

O'REILLY®

Mastering Bitcoin

UNLOCKING DIGITAL CRYPTOCURRENCIES

Andreas M. Antonopoulos



Mastering Bitcoin by [Andreas M. Antonopoulos LLC](#) is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](#).



Hungarian translation by Bitcoin333, Eureka, Dris is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](#)

Magyarra fordította Bitcoin333, Eureka, Dris. A "["Nevezd meg! Így add tovább! - 4.0"](#)" licenc feltételei érvényesek.

Ezt a "Szabad Kultúra" kompatibilis engedélyt az O'Reilly Media (<http://oreilly.com>) kiadó adta, amely méltányolja a nyílt forráskód értékét. Az O'Reilly Media nem csupán a világ legjobb műszaki könyvkiadója, hanem a szabad kultúra és a tudás-megosztás elkötelezettségi híve is. Köszönet az O'Reilly-nek!

Verzió: 2015. május 21., V0.31

Bitcoin fejlesztőknek

Tartalom

A „Bitcoin fejlesztőknek” című könyv méltatása.....	10
Előszó	12
A bitcoin könyv megírása	12
Olvasóközönség.....	12
Miért vannak a borítón bogarak?.....	12
A könyvben használt szabályok.....	13
Példa programok	14
A programozási példák használata.....	14
Safari® Books Online	14
Hogyan tud kapcsolatba lépni velünk	15
Köszönhetetlenítésök	15
Korai változat (Github segítők).....	17
Szómagyarázat	20
1. Bevezetés.....	24
Mi a bitcoin?	24
A bitcoin története	26
A bitcoin használata a felhasználók szemszögéből	27
Elindulás	28
Gyors elindulás	29
Az első bitcoinok beszerzése	31
Bitcoin küldés és fogadás	32
2. A bitcoin működése	36
Tranzakciók, blokkok, bányászat és a blokklánc.....	36
A bitcoin áttekintése	36
Vegyük egy csésze kávét	37
Bitcoin tranzakció	38
A leggyakrabban előforduló tranzakciók.....	40
Egy tranzakció létrehozása	42

A megfelelő bemenetek kiválasztása	42
A kimenetek létrehozása	43
A tranzakció hozzáadása a nyilvántartáshoz	44
Bitcoin bányászat.....	46
Blokkok létrehozása a tranzakciókból	47
A tranzakció elköltése	48
3. A bitcoin kliens.....	50
Bitcoin Core – a referencia implementáció.....	50
A Bitcoin Core kliens első futtatása.....	50
A Bitcoin Core kliens lefordítása a forráskódból	51
A Bitcoin Core JSON-RPC API-jának a használata a parancssorból.....	57
A Bitcoin Core kliens státuszának lekérdezése.....	58
A pénztárca beállítása és titkosítása	59
A pénztárca biztonsági mentése, egyszerű szövegként történő kivitele és visszaállítása	60
Pénztárca címek és pénzt fogadó tranzakciók	60
Tranzakciók vizsgálata és dekódolása	62
Blokkok vizsgálata.....	65
Tranzakciók létrehozása, aláírása és feladása az el nem költött kimenetek alapján	67
Alternatív kliensek, könyvtárak és eszközök	72
A Libbitcoind és a Bitcoin Explorer	73
pycoin	74
btcd.....	76
4. Kulcsok, címek, pénztárcák	78
Bevezetés.....	78
A nyilvános kulcsú titkosítás és a digitális pénz.....	78
Titkos és nyilvános kulcsok.....	79
Titkos kulcsok	80
Nyilvános kulcsok	81
Elliptikus görbükkkel történő titkosítás.....	82
Egy nyilvános kulcs előállítása	85
Bitcoin címek	86
Base58 és Base58Check kódolás	88
Kulcs formátumok	92
Kulcsok és címek kezelése Pythonban.....	97

Pénztárcák	102
Nem-determinisztikus (véletlen) pénztárcák	102
Determinisztikus (magot használó) pénztárcák	102
Mnemonikok	103
Hierarchikus determinisztikus pénztárcák (BIP0032/BIP0044)	104
Kódolt titkos kulcsok (BIP0038)	113
Titkosított (kódolt) titkos kulcsok (BIP0038)	113
Fizetés script hashnek (P2SH, pay to script hash) címek és több aláírást megkövetelő (multi-sig) címek	114
Kérkedő címek	116
Papír pénztárcák	121
5. Tranzakciók	124
Bevezetés	124
A tranzakció életciklusa	124
Tranzakciók létrehozása	124
A bitcoin tranzakció elküldése a bitcoin hálózatnak	125
A tranzakciók szétterjedése a bitcoin hálózatban	125
A tranzakciók szerkezete	126
Tranzakció kimenetek és bemenetek	127
A tranzakció kimenetei	128
Költési feltételek (akadályok)	130
A tranzakció bemenetei	130
Tranzakciós díjak	133
A tranzakciós díj megadása	134
Tranzakciós láncok, árva tranzakciók	135
A tranzakciós scriptek és a script nyelv	136
Script létrehozása (zárolás + zárolás feloldás)	136
Turing nem teljesség	140
Állapotmentes ellenőrzés	140
Szabványos tranzakciók	140
Fizetés nyilvános kulcs hashnek, (P2PKH, Pay-to-Public-Key-Hash)	141
Fizetés nyilvános kulcsnak (Pay-to-Public-Key)	142
Többszörös aláírás (Multi-Signature)	143
Adat kimenet (OP_RETURN)	144

Fizetés script hashnek (Pay to Script Hash, P2SH).....	146
6. A bitcoin hálózat	150
Peer-to-peer hálózati felépítés.....	150
Csomópont típusok és szerepek.....	150
A kiterjesztett bitcoin hálózat	152
Hálózat felderítés	155
Teljes csomópontok	158
„Leltár” egyeztetés	159
Egyszerűsített fizetés ellenőrzést használó csomópontok (SPV csomópontok)	160
Bloom szűrők	163
A Bloom szűrők és a leltár frissítések	167
Tranzakció poolok.....	167
Figyelmeztető üzenetek	168
7. A blokklánc.....	170
Bevezetés.....	170
Egy blokk szerkezete.....	171
A blokkfejléc	171
Blokk azonosítók: Blokkfejléc, Hash, és Blokk magasság	172
A „Genesis” Blokk	173
A blokkok blokkláncá történő összekapcsolása.....	174
Merkle-fák	175
A Merkle-fák és az egyszerűsített fizetés ellenőrzés (Simplified Payment Verification (SPV))	181
8. Bányászat és konszenzus.....	182
Bevezetés.....	182
A bitcoin gazdaság és a pénz kibocsátás	183
Decentralizált konszenzus	185
A tranzakciók egymástól független ellenőrzése.....	186
Bányász csomópontok.....	187
A tranzakciók blokkokba gyűjtése	187
Tranzakció életkor, díjak és prioritás.....	188
A generáló tranzakció	190
A coinbase jutalom és tranzakciós díjak.....	191
A generáló tranzakció felépítése	192

Coinbase adatok	193
A blokkfej előállítása.....	196
A blokk kibányászása	197
Munkabizonyíték algoritmus.....	197
A nehézségi szint ábrázolása	202
A cél nehézségi szint és a nehézségi szint újraszámítása	203
A blokk sikeres kibányászása.....	205
Az új blokk ellenőrzése	206
A blokklánc összeállítása és kiválasztása.....	206
Blokklánc elágazások	208
Bányászat és versenyfutás a hash kapacitásban	212
Az extra nonce megoldás	214
Bánytársaságok (Mining Pools)	214
Felügyelt bányatársaságok (Managed pools).....	216
A konszenzus elleni támadások.....	218
9. Alternatív blokkláncok, fizetőeszközök, alkalmazások.....	222
Az alternatív fizetőeszközök és blokkláncok osztályozása	222
Meta-coin platformok	223
Színezett érmék (Colored Coins)	223
Mastercoin.....	224
Counterparty	225
Alt-coinok	225
Egy alt-coin kiértékelése.....	226
Pénzügyi paraméterű alternatívák: Litecoin, Dogecoin, Freicoin.....	227
A konszenzusrendszer újításai: Peercoin, Myriad, Blackcoin, Vericoin, NXT	228
Kettős-célú bányászati innovációk: Primecoin, Curecoin, Gridcoin	230
Anonimitás-fókuszú alt-coinok: CryptoNote, Bytecoin, Monero, Zerocash/Zerocoin, Darkcoin	231
Nem fizetőeszközként használt alt-chainek	232
Namecoin	232
Ethereum	234
A fizetőeszközök jövője	235
10. A bitcoin biztonsági kérdései.....	236
Biztonsági alapelvek	236
Bitcoin rendszerek biztonságos fejlesztése	237

A bizalom alapja	238
A felhasználók számára követendő példák	238
Fizikai bitcoin tárolás	239
Hardverrel megvalósított pénztárcák.....	240
Kockázat kezelés.....	240
Kockázat megosztás.....	240
A többszörös aláírás és a vállalatirányítás.....	240
Túlélési képesség.....	240
Befejezés	241
A. függelék: A Script tranzakciós nyelv műveletei, konstansai és szimbólumai	
242	
B. függelék: Bitcoin továbbfejlesztési javaslatok (Bitcoin Improvement Proposals)	248
C. függelék: pycoin, ku és tx	252
Key Utility (KU)	252
Transaction Utility (TX)	258
D. függelék: Bitcoin Explorer (bx) parancsok	262
Példa a bx parancsok használatára.....	263
Kolofon.....	266

A „Bitcoin fejlesztőknek” című könyv méltatása

„Mikor a nagyközönség előtt beszélek a bitcoinról, meg szokták kérdezni tőlem: *de hogyan működik ez az egész?* Most már van egy jó válaszom erre a kérdésre, mert bárki, aki elolvassa a *Bitcoin fejlesztőknek* című könyvet, mély ismereteket fog szerezni arról, hogy hogyan működik a bitcoin, és jó eszközei lesznek ahhoz, hogy megírhassa a digitális pénzekkel kapcsolatos bámulatos alkalmazások következő generációját.”

Gavin Andresen

— a Bitcoin Foundation vezető fejlesztője

„A bitcoin és a blokklánc technológiája lesz az Internet következő generációjának az alapvető építőeleme. A Szilícium-völgy legjobb és legokosabb elméi dolgoznak rajta. Andreas könyve megkönnyíti Önöknek, hogy csatlakozni tudjanak a pénzügyi világ szoftver forradalmához.”

Naval Ravikant

— az AngelList társalapítója

„A *Bitcoin fejlesztőknek* a ma elérhető legjobb műszaki kézikönyv a bitcoinról. Visszatekintve, valószínűleg a bitcoint fogjuk az évtized legjelentősebb technológiájának tartani. Emiatt a könyv minden fejlesztőnek nélkülözhetetlen, különösen azoknak, akik a bitcoin protokollra építve szeretnének alkalmazásokat létrehozni. Nagyon ajánlom ezt a könyvet.”

