندارد طراحی شوند، تنها لازم است یک برنامه جهت تبادل توکن‌ها نوشته شود. اولین استاندارد طراحی توکن در اتریوم ERC-20 نام دارد. ERC-20 معروف ترین استاندارد طراحی توکن در اتریوم است و بسیاری از استانداردهای دیگر بر مبنا ERC-20 هستند.
در ادامه می‌خواهیم توکنی با استاندارد ERC-721 بسازیم که این استاندارد برمبنای ERC-20 ارایه شده است.

ERC-20

این استاندارد که در سال ۲۰۱۵ توسط ویتالیک بوترین نوشته شده، یک فرمت نسبتا ساده جهت طراحی توکن‌های شبکه اتریوم است. با پیروی از این طرح کلی، هم توکن‌های ساخته شده از نظر طراحی و تعامل شبیه هم می‌شوند و هم برنامه‌نویسان لازم نیست همه چیز را از ابتدا بنویسند.

توکنی که استاندارد ERC-۲۰ را رعایت کرده، قرارداد هوشمندی است که شامل بخش‌های زیر است. دقت شود که در قرارداد هوشمند باید از همین اسم‌ها استفاده کرد. به‌عنوان مثال حتما باید در این قرارداد تابعی با نام balanceOf وجود داشته باشد که از نوع view است و خروجی‌اش یک uint256 است. بقیه توابع هم به‌همین صورت هستند:

Methods:

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
function transfer(address _to, uint256 _value) public returns (bool success)
function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
function approve(address _spender, uint256 _value) public returns (bool success)
function allowance(address _owner, address _spender) public view returns (uint256 remaining)
```

Events:

```
event Transfer(address indexed _from, address indexed _to, uint256 _value)
event Approval(address indexed _owner, address indexed _spender, uint256 _value)
```

در گیت‌هاب OpenZeppelin پیاده‌سازی‌ای از یک توکن ERC۲۰ وجود دارد. کاری که ما لازم است انجام بدهیم این است که صرفا این پیاده‌سازی را ایمپورت کرده و به مقادیرِ مورد نیاز یعنی نام و سمبول توکن مقدار دهیم:

OurToken.sol smart contract:

```sol
// contracts/OurToken.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract OurToken is ERC20 {
    // wei
    constructor(uint256 initialSupply) ERC20("OurToken", "OT") {
        _mint(msg.sender, initialSupply);
    }
}
```

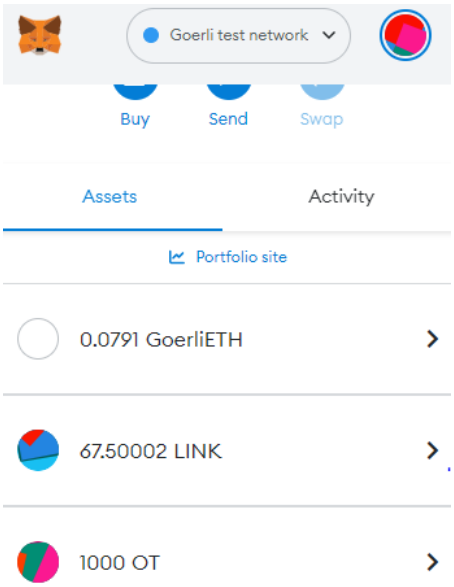
هنگام قراردادن این smart contract روی تست نتِ Goerli لازم است به متغیر initial-Supply آن مقدار دهیم.
لازم به‌ذکر است که برای deploy کردن برنامه‌ها روی شبکه‌های بلاکچین از پایتون و پکیج brownie استفاده شده است.

DeployToken.py:

```python
from brownie import OurToken
from scripts.utils import get_account
from web3 import Web3