Balaji S. Srinivasan (@balajis)

— General Partner, Andreessen Horowitz

„Az az újítás, amit a bitcoin blokklánc jelent, egy teljesen új programozási platformot teremt, amely épp olyan széleskörű és változatos ökoszisztemát tesz majd lehetővé, mint maga az Internet. Kiváló gondolkodóként Andreas Antonopoulos tökéletes választás volt a könyv megírására.”

Roger Ver

— bitcoin vállalkozó és befektető

Előszó

A bitcoin könyv megírása

Először 2011 közepén botlottam a bitcoinba. A reakcióm többé-kevésbé az volt, hogy „Brr! Egy újabb pénz a számítógép megszállottjainak!” – és további 6 hónapig nem törödtem vele, nem értettem meg a fontosságát. Az általam ismert legokosabb embereknél is sokszor láttam ugyanezt a reakciót, ami egy kicsit vigasztaló. Másodszor, amikor egy levelező listán került szóba a bitcoin, elhatároztam, hogy elolvassom Satoshi Nakamoto dolgozatát, azaz áttanulmányozom a hiteles forrást és megnézem, hogy miől van szó. Még mindig emlékszem arra a pillanatra, amikor befejeztem a 9 oldalas dolgozatot és megértettem, hogy a bitcoin nem egyszerűen csak egy digitális pénz, hanem egy decentralizált bizalmi hálózat, és a pénznél sokkal több mindennek az alapjául szolgálhat. Ez a felismerés indított arra, hogy 4 hónapon keresztül a bitcoinra vonatkozó minden információmorzsát összeszedjek és elolvassak. Lelkesedtem és megszállottá váltam, napi 12 órát vagy még többet töltöttem a képernyőre tapadva, olvastam, írtam, kódoltam, és tanultam, amennyit csak tudtam. 10 kilót fogytam ezalatt, mert nem ettem rendesen, hanem csak a bitcoinnal foglalkoztam.

Két évvel később, miután számos kis céget alapítottam a bitcoinnal kapcsolatos szolgáltatások és termékek vizsgálatára, elhatároztam, hogy megírom az első könyvemet. A bitcoin volt az a téma, ami kreatívvá tett, lefoglalta a gondolataimat: ez volt a legizgalmasabb technológia, amiőt csak találkoztam az Internettel. Itt az idő, hogy a tágabb olvasóközönséggel is megosszam azt az elragadtatásomat, amit ezzel az izgalmas technológiával kapcsolatban érzek.

Olvasóközönség

A könyvet leginkább programozóknak szántam. Ha az Olvasó ismer egy programozási nyelvet, akkor ez a könyv megtanítja neki, hogyan működnek a digitális pénzek, hogyan lehet használni őket és hogyan írható olyan szoftver, amelyik ezekre épül. Az első néhány fejezet a nem programozók számára is jó bevezetést jelent a bitcoin világába – vagyis azoknak, akik szeretnék megérteni a bitcoin és a digitális pénzek belső működését.

Miért vannak a borítón bogarak?

A levélvágó hangya egy olyan faj, amely egy szuper-kolóniában nagyon bonyolult viselkedésre képes. Ugyanakkor az egyes hangyák nagyon egyszerű szabályok szerint járnak el, amelyeket a szociális érintkezés és kémiai szaganyagok (feromonok) cseréje határoz meg. A Wikipédia szerint: „A Földön az emberek mellett a levélvágó hangyák alkotják a legnagyobb és legbonyolultabb társadalmakat.” A levélvágó hangyák ténylegesen nem eszik meg a leveleket, hanem gombatermelésre használják, ami a hangyaboly fő élelemforrása. Értik? Ezek a hangyák termelnek!

Noha a hangyáknál kaszt-alapú a társadalom, és egy királynő hozza létre az utódokat, de a hangyabolyban nincs központi szervezet vagy vezető. A sok millió tagot számláló hangyaboly rendkívül intelligens és okos viselkedése az egyes egyedek kölcsönhatásai révén előálló sajátosság („emergens tulajdonság”).

A természet azt szemlélteti, hogy a decentralizált rendszerek ellenállóak tudnak lenni és központi szervezet, hierarchia vagy bonyolult részek nélkül is emergens bonyolultságot és okosságot tudnak létrehozni.

A bitcoin egy nagyon összetett decentralizált bizalmi hálózat, amely pénzügyi folyamatok milliárdjainak a támogatására képes. Ugyanakkor minden egyes csomópont néhány egyszerű matematikai szabályt követ. A bonyolult viselkedés megjelenése a csomópontok közötti kölcsönhatások eredménye, nem pedig valamilyen belső bonyolultságnak vagy az egyes csomópontokba vetett bizalomnak köszönhető. Egy hangyabolyhoz hasonlóan a bitcoin hálózat is egyszerű szabályokat követő csomópontok hálózata, amely képes ellenállni a támadásoknak, és együttesen központi koordináció nélkül is csodálatos dolgokra képes.

A könyvben használt szabályok

A könyv a következő tipográfiai szabályokat használja:

Dőlt betűs szedés

egy új szó, URL, e-mail cím, állománynév vagy kiterjesztés jelölésére szolgál.

Fix szélességű szedés

Programlisták, valamint egy bekezdésen belül valamelyen programrészlet, pl. változók vagy függvények, adatbázisok, adattípusok, környezeti változók, utasítások és kulcsszavak esetén használatos.

Fix szélességű, vastag szedés

Parancsok vagy más szövegek, melyeket a felhasználónak pontosan ilyen formában kell beadnia.

Fix szélességű, dőlt betűs szedés

Olyan szöveg, melyet a felhasználó által megadott értékekkel vagy a környezettől függő értékekkel kell helyettesíteni.



Ez az ikon egy tippet, javaslatot vagy általános megjegyzést jelöl.



Ez az ikon egy figyelmeztetést jelöl.



Ez az ikon egy megjegyzést jelöl.

Példa programok

A példák szemléltetésére Python, C++ és egy Unix-szerű operációs rendszer (Linux, Mac OS X) parancssorai szolgáltak. Az összes kódrészlet megtalálható a weben, és online módon is elérhető a [GitHub kódítár](#) URL címen, a főkönyvtár *code* alkönyvtárában. A GitHub-ról töltse le a kódot, próbálja ki a példákat, vagy jelezze a hibákat.

A kódrészletek a legtöbb operációs rendszer alatt a megfelelő nyelvek installálásával, minimális munkával működőképessé tehetők. Ha szükségesnek láttuk, akkor megadtuk az installáláshoz szükséges utasításokat és az utasítások lépésről lépésre történő végrehajtása során keletkező kimeneteket.

Némelyik kódrészletet és kódrészlet kimenetet a nyomtatás érdekében újraformáztuk. Az összes ilyen esetben a sorokat egy "\\" karakterrel választottuk el, melyet egy új sor karakter követ. A példák újból futtatásakor távolítsák el ezt a két karaktert és egyesítsék újra a sorokat, ekkor a példában szereplő eredménnyel egyező kimenetet fognak kapni.

Az összes kódrészlet lehetőleg valós értékeket és számításokat használt, emiatt példáról példára haladva ugyanazokat az eredményeket fogja kapni, mint amelyek a példa számításokban szerepelnek. Például a titkos kulcsok és a hozzájuk tartozó nyilvános kulcsok valamint címek minden valódiak. A minta tranzakciók, blokk és blokklánc hivatkozások a tényleges bitcoin blokkláncban lettek elvégezve és a nyilvános főkönyv részei, ezért bármelyik bitcoin rendszerben lekérdezhetők.

A programozási példák használata

A könyv szeretné segíteni az önök munkáját. Ha a könyv egy mintapéldát ad valamire, akkor ön ezt felhasználhatja a saját programjaiban vagy dokumentációiban. Nem kell tölünk engedélyt kérnie, ha csak kisebb részleteket használ a példákból. Például ha ír egy programot, amely a könyv néhány kódrészletét használja, akkor ehhez nem kell engedélyt kérnie. Az O'Reilly könyvek CD-ROM-jain közölt példák árusítása vagy sokszorosítása azonban engedélyköteles. Ha a könyvből vett idézettel válaszol egy kérdésre, vagy idéz egy példa programot, az nem engedélyköteles. De ha jelentős mennyiségű példaprogramot vesz át a könyvből, az engedélyköteles.

Köszönnett vesszük, de nem kötelező a hivatkozás. A hivatkozás általában a címet, a szerzőt, a kiadót és az ISBN számot tartalmazza. Például: "Mastering Bitcoin, by Andreas M. Antonopoulos (O'Reilly). Copyright 2015 Andreas M. Antonopoulos, 978-1-449-37404-4."

A könyv egyes kiadásaihoz nyílt forráskódú engedély tartozik, mint pl. CC-BY-NC (creativecommons.org). Ebben az esetben a vonatkozó engedély a hatályos.

Ha úgy érzi, hogy a példa programok felhasználása során túllépte a fent engedélyezett mértékét, akkor lépj kapcsolatba velünk: permissions@oreilly.com.

Safari® Books Online

A Safari Books Online egy on-demand digitális könyvtár, amely könyvek és videók formájában tartalmazza a világ legjobb szakíróinak műveit.

A műszaki szakemberek, szoftver fejlesztők, web tervezők és egyéb szakemberek a Safari Books Online-t kutatásra, problémamegoldásra, tanulásra és vizsga felkészítő forrásként használják.

A Safari Books Online számos terméket és árajánlatot ad a szervezetek, kormányszervek és magánemberek számára. Az előfizetők több ezer könyvhöz, oktató videóhoz és könyv vázlathoz férnek hozzá egy kereshető adatbázisban. A kiadók között megtalálható az O'Reilly Media, a Prentice Hall Professional, az Addison-Wesley Professional, a Microsoft Press, a Sams, a Que, a Peachpit Press, a Focal Press, a Cisco Press, a John Wiley & Sons, a Syngress, a Morgan Kaufmann, az IBM Redbooks, a Packt, az Adobe Press, az FT Press, az Apress, Manning, a New Riders, a McGraw-Hill, a Jones & Bartlett, a Course Technology, és több tucat további kiadó. Ha további információkat szeretne kapni a Safari Books Online-ról, kérjük, látogasson el online webhelyünkre.

Hogyan tud kapcsolatba lépni velünk

A könyvvel kapcsolatos megjegyzéseit és kérdéseit kérjük, küldje el a kiadónak:

O'Reilly Media, Inc.
1005 Gravenstein Highway North Sebastopol, CA
95472 800-998-9938 (az Amerikai Egyesült Államokban vagy Kanadában)
707-829-0515 (nemzetközi vagy helyi)
707-829-0104 (fax)

A könyvnek van egy weblapja, ahol hibajegyzék, példák és további információk szerepelnek. A weblap a http://bit.ly/mastering_bitcoin címen érhető el.

A könyvvel kapcsolatos megjegyzéseit vagy műszaki kérdéseit az alábbi címre küldheti:
bookquestions@oreilly.com.

Könyveinkről, tanfolyamainkról, konferenciáinkról, híreinkről további információkat találhat a webhelyünkön: <http://www.oreilly.com>.

Fent vagyunk a Facebook-on: <http://facebook.com/oreilly>
Kövessen minket a Twitter-en: <http://twitter.com/oreillymedia>
Nézzen minket a YouTube-on: <http://www.youtube.com/oreillymedia>

Köszönetnyilvánítások

Ez a könyv sok ember munkájának és közreműködésnek köszönhető. Hálás vagyok azért a segítségért, amelyet a barátaimtól, kollégáimtól és teljesen ismeretlen emberektől kaptam, akik velem együtt részt vettek a bitcoinról és a digitális pénzkről szóló definitív szakkönyv megríásában.

Lehetetlen különbséget tenni a bitcoin technológia és a bitcoin közösség között, és ez a könyv épp annyira a közösség által létrehozott eredmény, mint amennyire a technológiáról szóló könyv. A könyvvel kapcsolatos munkámat a kezdetektől a legvégeig az egész bitcoin közösség bátorította, örömmel fogadta és támogatta. Ez a könyv mindenek előtt lehetővé tette, hogy két éven keresztül része lehettem ennek a csodálatos közösségnak, és nem tudom eléggé

megköszönni, hogy befogadtak maguk közé. Túl sok embert kellene megemlítenem név szerint – olyanokat, akikkel konferenciákon, különféle eseményeken, szemináriumokon, pizzázás során és személyes megbeszéléseken találkoztam, továbbá azokat, akik twitter-en, reddit-en, a bitcointalk.org-on és a github-on kommunikáltak velem, és akik hatással voltak erre a könyvre. minden egyes ötletet, analógiát, kérdést, választ és magyarázatot, amely a könyvben megtalálható, bizonyos szempontból a közösségi kommunikáció inspirálta, tesztelte vagy tette jobbá. mindenkinél köszönöm a támogatását. Enélkül ez a könyv nem születhetett volna meg. Örökre hálás vagyok ezért.

A szerzővé válás folyamata természetesen már sokkal korábban elkezdődik, mielőtt az ember megírná az első könyvét. Az anyanyelvem görög, a tanulmányaimat is ezen a nyelven végeztem, ezért mikor első éves egyetemista voltam, egy angol tanfolyamon kellett részt vennem, hogy jobban tudjak írni angolul. Köszönettel tartozom Diana Kordasnak, az angoltanáromnak, aki segített abban, hogy magabiztosabban és jobban írjak. Később, szakemberként és a *Network World* magazin egyik szerzőjeként az adatközpontok téma körében fejlesztettem írói képességeimet. Köszönettel tartozom John Dixnek és John Gallantnak, akik először bíztak meg azzal, hogy a *Network World*-be írjak, valamint szerkesztőmnek, Michael Cooneynek és kollégámnak, Johna Till Johnsonnak, aikik szerkesztői munkájukkal alkalmassá tették cikkeimet a megjelentetésre. Négy éven keresztül minden héten 500 szót írtam, ennek során elég gyakorlatot szereztem ahoz, hogy végül is szerzőnek tekinthessem magam. Köszönöm Jean korai bátorítását, hitét, és meggyőződését, hogy egyszer még könyvet fogok írni.

Köszönet azoknak, akik hivatkozásaikkal és kritikáikkal támogattak, amikor benyújtottam az O'Reilly-nek a könyvre vonatkozó javaslatomat. Nevezetesen, köszönettel tartozom John Gallantnak, Gregory Nessnek, Richard Stiennonak, Joel Snydernek, Adam B. Levine-nek, Sandra Gittlennek, John Dixnek, Johna Till Johnsonnak, Roger Vernek és Jon Matonisnak. Külön köszönet Richard Kagannak és Tymon Mattosznak aikik a javaslatom korai változatait elbírálták, valamint Matthew Owain Taylornak, aki a javaslatot megszerkesztette.

Köszönet Cricket Liunak, aki a *DNS* és *BIND* című O'Reilly könyv szerzője. Ő mutatott be engem a kiadónak. Köszönet Michael Loukidesnek és Allyson MacDonaldnek, aikik az O'Reilly dolgozói, és hónapokon át együtt dolgoztak velem, hogy ez a könyv létrejöhessen. Allyson különösen türelmes volt, ha túlléptem a határidőket, és késve nyújtottam be valamit, mert az élet felülírta a tervezeteket.

Az első néhány vázlat és az első pár fejezet megírása volt a legnehezebb, mert a bitcoin téma jának a kifejtése nehéz. minden egyes alkalommal, amikor megváltoztattam valamit a bitcoin technológia magyarázata során, az egész anyagot át kellett dolgoznom. Sokszor megakadtam, és kicsit kétségebe estem, mikor azzal küzdöttem, hogy az egész témát könnyen érthetővé tegyem, és leírjam ezt a bonyolult műszaki témát. Végül úgy döntöttem, hogy a bitcoin történetét a bitcoin felhasználónak a szemszögéből mondjam el. Ez nagyban megkönnyítette a könyv megírását. Köszönettel tartozom barátomnak és mentoromnak, Richard Kagannak, aki segített kibontani a történetet, és segített legyőzni az írói leblokkolásokat, valamint Pamela Morgannek, aki a fejezetek korai vázlatait átnézte, és nehéz kérdéseket tett föl nekem – ezektől lettek jobbak a fejezetek. Köszönettel tartozom a San Francisco-i bitcoin fejlesztők társaságának, valamint Taariq Lewisnak, a csoport egyik alapítójának, mert segítették az anyag elbírálását a korai fázisban.

A könyv megírása során a korai vázlatokat elérhetővé tettek a Github-on, és várta a megjegyzéseket. Több, mint száz megjegyzést, javaslatot, javítást és hozzájárulást kaptam. Ezeket a hozzájárulásokat külön is kiemeltem és megköszöntem a [Korai változat \(Github segítők\)](#) részben. Külön köszönetet szeretnék mondani Minh T. Nguyennek, aki önként vállalta, hogy karbantartja a Github hozzászólásokat, és saját maga is jelentős módon hozzájárult a könyvhöz. Köszönet továbbá Andrew Nauglernek az infografika megtervezéséért.

A könyv első változata számos műszaki szemlén ment keresztül. Köszönet Cricket Liunak és Lorne Lantznak az alapos műszaki kritikájukért, megjegyzésekért és támogatásukért.

Számos bitcoin fejlesztőtől kaptam programokat, kritikákat, megjegyzéseket és bátorítást. Köszönetet mondok Amir Taakinak a kód részletekért és a sok nagyszerű megjegyzéséért, Vitalik Buterinnek és Richard Kissnek az elliptikus görbükkkel kapcsolatos matematikai megjegyzésekért, Gavin Andresennek a javításaiért, megjegyzéseiért és bátorításáért, Michalis Karagisnek a megjegyzéseiért, közreműködéséért és a btcd összefoglalójáért.

A szavak és könyvek szeretetét anyámnak, Theresának köszönhetem, aki egy olyan házban nevelt fel, amelyben minden falon könyvek voltak. Anyám vette nekem az első számítógépemet is 1982-ben, bár ő maga technofóbiás, a saját állítása szerint. Apám, Menelaos, építőmérnök, és 80 éves korában jelentette meg az első könyvét. Ő volt az, aki megtanított a logikus és analitikus gondolkodásra, valamint a műszaki tudományok szeretetére.

Köszönet mindenkinnek, aki segített nekem megtenni ezt az utat.

[Korai változat \(Github segítők\)](#)

Sok segítőtől kaptam megjegyzéseket, javításokat és bővítéseket a Github-on lévő korai kiadáshoz. Köszönet az összes segítségért! A legjelesebb GitHub segítők a következők voltak (a GitHub azonosítójuk zárójelben látható):

- Minh T. Nguyen, GitHub szerkesztő (enderminh)
- Ed Eykholt (edeykholt)
- Michalis Kargakis (kargakis)
- Erik Wahlström (erikwam)
- Richard Kiss (richardkiss)
- Eric Winchell (winchell)
- Sergej Kotliar (ziggamon)
- Nagaraj Hubli (nagarajhubli)
- ethers
- Alex Waters (alexwaters)
- Mihail Russu (MihailRussu)
- Ish Ot Jr. (ishotjr)
- James Addison (jayaddison)
- Nekomata (nekomata-3)
- Simon de la Rouviere (simondlr)
- Chapman Shoop (belovachap)
- Holger Schinzel (schinzelh)
- effectsToCause (vericoin)

- Stephan Oeste (Emzy)
- Joe Bauers (joebauers)
- Jason Bisterfeldt (jbisterfeldt)
- Ed Leafe (EdLeafe)

Szómagyarázat

A lenti szómagyarázatban a bitcoinnal kapcsolatos szavak, kifejezések vannak, melyek nagyon sokszor előfordulnak ebben a könyvben. Tegyen ide egy könyvjelzőt, hogy gyorsan megtalálhassa és tisztázhassa a kérdéses kifejezéseket.

bányász

Egy olyan hálózati csomópont, amely új blokkok előállítása érdekében, ismételten végrehajtott hash számítás segítségével munkabizonyítékot (proof-of-work) keres.

bip

Bitcoin Improvement Proposals (a bitcoin tökéletesítésére tett javaslatok). Olyan javaslatok, melyeket a bitcoin közösség tagjai tettek a bitcoin javítása, tökéletesítése érdekében. Például a BIP0021 a bitcoin URI sémajának tökéletesítésére tett javaslat.

bitcoin

A pénzegység (érme), a hálózat és a szoftver neve.

blokk

Tranzakciók csoportja, mely egy időbélyeget, valamint az előző blokk ujjlenyomatát tartalmazza. A blokk blokkfejlécének hashelése révén áll elő a munkabizonyíték (proof-of-work), és válnak érvényessé a tranzakciók. Az érvényes blokkok a hálózati konszenzus alapján bekerülnek a fő blokkláncba.

blokklánc

Érvényesített blokkok listája, amelyben mindenki blokk kapcsolódik az előzőhez, egészen a genezis blokkig visszamenőleg.

cél nehézségi szint

Az a nehézségi szint, amely mellett a hálózat kb. 10 percenként fog egy blokkot találni.

cím

Egy bitcoin cím a következőképpen néz ki: 1DSrfJdB2AnWaFNgSbv3MZC2m74996JafV. A bitcoin cím betűkből és számokból áll, és egy „1”-gyel kezdődik. Egy bitcoin címre bitcoinok küldhetők, pont úgy, ahogy egy e-mail címmel elektronikus leveleket lehet fogadni.