initial_supply = Web3.toWei(1000,"ether")
# initial supply is 1000 ether

def main():
    account = get_account()
    our_token = OurToken.deploy(initial_supply, {"from":account
        })
    print(our_token.name())
```

پس از deploy کردن برنامه می‌توان آدرس قرارداد را دید:

آدرس قرارداد: 0xDb57B22b7B21a389018732B9976010C043D2b86a
این آدرس را می‌توان در goerli etherscan جستجو کرد و جزییات آن را دید. اینجا لینک etherscan قرارداد است.
همچنین این توکن به آدرسی که قرارداد را اجرا کرده واریز می‌شود. در کیف پول metamask باید با آدرس قرارداد آن‌را تعریف کنیم و پس از اضافه کردن این توکن به کیف پولمان می‌توانیم ببینیم که واقعا ۱۰۰۰ تا از این توکن اضافه شده است.



ERC-721

توکن‌های ERC20 قابل تعویض یا **fungible** هستند. در مثال بالا دیدیم که برای خود ۱۰۰۰ توکن به‌نام **OT** طراحی کردیم که همه‌شان یکسان اند. اما توکن‌های ERc721 غیرقابل تعویض یا **Non-Fungible** هستند و از این رو به آن‌ها **NFT** [۴] گوییم.
تفاوت در پیاده‌سازی این دو استاندارد این است که در قراردادهای ERC20 بین هر آدرس و میزان توکنی که در اختیار دارد یک **mapping** داریم. یعنی مشخص می‌شود که هر آدرس چه تعداد از این توکن را در اختیار دارد.

```
1 mapping (address => uint256) private _balances;
2 // mapping from address of account to how much of that token
     they have
```

اما در قراردادهای ERC721 چون هر توکن با توکن دیگری تفاوت دارد و این تفاوت را با عددی به‌نام **tokenID** مشخص می‌کنیم، یک **mapping** از **tokenID** به آدرس داریم که نشان می‌دهد مالک توکن با یک **tokenID** خاص، چه آدرسی است.

```
1 mapping (uint256 => address) private _owners;
2 // mapping from token ID of that NFT (token ID represents the
     unique NFT)
3 // to address of owner of the NFT
```

NFT یک دارایی را نشان می‌دهد که ویژگی‌های خاص خودش را دارد. باید راهی داشته باشیم که این ویژگی‌ها را نشان دهیم. مثلا این توکن می‌تواند یک شخصیت در یک بازی ویدیویی باشد

[۴] Non-Fungible Token

و باید راهی داشته باشیم که ویژگی‌های این شخصیت را نشان دهیم. یا اگر یک نقاشی است باید نشان داد که شکل نقاشی چیست. برای این‌کار از **metadata** و **tokenURI** استفاده می‌کنیم.
یک **metadata** چیزی شبیه به زیر است که اسم، توضیحات توکن، محل تصویر(لینکی که به تصویر اشاره می‌کند) و بقیه ویژگی‌هایی که میخواهیم در توکن بنویسیم را دربرمی‌گیرد.

metadata:

```
{
“name” : “Name”,
“description” : “Description”,
“image” : “URI”, //points to image location
“attributes” : [ ] //can have anything here
}
```

tokenURI() تابعی است که به محلی که **metadata** توکن ذخیره شده، اشاره می‌کند. فایل **metadata** از جنس **json**. است. نمی‌توان **metadata** را در سیستمی متمرکز ذخیره کرد پس از **IPFS**[۵] استفاده می‌کنیم.
IPFS یک پروتکل و شبکه همتا-به-همتا برای ذخیره اطلاعات و به اشتراک‌گذاری داده‌ها در یک سیستم فایل توزیع‌شده[۶] است. **IPFS** از آدرس‌دهی محتوا[۷] برای شناسایی منحصر به فرد هر فایل در یک فضای نام جهانی[۸] که همه دستگاه‌های محاسباتی را به هم متصل می‌کند، استفاده می‌کند.
دلیل این‌که **metadata** را روی بلاکچینی که توکن‌ها هستند ذخیره نمی‌کنیم این است که هزینه ذخیره اطلاعات در بلاکچین بسیار زیاد است. پس **metadata** را روی سیستمی توزیع‌شده به‌نام **IPFS** قرار می‌دهیم و تنها آدرسی که به **metadata** اشاره می‌کند را روی بلاکچین قرار می‌دهیم. به این آدرس **URI** می‌گوییم.
پس به‌طور خلاصه هر **NFT** ویژگی‌هایی دارد که آن ها را در فایلی از جنس **json**. به نام **meta-data** ذخیره می‌کنیم. به‌خاطر هزینه ذخیره و نگه‌داری اطلاعات در بلاکچین، فایل **metadata** را در سیستم فایل توزیع‌شده‌ای به‌نام **IPFS** قرار می‌دهیم و در بلاکچین صرفا **URI** را ذخیره می‌کنیم که به آدرس **metadata** اشاره دارد.

برای پیاده‌سازی، همانند ERC-20 در **OpenZeppeline** استاندارد ERC-721 که در اینجا پیاده‌سازی شده را در قراردادمان وارد می‌کنیم. مانند قبل تنها باید به مقادیر مورد نیاز مقدار دهیم و برای این‌که پس از تولید هر **NFT** توکن‌ها با هم متفاوت شوند، از متغیر **tokenCounter** استفاده می‌کنیم.

SimpleCollectible.sol:

```
1 // SPDX-License-Identifier: MIT
```

[۵] InterPlanetary File System
[۶] distributed file system
[۷] Content-Addressable Storage - CAS
[۸] Global Name Space - GNS

```
pragma solidity 0.6.6;
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract SimpleCollectible is ERC721{
    uint256 public tokenCounter;

    // constructor of erc721 takes a name and a symbol
    constructor () public ERC721 ("Dogie" , "Dog"){
        tokenCounter = 0;
    }

    function createCollectible(string memory tokenURI) public
       returns (uint256){
        uint256 newTokenId = tokenCounter;
        _safeMint(msg.sender, newTokenId);
        _setTokenURI(newTokenId, tokenURI);
        tokenCounter = tokenCounter + 1;
        return newTokenId;
    }
}
```

همچنین باید فایل metadata را بسازیم:

```
{"name": "ST_BERNARD",
 "description": "a ST_BERNARD pup ",
 "image": "https://ipfs.io/ipfs/
    QmUPjADFGEKmfohdTaNcWhp7VGk26h5jXDA7v3VtTnTLcW?filename=st-
    bernard.png",
 "attributes": [{"trait_type": "speed", "value": 99}]}
```

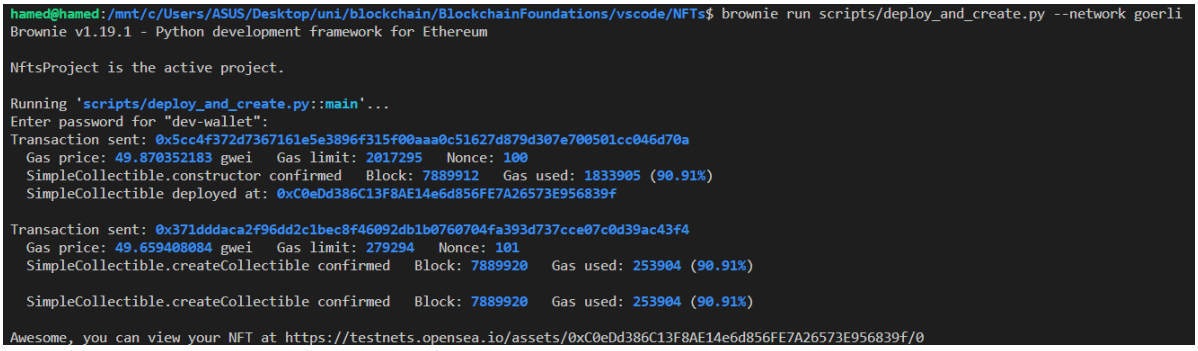
عکس را در IPFS بارگزاری کرده‌ایم و سپس کل فایل metadata را نیز در IPFS می‌گذاریم. در قرارداد SimpleCollectible.sol تابع createCollectible به‌عنوان ورودی tokenURI را می‌گیرد. همان‌طور که در زیر مشاهده می‌شود، زمانی‌که قرارداد را deploy می‌کنیم، برای فراخوانی این تابع، URI را به‌عنوان ورودی به تابع می‌دهیم. tokenURI در این لینک قرار دارد.

deploy.py:

```
from scripts.utils import get_account
from brownie import SimpleCollectible

OPENSEA_URL = "https://testnets.opensea.io/assets/{}/{}"

# uri of our nft:
sample_token_uri = "https://ipfs.io/ipfs/
    Qmd9MCGtdVz2miNumBHDbvj8bigSgTwnr4SbyH6DNnpWdt?filename=0-PUG
    .json"