díjak

A tranzakció küldője által megfizetett díj, amelyet a hálózat kap a tranzakció feldolgozásáért. A legtöbb tranzakcióhoz min. 0.1 mBTC díjra van szükség.

genezis blokk

A blokklánc első blokkja, mely a digitális pénz inicializálására szolgál.

hash

Valamilyen bináris bemenet digitális ujjlenyomata. Magyarul néha zanzának is nevezik, de ez nem terjedt el.

hálózat

Egyenrangú csomópontokból álló, peer-to-peer hálózat, amely a hálózatban lévő összes többi bitcoin csomópontnak továbbítja a tranzakciókat és a blokkokat.

jutalom

Az új blokkokban szereplő pénzösszeg, melyet a hálózat annak a bányásznak ad, aki megtalálta a blokkhoz a munkabizonyítékot. A jutalom jelenleg 25 BTC/blokk.

megerősítések

Ha egy tranzakció bekerült egy blokkba, akkor „egy megerősítéssel rendelkezik”. Ha már egy *újabb* blokk is előállt ugyanezen a blokkláncon, akkor a tranzakciónak két megerősítése van stb. Hat vagy még több megerősítés már elégéges bizonyítéknak tekinthető arra nézve, hogy a tranzakciót nem lehet visszafordítani.

munkabizonyíték, proof-of-work

Egy olyan számérték, melynek előállítása jelentős számítási kapacitást igényel. A bitcoin esetében a bányászok az SHA256 algoritmust használják arra, hogy egy olyan hasht találjanak, amely megfelel a hálózat egészében fennálló cél nehézségi szintnek.

nehézségi szint

Egy hálózati beállítás, amely azt határozza meg, hogy mennyi számítási munkára van szükség egy munkabizonyíték (proof-of-work) előállításához.

nehézségi szint újraszámítás

A nehézségi szint újraszámítása, amely 2016 blokkonként az egész hálózatban megtörténik, és az előző 2016 blokk előállításához használt hash kapacitást/teljesítményt veszi figyelembe.

pénztárca

Egy szoftver, amely a felhasználó bitcoin címeit és titkos kulcsait kezeli. A pénztárca bitcoinok küldésére, fogadására és tárolására használható.

titkos kulcs (azaz privát kulcs)

Egy titkos szám, amely megszünteti a neki megfelelő címre küldött bitcoinok zárolását. Egy titkos kulcs így néz ki pl.:

5J76sF8L5jTtzE96r66Sf8cka9y44wdpJjMwCxR3tzLh3ibVPxh.

tranzakció

Egyszerűen bitcoin küldés az egyik címről a másikra. Pontosabban, a tranzakció egy aláírással rendelkező adatstruktúra, amely értéktovábbításnak felel meg. A tranzakciókat a bitcoin hálózat továbbítja, a bányászok blokkokba foglalják őket, és ezáltal bekerülnek a blokkláncba.

1. Bevezetés

Mi a bitcoin?

A bitcoin olyan alapelvek és technológiák összessége, melyek egy digitális pénzrendszer alapját képezik. A bitcoinnak nevezett pénzegység érték tárolásra és továbbításra szolgál a bitcoin hálózat résztvevői között. A bitcoin felhasználók a bitcoin protokoll segítségével kommunikálnak egymással, főleg az Interneten, de egyéb átviteli hálózatok is használhatók. A bitcoin protokoll, amely nyílt forráskódú szoftverként érhető el, széles eszközválasztékon futtatható, többek között notebookokon és okostelefonokon, ami könnyen elérhetővé teszi ezt a technológiát.

A hálózaton át továbbított bitcoinokkal nagyjából ugyanaz megtehető, mint a hagyományos pénzzel, pl. áruvásárlás vagy eladás, pénz küldése magánembereknek vagy szervezeteknek, vagy hitelnyújtás. Bitcoinok az erre specializált pénzváltókban vehetők, adhatók el, vagy válthatók át egyéb pénznemekre. A bitcoin bizonyos értelemben az Internetes pénz tökéletes formája, mivel gyors, biztonságos és határokat átívelő.

A hagyományos pénzekkel ellentétben a bitcoin teljesen virtuális. Nincsenek fizikai érmék, de még digitális érmék sem. Az érméket implicit módon azok a tranzakciók tartalmazzák, melyek a feladótól a címzethez továbbítják az értéket. A bitcoin felhasználóknak csupán kulcsai vannak, amelyekkel a bitcoin hálózaton belül bizonyítani tudják a tranzakcióik tulajdonjogát, fel tudják szabadítani az elköltendő összeget, és továbbítani tudják azt egy új címzettnek. A kulcsok gyakran az egyes felhasználók számítógépén, egy digitális pénztárcában vannak tárolva. A bitcoinok elköltésének az egyetlen előfeltétele az, hogy a felhasználó rendelkezzen a tranzakciók zárolásának feloldásához szükséges kulccsal. Ez azt jelenti, hogy minden egyes felhasználó teljes mértékben maga rendelkezik a pénzával.

A bitcoin egy egyenrangú csomópontkból álló peer-to-peer rendszer. Nincs benne semmilyen „központi” szerver vagy irányítás. A bitcoinok az ún. „bányászat” során jönnek létre. A bányászat egy olyan verseny, melyben a bitcoin tranzakciók feldolgozása során egy matematikai feladat megoldásának a keresése folyik. A bitcoin hálózat bármelyik tagja (vagyis bárki, aki a teljes bitcoin protokollt futtató eszközt használ) bányászként is képes működni, vagyis a számítógépe segítségével képes a tranzakciók ellenőrzésére és tárolására. Átlagosan 10 percenként sikerül valakinek az utolsó tíz percben született tranzakciók érvényesítése, és ezért vadonatúj bitcoinokat kap jutalmul. A bitcoin bányászat lényegében decentralizálja egy központi bank pénzkibocsátási és elszámolási feladatait, és a fenti globális versennel váltja ki a központi bankok iránti igényt.

A bitcoin protokoll olyan beépített algoritmusokat tartalmaz, melyek a hálózat egészében szabályozzák a bányászatot. A bányászok feladata az, hogy sikeresen eltárolják a tranzakciókból álló blokkokat a bitcoin hálózaton belül. E feladat nehézségét a hálózat dinamikusan úgy állítja be, hogy átlagosan minden 10 percben sikkerrel járjon valaki, függetlenül attól, hogy egy adott pillanatban hány bányász (és CPU) dolgozik a probléma megoldásán. A protokoll egyúttal minden 4 évben a felére csökkeni a 10 perceként kibocsátott új bitcoinok számát, és ezáltal az összes létrejövő bitcoinok számát 21 millióra korlátozza. Emiatt a forgalomba kerülő bitcoinok száma egy könnyen megjósolható görbe szerint alakul, amely 2140-re éri el a 21 milliót. Az egyre csökkenő mértékű kibocsátás miatt

a bitcoin hosszabb távon deflációs pénz. A bitcoint nem lehet a tervezett kibocsátási ütemet meghaladó mértékű új pénz ”kinyomtatásával” elinflálni.

A színfalak mögött a protokollt, a hálózatot és az osztott feldolgozás együttesét is bitcoinnak hívják. A bitcoin mint pénz ennek az innovációnak csupán az első alkalmazása. Fejlesztőként számomra a bitcoin a pénzvilág Internet-e: egy hálózat, mellyel érték továbbítható, és ami osztott feldolgozás révén biztosítja a digitális vagyontárgyak tulajdonjogát. A bitcoin sokkal több annál, mint első ránézésre hinnénk.

Ebben a fejezetben először a legfontosabb fogalmakat és kifejezéseket ismertetjük, letölthetők a szükséges szoftvert és egyszerű tranzakciókra használjuk a bitcoint. A további fejezetekben hozzájárultunk azoknak a technológiai rétegeknek a feltáráshoz, melyek lehetővé teszik a bitcoin működését, és megvizsgáljuk a bitcoin hálózat és protokoll belső működését.

A bitcoin előtti digitális pénzek

Az életképes digitális pénz megjelenése szorosan kötődik a kriptográfia fejlődéséhez. Ez nem meglepő, ha szemügyre vesszük azokat az alapvető feladatokat, melyek akkor lépnek föl, ha árukra és szolgáltatásokra elcserélhető értéket szeretnénk bitekkel ábrázolni. Bárki, aki digitális pénzt fogad el, az alábbi két alapvető kérdést teszi fel magának:

1. Bízhatok-e abban, hogy a pénz valódi és nem hamisítvány?
2. Biztos lehetetek-e abban, hogy ez a pénz csak az enyém, és senki másé? (ez az ún. „kettős költés” problémája).

A papírpénz kibocsátás során egyre speciálisabb papírokkal és egyre kifinomultabb nyomtatási módszerekkel veszik fel a harcot a hamisítás ellen. A fizikai pénz esetében a kettős költés kérdésének kezelése egyszerű, hiszen ugyanaz a bankjegy nem lehet egyszerre két helyen. Természetesen a hagyományos pénz tárolása és továbbítása is gyakran digitálisan történik. Ebben az esetben a hamisítás és kettős költés problémáját úgy kezelik, hogy az összes elektronikus tranzakciót központi szervezeteken keresztül bonyolítják le. A központi szervezeteknek globális rálátásuk van a forgalomban levő pénzre. A digitális pénznél, amely nem támaszkodhat különleges tintára vagy holografikus csíkokra, a kriptográfia biztosítja a felhasználók értékekre vonatkozó állításainak a valódiságát. Nevezetesen, egy digitális eszköz vagy tranzakció digitális aláírással látható el, a felhasználó pedig ennek révén képes bizonyítani egy adott eszköz tulajdonjogát. A megfelelő architektúra segítségével a digitális aláírások a kettős költés problémájának a kezelésére is alkalmasak.

Amikor a kriptográfia az 1980-as évek végén kezdett sokkal szélesebb körben elterjedni, sok kutató próbált a kriptográfia segítségével digitális pénzeket létrehozni. Ezek a korai projektek olyan digitális pénzeket hoztak létre, amelyek egy nemzeti valutára vagy valamilyen nemesfémre, pl. az aranya épültek.

Ezek a korai digitális pénzek működtek ugyan, de a centralizáltságuk miatt a kormányok vagy a hackerek támadásainak könnyű célpontjai voltak. A korai digitális pénzek a hagyományos bankrendszerhez hasonlóan egy központi elszámolóházat használtak, amely a tranzakciókat rendszeres időközönként elszámolta. Sajnos a legtöbb esetben ezek a születőfélben lévő digitális pénzek az aggódó kormányok céltáblái lettek, és végül jogi úton felszámolták őket. Voltak közülük olyanok, melyek látványosan összeomlottak, mikor az anyacég hirtelen felszámolásra került. Ahhoz, hogy egy digitális pénz robusztusan ellen

tudjon állni az ellenfelek támadásainak, legyenek ezek törvényes kormányok vagy bűnöző elemek, olyan új decentralizált digitális pénzre volt szükség, amelyben nem volt egypontos támadási felület. A bitcoin egy ilyen rendszer: teljesen decentralizált, és nincs benne semmilyen központi szervezet vagy irányítás, amely megtámadható vagy korrumplítható.

A bitcoin a kriptográfia és az osztott rendszerek évtizedes kutatásának a végeredménye. Négy alapvető újítást tartalmaz, melyek egyedi és hatékony módon vannak kombinálva egymással. A bitcoin alkotó elemei:

- egy decentralizált peer-to-peer hálózat (a bitcoin protokoll),
- egy nyilvános tranzakciós főkönyv (a blokklánc),
- decentralizált, determinisztikus matematikai pénzkibocsátás (osztott bányászat),
- egy decentralizált tranzakció ellenőrző rendszer (tranzakciós scriptek).

A bitcoin története

A bitcoint 2008-ban találta fel Satoshi Nakamoto, amikor megjelentette a „Bitcoin: egy peer-to-peer elektronikus pénzrendszer” című dolgozatát. Satoshi Nakamoto számos előző felfedezés, pl. a b-pénz és a HashCash kombinálásával egy teljesen decentralizált elektronikus pénzrendszeret hozott létre, amelyben a pénzkibocsátás, valamint a tranzakciók elszámolása és ellenőrzése nem egy központi szervezetre támaszkodik. A legfontosabb újítása a munkabizonyíték algoritmuson alapuló, 10 percenténti globális „szavazás”. Ennek révén a decentralizált hálózatban a tranzakciók állapotáról *konszenzus* alakulhat ki. Ez elegáns megoldást jelent. a kettős költés kérdésére – vagyis arra, hogy egy pénzegységet ne lehessen kétszer is elkölni. Korábban a digitális pénz egyik gyengeségét éppen a kettős költés jelentette, melyet úgy kezeltek, hogy az összes tranzakciót egy központ elszámolóházon keresztül rendezték.

A bitcoin hálózat a Nakamoto által publikált referencia implementáció alapján, 2009-ben kezdte meg a működését. A referencia implementációt azóta számos programozó felülvizsgálta. A bitcoin biztonságát és ellenállóképességét biztosító osztott feldolgozási kapacitás exponenciálisan nőtt, és manapság meghaladja a világ leggyorsabb szuper-számítógépeinek a feldolgozási kapacitását. A bitcoin teljes piaci értéke a bitcoin-dollár árfolyamtól függően becslések szerint 5 és 10 milliárd US dollár között van. A hálózat által feldolgozott eddigi legnagyobb tranzakció 150 millió US dollár volt. A tranzakció azonnal továbbításra került, és a feldolgozása díjtalan volt.

Satoshi Nakamoto 2011 áprilisában visszavonult a nyilvánosságtól. A programkódot és a hálózat fejlesztését az egyre gyarapodó önkéntesek csoportjára hagyta. Még ma sem tudjuk, hogy ki vagy kik állhatnak a bitcoin mögött. De a bitcoint se Satoshi Nakamoto, se mások nem tudják befolyásolni, mert a rendszer teljesen átlátható matematikai alapelvek szerint működik. Maga a felfedezés korszakalkotó, és márás egy új tudományárat hozott létre az osztott feldolgozás, a közgazdaságtan és az ökonometria területén.

Megoldás egy osztott feldolgozási problémára

Satoshi Nakamoto felfedezése egyúttal az osztott feldolgozás egy korábban megoldatlan problémájának, az ún. „bizánci generálisok problémájának” a gyakorlati megoldását jelenti. Röviden, a probléma abból áll, hogy hogyan lehet megegyezni a tennivalókról egy megbízhatatlan és potenciálisan kompromittált hálózatban végzett információcsere révén. Satoshi Nakamoto megoldása, amely a munkabizonyíték fogalmának használatával, központi szervezet nélkül éri el a konszenzust, áttörést jelent az elosztott feldolgozás terén, és a pénzügyeken kívül egyéb területeken is széles körben alkalmazható. Bizonyíthatóan igazságos választások, lottójáték, tulajdoni nyilvántartások, digitális közzéjegyzői szolgáltatások és sok minden más esetén is konszenzust lehet vele elérni a decentralizált hálózatokban.

A bitcoin használata a felhasználók szemszögéből

A bitcoin pénz kezelésre szolgáló technológia, vagyis alapjában véve emberek közötti értékcseré. Tekintsünk néhány bitcoin felhasználót és a segítségükkel vizsgáljuk meg, hogy melyek a bitcoin leggyakoribb felhasználási módjai. Az egész könyvben ezekkel a történetekkel fogjuk szemléltetni, hogy hogyan használható a digitális pénz a valós életben, és a bitcoin részét képező különféle technológiák hogyan teszik mindezt lehetővé.

Észak-amerikai kiskereskedeleml

Alíz Észak-Kaliforniában, a Bay Areán él. A bitcoinról műszaki érdeklődésű barátaitól hallott, és szeretné elkezdeni a használatát. Őt követjük majd, amint megisméri a bitcoint, vesz egy keveset belőle, majd elkölt belőle valamennyit, hogy vegyen egy csésze kávét Bob Palo Alto-i kávézójában. Ennek a történetnek a révén fogunk megismerni egy vevő szemszögéből a szoftverrel, a pénzváltókkal és a legegyszerűbb tranzakciókkal.

Észak-amerikai nagykereskedeleml

Carol egy galéria tulajdonosa San Francisco-ban. Drága képeket árul bitcoinért. Az ő történetével fogjuk bemutatni a konszenzus elleni 51%-os támadás által jelentett veszélyt, mely a nagy értékű tételek eladásánál jelentkezik.

Offshore szerződéses szolgáltatások

Bob, a Palo Alto-i kávéház tulajdonosa egy új webhelyet szeretne. Egy indiai web fejlesztővel, Gopesh-sel kötött szerződést, aki Bangalore-ban él. Gopesh beleegyezett abba, hogy bitcoinban kapja a fizetését. Ebben a történetben azt fogjuk megvizsgálni, hogyan használható a bitcoin kiszervezésre, szerződéses szolgáltatásokra és nemzetközi banki átutalásra.

Jótékonysági adományok

Eugénia egy Fülöp-szigeti gyermek jótékonysági alap igazgatója. Nemrég bukkant rá a bitcoinra, és szeretné külföldi és hazai adományozók új csoportját elérni vele, hogy adományokat gyűjtsön. Eugénia a bitcoin használatával szeretne a szükséget szenvedő

területekre pénzt küldeni. Az ő története fogja bemutatni, hogyan lehet a bitcoinnal határonkon és pénznemeken átívelő módon globális adománygyűjtést szervezni, és hogyan használható a nyílt fókonyv a jótékonyiségi szervezetek átláthatósága érdekében.

Import/export

Mohammed elektronikai cikkeket importál Dubaiba. Arra akarja használni a bitcoint, hogy az USÁ-ból és Kínából elektronikai termékeket importáljon az Egyesült Arab Emirátusokba, és hogy felgyorsítsa az import termékek fizetési folyamatát. Ez a történet fogja bemutatni, hogyan használható a bitcoin a fizikai termékekkel kapcsolatos nemzetközi nagykereskedelmi tranzakciókban.

Bitcoin bányászat

Jing számítástechnikát tanul Shanghaiban. Jövedelemkiegészítés céljából épített egy bitcoin „bányász” platformot. Ez a történet a bitcoin „ipari” oldalát fogja bemutatni: milyen speciális berendezések biztosítják a bitcoint hálózatot és az új pénz létrejöttét.

Mindegyik fenti történet valós szereplőkön és valós iparágakon alapul. Ezek a szereplők jelenleg arra használják a bitcoint, hogy új piacokat, új iparágakat teremtsenek, és a globális közgazdasági kérdésekre újító megoldásokat adjanak.

Elindulás

Ha az olvasó szeretne csatlakozni a bitcoin hálózathoz, és szeretné megkezdeni a bitcoin használatát, akkor ehhez csupán egy alkalmazás letöltésére vagy egy web alkalmazás használatára van szükség. Mivel a bitcoin szabványos, a bitcoin kliensnek számos különféle megvalósítása van. Van egy „referencia implementáció” is, az ún. Satoshi kliens. Ez egy nyílt forráskódú projekt, amelyet egy fejlesztő csoport kezel, és a Satoshi Nakamoto által írt eredeti implementációból származik.

A bitcoin kliensek három fő típusa:

Teljes kliens

A teljes kliens vagy „teljes csomópont” egy olyan kliens, amely a bitcoin tranzakciók teljes történetét tárolja, kezeli a felhasználó pénztárcáját és a bitcoin hálózatban közvetlenül képes tranzakciók indítására. Hasonló ez egy önálló e-mail szerverhez, mert önállóan kezeli a protokoll összes vonatkozását, de nem függ semmilyen más szervertől vagy harmadik fél által nyújtott szolgáltatástól.

Pehelysúlyú kliens

A pehelysúlyú kliens tárolja ugyan a felhasználó pénztárcáját, de egy harmadik fél szolgáltatásaira támaszkodva éri el a bitcoin tranzakciókat valamint a bitcoin hálózatot. A pehelysúlyú kliensben nincs meg az összes tranzakció teljes másolata, emiatt a tranzakciók ellenőrzésekor egy harmadik félben kell megbízni. Hasonló ez

egy önálló e-mail klienshez, amely egy postaláda elérésekor egy mail szerverhez kapcsolódik, vagyis a hálózati kapcsolatot egy harmadik fél segítségével teremti meg.

Web kliens

A web kliensek web böngészőkön keresztül használhatók, és a felhasználó pénztárcáját egy harmadik fél tulajdonában lévő web szerveren tárolják. Hasonló ez a webmailhez, mert teljes egészében egy harmadik fél szerverére támaszkodik.

Mobil bitcoin

Az okostelefonokhoz írt mobil kliensek, például azok, melyek az Android rendszeren alapulnak, lehetnek teljes kliensek, pehelysúlyú kliensek, vagy akár web kliensek is. Némelyik mobil klienst egy web vagy asztali klienssel szinkronizálnak, ezáltal több platformos pénztárca valósul meg, amely több eszközön is használható, de közös pénzforrással rendelkezik.

A bitcoin kliens kiválasztása attól függ, hogy milyen mértékben szeretnénk ellenőrzést gyakorolni a pénzünk fölött. A legnagyobb fokú kontroll és függetlenség egy teljes klienssel valósítható meg, viszont a mentések és a biztonsági kérdések terhét a felhasználónak kell viselnie. A másik végletet a web kliens jelenti, mert könnyen üzembe állítható és használható, de a web kliens esetében megjelenik az a kockázat, hogy biztonságot és a felügyeletet a felhasználó és web szolgáltatás tulajdonosa együtt gyakorolja. Ha egy webes pénztárca szolgáltatója kompromittálódik, mint ahogy a múltban ez már sokszor megtörtént, akkor a felhasználó az összes pénzét elveszítheti. Ha viszont a felhasználónak teljes kliense van, de nem rendelkezik megfelelő mentésekkel, akkor egy számítógépes baleset miatt veszítheti el a pénzét.