# token will be deployed on opensea in this format:
```

```
10 # https://testnets.opensea.io/assets/AddressOfSmartContract/
       tokenId
11
12 def main():
13     account = get_account()
14     simple_collectible = SimpleCollectible.deploy({"from":
           account})
15     # we mint the NFT with createCollectible functino from
           SimpleCollectible contract
16     tx = simple_collectible.createCollectible(sample_token_uri,
           {"from": account})
17     tx.wait(1)
18     print(
19         f"Awesome, you can view your NFT at {OPENSEA_URL.format(
               simple_collectible.address, simple_collectible.
               tokenCounter() - 1)}"
20     )
21     #tokenCounter-1 because tokenCounter will be incremented
           after we assign it as tokenId of the minted NFT
22
23     return simple_collectible
```

در پایان قرارداد را روی تست‌نت goerli اجرا کرده و به نتیجه زیر میرسیم:



توکن را می‌توان در اینجا مشاهده کرد. برای باز کردن لینک ممکن است نیاز به فیلترشکن داشته باشید.

نتیجه‌گیری

در این پژوهش مروری کلی روی مفهوم بلاکچین ارایه نمودیم و سپس به بررسی دو بلاکچین بیتکوین و اتریوم پرداختیم. در بلاکچین اتریوم قابلیت نوشتن قراردادهای هوشمند وجود دارد و یک مثال لاتاری را پیاده‌سازی کردیم و همچنین با مفهوم اراکل آشنا شدیم که یکی از کاربردهایش یعنی تولید عدد تصادفی را در این مثال نشان دادیم. در پایان به معرفی مفهوم توکن و استانداردهای طراحی توکن پرداختیم و سپس دو توکن با استانداردهای ERC20 و ERC721 را پیاده‌سازی کردیم.

واژه‌نامه

بلاکچین ..... Blockchain
دفترکل ..... ledger
توزیع‌شده ..... distributed
غیرمتمرکز ..... decentralized
لیست پیوندی ..... linked list
بلاک ..... block
هش ..... hash
رمزنگاری ..... cryptography
اثبات کار ..... proof of work
بیتکوین ..... bitcoin
اتریوم ..... ethereu
قرارداد هوشمند ..... smart contract
اراکل ..... oracle
توکن ..... token
توکن غیرقابل تعویض ..... non fungible token
بیت‌گلد ..... bitgold
ماین ..... mine
ماینر ..... miner
گره ..... node
گره بیزانسی ..... byzantine node
همتابه‌همتا ..... peer to peer
اجماع ..... consensus
تراکنش ..... transaction
درخت مرکل ..... merkle tree
بلاک اولیه ..... genesis block
نانس ..... nonce
مهاجم ..... adversary
عدد تصادفی ..... random number
عدد شبه تصادفی ..... pseudorandom number
قطعی ..... deterministic
کلید عمومی ..... public key

کلید خصوصی ..... private key
امضای دیجیتالی ..... digital signature
تحمل خطا ..... fault tolerance
اثبات سهم ..... proof of stake
حمله سیبیل ..... sybil attack
کیف‌پول ..... wallet
سالیدیتی ..... solidity
ماشین مجازی اتریوم ..... ethereum virtual machine
حم بیضوی ..... elliptic curve
درخت تراکنش ..... transaction trie
رسیدگی به استثنا ..... exception handling
تورینگ کامل ..... turing complete
قابل تعویض ..... fungible
غیرقابل تعویض ..... non fungible

کتاب‌نامه

[1] Bashir, I. (2020) *MASTERING BLOCKCHAIN - THIRD EDITION: A Deep Dive Into Distributed Ledgers, Consensus Protocols,. . . Smart Contracts, Dapps, Cryptocurrencies, Ethereum*

[2] Lamport, L., Shostak, R. E., Pease, M. C. (2019). *The Byzantine generals problem*

[3] Szabo, N. (1997). *Formalizing and Securing Relationships on Public Networks.*

[4] Satoshi Nakamoto (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System.*

[5] Buterin, V. (2013). *Ethereum white paper.*

[6] Buterin, V. (2014). *A next-generation smart contract and decentralized application platform.*

Abstract

In this research after introducing distributed systems, we will eventually discuss general concepts we need to understand what a blockchain is. Blockchain is a decentralized, distributed ledger which is used to store data in a data structure similair to a linked list. Elements of this linked list are called a block which are connected to the hash of the previous block. In the following chapter we will talk about mechanism of a blockchain and some of it's benefits and it's disadvantages. Next we will discuss some concepts in cryptography and it's use cases in blockchain technology. Then we will discuss consensus algorithms and describe PoW algorithm briefly. In chapters five and six we introduce BitCoin and Ethereum Blockchain and smartcontracs. At the end of chapter six we will write a lottery smart contract for the sake of explaining Oracle blockchains. In the final chapter we will introduce tokens and token standards and make one token with ERC20 standard and another one with ERC721 standard which is called an NFT.

College of Science
School of Mathematics, Statistics, and Computer Science

A review on Blockchain and designing an NFT

Hamed Marvi

Supervisor: Dr. Hadi Farahani, Dr. Morteza Mohammad Nouri

A thesis submitted in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science

Winter 2022