Ebben a könyvben sokféle bitcoin kliens használatát fogjuk szemléltetni, a referencia implementációtól (a Satoshi klienstől) kezdve a webes pénztárcákig. Némelyik példához a referencia klienst kell majd használni, mert ez nyújt API-kat a pénztárcához, a hálózathoz és a tranzakciós szolgáltatásokhoz. Ha önök a bitcoin rendszer programozható interfész felületeit szeretnék használni, akkor a referencia kliensre lesz szükségük.

Gyors elindulás

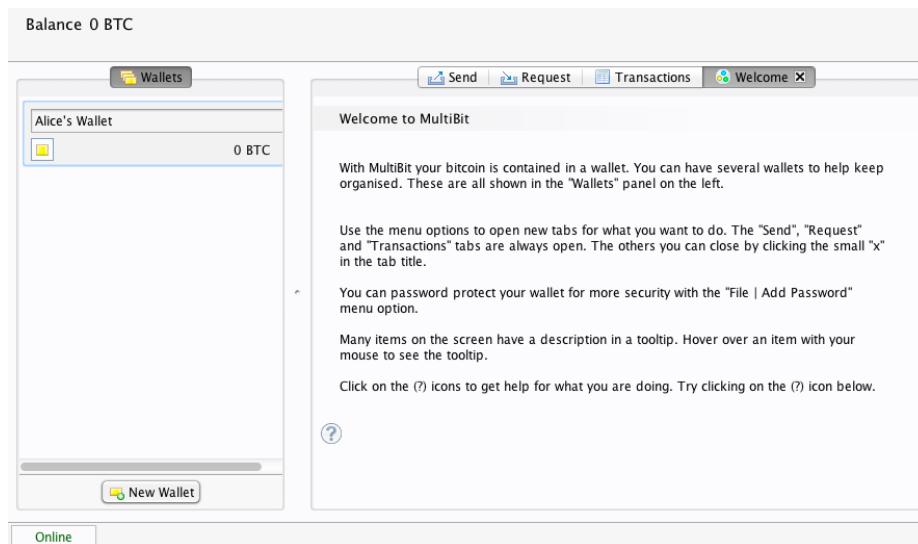
Alíz, akit [A bitcoin használata a felhasználók szemszögéből](#) részben ismerhettünk meg, nem műszaki felhasználó, és csak nemrég hallott a bitcoinról egy barátjától. Azzal kezdi az ismerkedést a bitcoinnal, hogy meglátogatja a [bitcoin.org](#) webhelyet. Itt a bitcoin kliensek széles választéka található meg. A bitcoin.org webhelyen lévő tanácsnak megfelelően a Multibit pehelysúlyú klienst választja.

Alíz a bitcoin.org webhelyen lévő hivatkozáson keresztül letölti a Multibitet az asztali gépére. A Multibit Windows, Mac és Linux számítógépen használható.



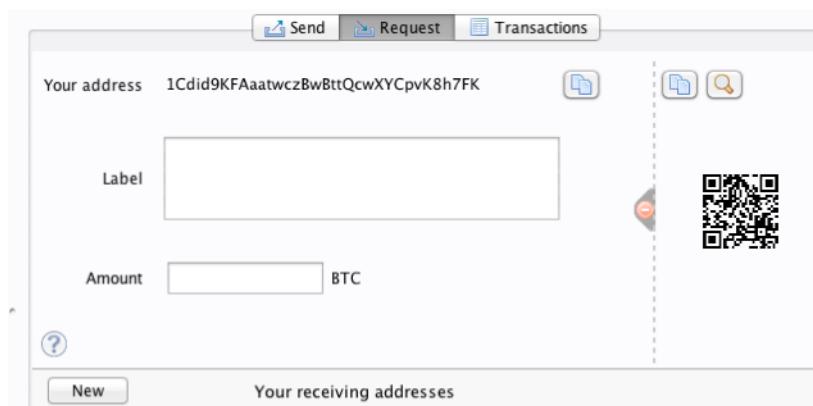
A bitcoin pénztárca létrehozásakor meg kell adni egy jelszót vagy jelmondatot, amely a pénztárca védelmére szolgál. Sok csibész próbálkozik a gyenge jelszavak feltörésével, ezért vigyázzon, hogy olyat válasszon, amelyet nem lehet könnyen feltörni. Használjon kis- és nagybetűkből, számokból és szimbólumokból álló kombinációt. Ne használjon személyes adatokat, pl. születési dátumokat, vagy focicsapatok nevét. Ne használjon szótárban előforduló szavakat. Ha teheti, használjon jelszó generátort, amely teljesen véletlenszerű jelszavakat állít elő. A jelszó hossza legalább 12 karakter legyen. Vésse az eszébe: a bitcoin pénz, és azonnal átutalható bárhová a világon. Kellő védelem hiányában a bitcoin könnyen ellopható.

Miután Alíz letöltötté és installálta a Multibit alkalmazást, elindítja azt. Egy „Üdvözlő” képernyő fogadja, amint azt az 1-1. ábra mutatja:



1-1.ábra: A Multibit bitcoin kliens üdvözlő képernyője

A Multibit automatikusan egy új pénztárcát és egy új bitcoin címet hoz létre Alíz számára. Az új cím a „Request” fülre történő kattintással tekinthető meg, amint azt az 1-2. ábra mutatja.



1-2. ábra: Alíz új bitcoin címe a Multibit kliens „Request” fulén

Ennek a képernyőnek a legfontosabb része Alíz *bitcoin címe*. A bitcoin cím egy e-mail címhez hasonlóan bárkivel megosztható. Segítségével bárki pénzt küldhet Alíz új pénztárcájába. A képernyőn a bitcoin cím egy betűkből és számokból álló hosszú karakterláncként jelenik meg: 1Cd1d9KFAaatwcxBwBttQcwXYCpvK8h7FK. A pénztárca bitcoin címe mellett van egy QR kód, amely egyfajta vonalkód, és ugyanezt az információt tartalmazza, de olyan formátumban, amely egy okostelefon fényképezőgépével könnyen bepásztázható. A QR kód az ablak jobb oldalán lévő, fekete és fehér kockákból álló kép. Alíz a bitcoin címet vagy QR kódot úgy tudja a „vágólapra” másolni, hogy rákattint a mellettük lévő gombra. Magára a QR kódra kattintva a kód kinagyítható, és egy okostelefonnal könnyen bepásztázható.

Alíz a QR kód kinyomtatásával könnyen meg tudja adni másoknak a címét, vagyis nincs szükség a betűk és számok hosszú sorozatának a begépelésére.



A bitcoin címek az '1' vagy a '3' számjeggyel kezdődnek. Egy e-mail címhez hasonlóan bármelyik másik bitcoin felhasználónak megadhatók. A bitcoin cím ismeretében a többi felhasználó bitcoinokat tud küldeni erre címre. Az e-mail címektől eltérően új címeket tetszőleges gyakran hozhatunk létre, és mindegyik közvetlenül a pénztárcához fog tartozni. A pénztárca egyszerűen címek és kulcsok gyűjteménye. A kulcsokkal lehet a pénztárcában lévő pénz pénz zárolását megszüntetni. A felhasználó által létrehozható bitcoin címek száma gyakorlatilag korlátlan.

Alíz új bitcoin pénztárcája ezzel készen áll a használatra.

Az első bitcoinok beszerzése

Bankokban vagy pénzváltó helyeken jelenleg nem lehet bitcoint venni. 2014-ben a legtöbb országban még mindig nehéz bitcoinhoz jutni. Számos speciális pénzváltó van, ahol a helyi valutáért bitcoin adásvétel végezhető. Ezek a pénzváltók webes pénzváltók. Ilyen többek között:

[Bitstamp](#)

egy európai pénzváltó, amely banki átutalással különböző pénznemeket támogat, többek között az Eurót (EUR) és az US dollárt (USD)

[Coinbase](#)

egy amerikai székhelyű tárca szolgáltató, amely a kereskedők és a vevők közötti bitcoin tranzakciókat támogatja. A Coinbase megkönnyíti a bitcoin adás-vételt, mert a felhasználók az ACH (Automated Clearing House) rendszeren keresztül az amerikai folyószámlájukhoz tudnak kapcsolódni.

Az ilyen digitális pénzváltó helyek a helyi pénznemek és a digitális pénzek metszéspontjában üzemelnek. Mint ilyenek, az adott terület helyi pénznemét használják, a nemzeti és nemzetközi szabályok hatásköre alá tartoznak, és gyakran egyetlen egy ország vagy egyetlen gazdasági övezet specialitásaihoz alkalmazkodnak. Egy pénzváltó használata függ az általunk használt pénznemtől, és hogy országunk jogrendszere szerint legalis-e a váltó. A fenti

szolgáltatóknál több napba vagy hétre telhet egy számla létesítése, hasonlóan egy bankszámla megnyitásához, mert különféle azonosító nyomtatványok kitöltésére van szükség a KYC (Know Your Customer, ismerd ügyfeledet) és AML (Anti-Money Laundering, pénzmosás elleni) banki szabályozásnak történő megfelelés miatt. Ha már rendelkezünk számlával egy bitcoin váltóban, akkor épp úgy kereshetünk a bitcoinokkal, mint ahogy azt egy külföldi devizával tennénk egy bróker számlán.

Részletesebb lista található a <http://bitcoincharts.com/markets/> helyen, amely több tucat pénzváltó árait és más piaci adatait tartalmazza.

Egy új felhasználó négy másik módszerrel tud még bitcoint szerezni:

- Keres egy barátot, akinek van bitcoinja, és közvetlenül tőle vesz. Sok bitcoin felhasználó kezde így.
- A localbitcoins.com-hoz hasonló szolgáltatással keres egy területileg közel eladót, akitől személyes tranzakció során, pénzért veszi meg a bitcoint.
- Valamilyen árut vagy szolgáltatást ad el bitcoinért. Egy programozó a programozói tudását tudja így értékesíteni.
- Keres egy bitcoin ATM-et. A bitcoin ATM-ek térképe a [CoinDesk](#) helyen található.

Alíz a barátja ismertette meg a bitcoinnal, ezért Alíz könnyen hozzájutott az első bitcoinjához, miközben arra várta, hogy a californiai pénzváltóban ellenőrizzék és aktiválják a számláját.

Bitcoin küldés és fogadás

Alíz létrehozott egy bitcoin pénztárcát, és most készen áll a pénz fogadására. A pénztárca alkalmazás generált neki egy véletlenszerű bitcoin címet és a hozzá tartozó kulcsot (ami egy elliptikus görbe privát kulcsa, és részletesebben a [Titkos kulcsok](#) rész ismerteti). Ebben a fázisban Alíz bitcoin címét a bitcoin hálózat még nem ismeri, a cím a bitcoin rendszer semelyik részében sincs „regisztrálva”. Alíz bitcoin címe egyszerűen csak egy szám, amely megfelel annak a kulcsnak, amellyel hozzá tud férfi a pénzéhez. Alíznak nincs semmilyen számlaszáma, és nincs semmilyen kapcsolat Alíz ezen címe és egy számla között. Mindaddig, amíg a bitcoin főkönyv (blokklánc) a benne tárolt tranzakción keresztül nem hivatkozik erre a címre, és a címre még nem küldtek pénzt, addig a cím egyszerűen csak egy a hatalmas számú lehetséges „érvényes” bitcoin cím közül. Ha a cím már kapcsolatba került egy tranzakcióval, akkor a hálózatban ismert címek egyike lesz, és bárki lekérdezheti a címhez tartozó egyenlegeit a publikus főkönyvből.

Alízzal barátja, Joe ismertette meg a bitcoint. Alíz egy helyi étteremben találkozik vele, hogy pár dollárért bitcoin vegyen tőle. Alíz kinyomtatva elhozta a bitcoin pénztárcája által megjelenített bitcoin címet és a QR kódöt. Biztonsági szempontból a bitcoin cím nem érzékeny adat. Bárhol feltehető anélkül, hogy biztonsági kockázatot jelentene.

Alíz csupán 10 dollárt szeretne bitcoinra váltani, mert nem szeretne túl sok pénzt kockáztatni ezzel az új technológiával. Átad Joe-nak egy 10 dolláros bankjegyet és a kinyomtatott címét, hogy Joe elküldhesse neki az ennek megfelelő összeget bitcoinban.

Joe-nak meg kell állapítania a váltási árfolyamot, hogy a helyes bitcoin összeget utalhassa át Alíznak. Alkalmazások és web helyek százai foglalkoznak az aktuális piaci árral. Íme, a legnépszerűbbek:

[Bitcoin Charts](#)

a bitcoincharts.com szolgáltatás a földgolyó számos pénzváltó helyének piaci adatait tartalmazza, a helyi pénznembe átszámítva

[Bitcoin Average](#)

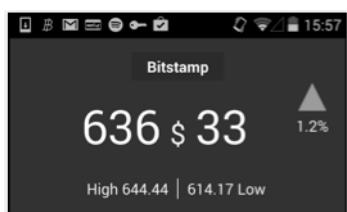
a bitcoinaverage.com, amely mindegyik pénznemre vonatkozóan tartalmaz egy forgalommal átlagolt árat

[ZeroBlock](#)

egy ingyenes Android és iOS alkalmazás, amellyel a különféle pénzváltó helyek bitcoin árai jeleníthetők meg (lásd az 1-3. ábrát)

[Bitcoin Wisdom](#)

egy másik, piaci adatokat megjelenítő alkalmazás.



1-3. ábra: A ZeroBlock – a bitcoin piaci árát mutató alkalmazás Androidra és iOS-re

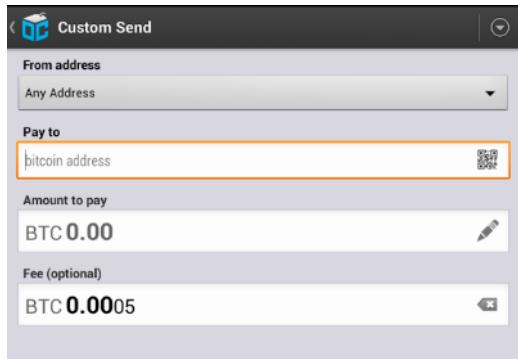
A fenti alkalmazások vagy webhelyek valamelyikével Joe meghatározza a bitcoin árát, amely történetesen kb. 100 US dollár / bitcoin. Ilyen árfolyamon 0.10 bitcoint (100 milliBitcoint) kell Alíznak elküldenie azért a 10 dollárért, amit tőle kapott.

Miután Joe megállapította a helyes átváltási árat, megnyitja a mobil pénztárca programját, és a bitcoin „küldés”-t választja. Például, ha a Blockchain mobil pénztárcát használja egy Android telefonon, akkor egy olyan képernyő jelenik meg, melynek két bemenő mezője van, amint azt a 1-4. ábra mutatja:

- a tranzakció címzettjének bitcoin címe
- a küldendő bitcoinok mennyisége

A bitcoin cím beviteli mezőben van egy kis ikon, amely úgy néz ki, mint egy QR kód. Ezzel Joe az okostelefonja fényképezőgépének be tudja pásztázni Alíz QR kódját, vagyis nem kell Alíz bitcoin címét begépeleznie (`1Cd1d9KFAaaatwczBwBttQcwXYCpvK8h7FK`), ami hosszú, és nehézkes folyamat. Joe megéríti a QR kód ikont, majd aktivizálja az okostelefon fényképezőgépét, és bepásztázza a QR kódot. A mobil pénztárca alkalmazás kitöltheti a bitcoin

címet, Joe pedig ellenőrzi, hogy helyes volt-e a pásztázás: összehasonlítja a bepásztázott cím néhány karakterét az Alíz által kinyomtatott címmel.



1-4. ábra: Mobil bitcoin pénztárca – a bitcoin küldési képernyő

Ezután Joe beadja a tranzakcióhoz tartozó bitcoin értéket, 0.10 bitcoint. Gondosan ellenőrzi az értéket, hogy helyes-e, mivel pénzről van szó, és egy hiba sokba kerülhet. Végül megnyomja a „Küldés” gombot, ekkor továbbításra kerül a tranzakció. Joe mobil bitcoin pénztárcája létrehoz egy tranzakciót, amely Joe pénzéből 0.10 bitcoin az Alíz által megadott címhez rendel hozzá, majd Joe privát kulcsával aláírja a tranzakciót. A bitcoin hálózat ebből tudja, hogy Joe a saját bitcoin címeinek valamelyikéről az adott értéket Alíz új címére szeretné továbbítani, és erre meghatalmazást adott. Amint a tranzakció továbbításra kerül a peer-to-peer protokollal, gyorsan szétterjed a bitcoin hálózatban. A hálózat legjobban kapcsolódó csomópontjai egy másodpercen belül megkapják a tranzakciót, és először találkoznak Alíz címével.

Ha Alíznak van egy okostelefonja vagy notebookja, akkor szintén látni fogja a tranzakciót. A bitcoin főkönyve egy állandóan növekvő állomány, amelyben minden, valaha előfordult bitcoin tranzakció rögzítve van. A bitcoin főkönyve publikus, ami azt jelenti, hogy Alíznak csupán meg kell néznie a főkönyvben, hogy érkezett-e a címére valamilyen pénz. Alíz ezt a blockchain.info webhelyen egészen egyszerűen megteheti, ha a kereső dobozba beadja a címét. A webhely által megjelenített lapon (<https://blockchain.info/address/1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK>) a címre vonatkozó összes bemenő és kimenő tranzakció szerepel. Ha Alíz azt követően, hogy Joe a megnyomta a „Küldés” gombot, megnézi ezt a lapot, hamarosan egy új tranzakció jelenik meg rajta, amely a 0.10 bitcoin ír jóvá a számláján.

Megerősítések

Alíz címén a Joe-tól jövő tranzakció először „Megerősítlen”-ként fog megjelenni. Ez azt jelenti, hogy a tranzakció már szétterjedt a hálózatban, de még nincs befoglalva a tranzakciós főkönyvbe, más néven a blokkláncba. Ahhoz, hogy a tranzakció befoglalásra kerülhessen, egy bányásznak „ki kell választania”, és bele kell foglalnia a tranzakciót egy tranzakciókból álló blokkba. Ha létrejött egy új blokk (amihez kb. 10 percre van szükség), akkor a blokkban lévő tranzakciókat a hálózat „Megerősített”-nek tekinti, és a tranzakciók elkölnhetők. Az egyes tranzakciókat mindenki azonnal látja, de csak akkor „bíznak meg” bennük, ha már be vannak foglalva egy újonnan kibányászott blokkba.

Alíz ezzel 0.10 bitcoin büszke tulajdonosa lett, melyet elkölthet. A következő fejezetben megnézzük, hogy mit fog venni Alíz a bitcoinjáért, részletesebben megvizsgáljuk a vásárlás mögött álló tranzakciót és a szétterjedés mögött álló technológiákat.

2. A bitcoin működése

Tranzakciók, blokkok, bányászat és a blokklánc

A szokásos banki és pénzügyi rendszerektől eltérően a bitcoin a decentralizált bizalomra épül. A bitcoin esetében a bizalom nem egy központi szervezet révén jön létre, melyben mindenki megbízik, hanem a bizalom a rendszer különböző résztvevői közötti interakciók eredményeképpen előálló tulajdonság. Ebben a fejezetben oly módon szerünk áttekintő képet a bitcoinról, hogy végigkövetjük egy tranzakciónak a rendszeren belüli útját, és megvizsgáljuk, hogy az elosztott közmegegyezés (konszenzus) módszere révén hogyan válik a tranzakció „megbízhatóvá”, és végül hogyan kerül rögzítésre az összes tranzakciót tartalmazó elosztott főkönyvben, vagyis a blokkláncban.

Mindegyik példa a bitcoin hálózat egy tényleges tranzakcióján alapul, és a felhasználók (Joe, Alíz és Bob) közötti történéseket szimulálja oly módon, hogy az egyik pénztárcából a másikba küld pénzt. Miközben a bitcoin hálózatban és a blokkláncon végigkövetjük a tranzakciókat, az egyes lépések megjelenítésére egy *blocklánc explorer* fogjuk használni. A blokklánc explorer egy olyan web alkalmazás, amely egy bitcoinos keresőgépként funkcionál, és lehetővé teszi, hogy a segítségével címek, tranzakciók és blokkok után kutassunk, és megvizsgáljuk a közöttük lévő összefüggéseket.

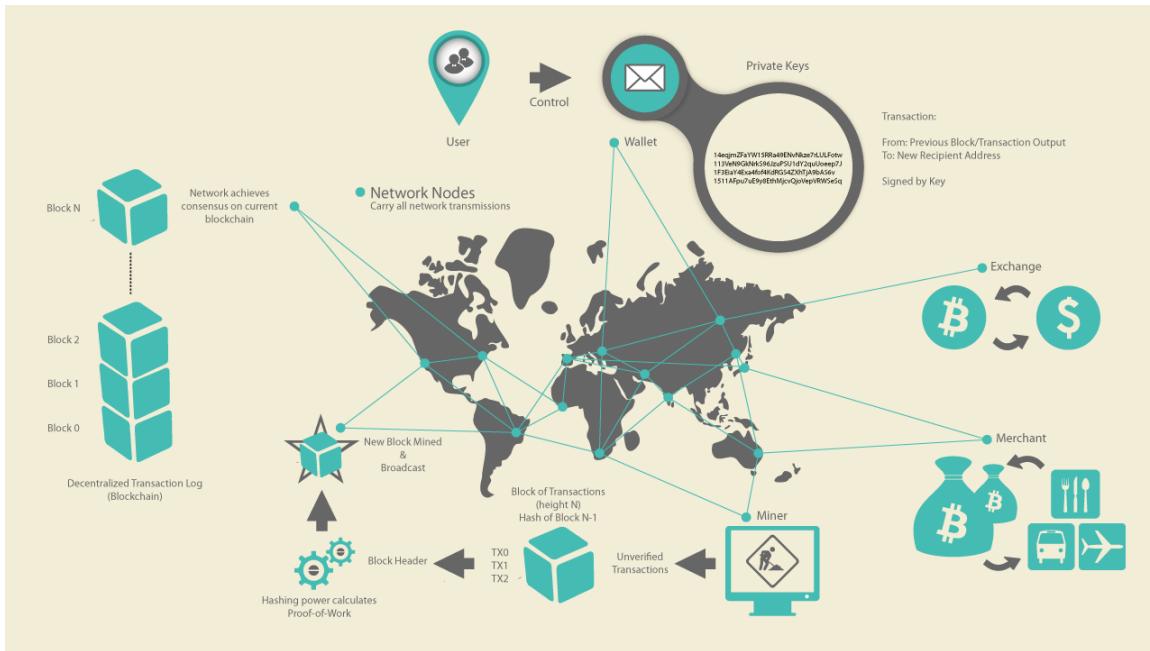
Népszerű blokklánc explorer többek között:

- [Blockchain info](#)
- [Bitcoin Block Explorer](#)
- [insight](#)
- [blockr Block Reader](#)

Mindegyikben van egy kereső funkció, amelynek egy cím, egy tranzakció hash (zanza) vagy egy blokkszám adható meg, és amely megkeresi az ennek megfelelő adatot a bitcoin hálózatban és a blokkláncon. Mindegyik példában egy URL-t is szerepelhetni fogunk, amely közvetlenül a megfelelő bejegyzésre mutat, ami ily módon részletesen tanulmányozható.

A bitcoin áttekintése

A 2-1. ábrán láthatjuk, hogy a bitcoin rendszer a következőkből áll: az egyes felhasználókhhoz tartozó, kulcsokat tartalmazó pénztárcákból, a hálózaton végigterjedő tranzakciókból, és bányászokból, akik a számításaik révén (egymással versenyezve) megteremtik a konszenzust a blokkláncon, amely az összes tranzakció hiteles tárháza. Ebben a fejezetben egy tranzakció útját kísérjük végig a hálózatban, és madártávlatból vizsgáljuk a bitcoin rendszer egyes részei közötti kölcsönhatásokat. A további fejezetek mélyebben belemennek a pénztárcák, a bányászat és a kereskedelmi rendszerek mögötti technológiába.



2-1. ábra: A bitcoin áttekintése

Vegyünk egy csésze kávét

Az előző fejezetben bemutatott új felhasználó, Alíz, épp most tett szert az első bitcoinjára. „[Az első bitcoin beszerzése](#)” részben Alíz találkozott a barátjával, Joe-val, hogy némi készpénzért bitcoin vásároljon tőle. A Joe által létrehozott tranzakció révén Alíz pénztárcájába 0.10 BTC került. Alíz első kiskereskedelmi tranzakciója az lesz, hogy vesz egy csésze kávét Bob Palo Alto-i kávézójában. Bob csak nemrég, elektronikus pénztárrendszerének modernizálása óta fogadja el a bitcoineket. Bob kávézójában az árak a helyi pénznemben (US dollárban) vannak feltüntetve, de a kasszánál az ügyfeleknek lehetőségük van dollárral vagy bitcoinnal fizetni. Alíz rendel egy csésze kávét, Bob pedig a kasszánál rögzíti a tranzakciót. A pénztárgép a végösszeget az aktuális árfolyam szerint US dollárról bitcoinra konvertálja, és minden pénznemben kijelzi az árat, továbbá megjelenít egy QR kódot, amely a tranzakcióhoz tartozó fizetés kérést tartalmazza (lásd a 2-2. ábrát):

Összeg:
\$1.50 USD
0.015 BTC



2-2. ábra: Fizetés kérést tartalmazó QR kód – próbálják meg bepásztázni!

A fizetés kérést tartalmazó fenti QR kód a BIP0021-ben definiált következő URL-t kódolja:

```
bitcoin:1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA?  
amount=0.015&  
label=Bob%27s%20Cafe&  
message=Purchase%20at%20Bob%27s%20Cafe
```

Az URL részei

Egy bitcoin cím: „1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA”

A fizetendő összeg: „0.015”

A bitcoin címhez tartozó címke: „Bob's Cafe” (Bob kávézója)

A fizetség leírása: „Purchase at Bob's Cafe” (Vásárlás Bob kávézójában)



A „fizetés kérés” nem csupán egy bitcoin címet tartalmazó QR kód, hanem egy QR kóddal kódolt URL, amely egy címet, a fizetendő összeget és egy általános leírást tartalmaz, pl. „Bob kávézója”. A bitcoin pénztárca ennek segítségével tudja összeállítani a fizetség elküldéséhez szükséges adatokat, és egyidejűleg a felhasználó számára olvasható formában megjeleníteni azokat. Ha bepásztázzuk a fenti QR kódot, akkor mi is azt látjuk, amit Alíz.

Bob azt mondja: „Ez egy dollár ötven, azaz tizenöt milliBitcoin lesz”.

Alíz az okostelefonjával bepásztázza a kijelzőn megjelenő QR kódot. A telefonon megjelenik a 0.0150 BTC kifizetés Bob kávézójának. Alíz a Küldés gombbal engedélyezi a kifizetést. Néhány másodpercen belül (körülbelül ugyanannyi idő alatt, mint amennyi egy hitelkártyás fizetéshez szükséges), Bob kasszáján megjelenik a tranzakció, és ezzel befejeződik a folyamat.

A következő részekben részletesebben meg fogjuk vizsgálni ezt a tranzakciót. Megnézzük, hogy Alíz pénztárcája hogyan hozza létre a tranzakciót, hogyan továbbítódik a tranzakció a hálózaton keresztül, hogy kerül ellenőrzésre, és végül hogyan tudja Bob további tranzakcióiban elkölni ezt az összeget.



A bitcoin hálózat képes kezelní a bitcoin tört részeit is, a millibitcoinoktól (ami a bitcoin 1/1'000 része) egészen a bitcoin 1/100'000'000 részéig, aminek Satoshi a neve. Ebben a könyvben az összeg nagyságától függetlenül a „bitcoin” szót fogjuk használni, a legkisebb egységtől (1 Satoshi) az összes, valaha kibányászásra kerülő (21'000'000) bitcoinig bezárolag.

Bitcoin tranzakciók

A tranzakció azt tudatja a hálózattal, hogy egy bitcoin tulajdonos engedélyezte bizonyos számú bitcoin átutalását egy másik tulajdonos számára. Ha az új tulajdonos el akarja költeni ezeket a bitcoinokat, akkor létrehoz egy újabb tranzakciót, amely engedélyezi az átutalást egy harmadik felhasználó számára, és így tovább, véges-végig egy tulajdonosi láncban.

A tranzakciók olyanok, mint egy kettős könyvelés főkönyvének a sorai. minden egyes tranzakcióból egy vagy több „bemenet” van, ami terhelést jelent egy bitcoin számlával szemben. A tranzakció másik oldalán egy vagy több „kimenet” van, ami jóváírásként hozzáadódik egy bitcoin számlához. A bemenetek és kimenetek (terhelések és jóváírások)

nem szükségszerűen ugyanazt a számot eredményezik, ha összeadjuk őket. A kimenetek összege kicsit kevesebb, mint a bemenetek összege, a különbség pedig egy hallgatólagos „tranzakciós díj”, vagyis egy kis fizetség, melyet az a bányász kap meg, amelyik a tranzakciót a főkönyvbe befoglalja. A 2-3. ábrán a tranzakciónk a főkönyvi nyilvántartás egy soraként jelenik meg.

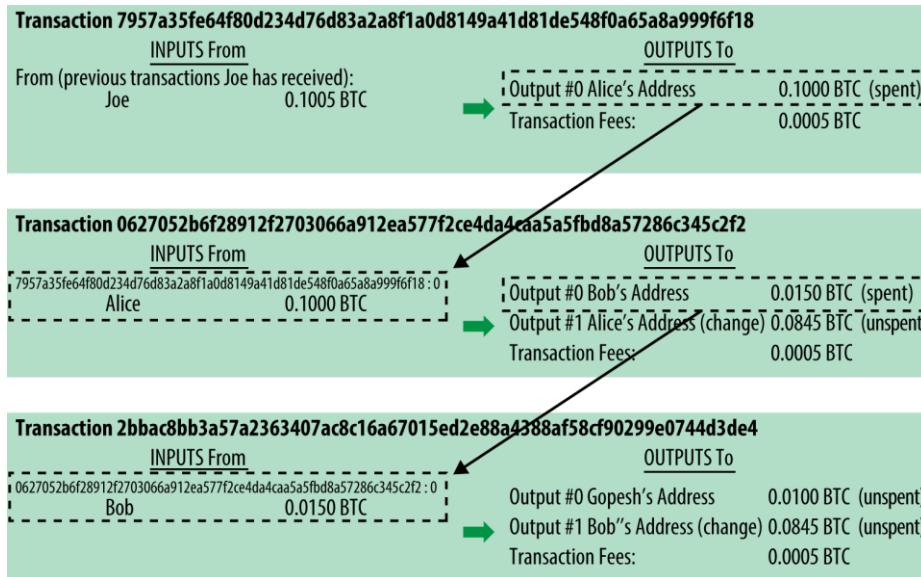
A tranzakciók minden egyes átutalásra kerülő (bemeneti) bitcoin összegre vonatkozóan bizonyítékot tartalmaznak arra vonatkozóan, hogy a tulajdonos valóban birtokolja őket. Ez a bizonyíték a tulajdonos digitális aláírása, amelyet bárki ellenőrizni tud. A bitcoin szóhasználatával élve, „elkölteni” valamit azt jelenti, hogy aláírunk egy tranzakciót, amely egy előző tranzakcióból egy új tulajdonosnak továbbít értéket. Az új tulajdonost a bitcoin címe azonosítja.



A tranzakciók az értéket a tranzakció bemeneteiből a tranzakció kimeneteibe továbbítják. A bemenet azt adja meg, hogy honnan származik az érték: általában egy előző tranzakció kimenetéből. Egy tranzakció kimenete új tulajdonost rendel az értékhez oly módon, hogy az értéket egy kulccsal rendeli össze. A cél kulcs neve: *akkadály*. Aláírási kötelezettséget jelent annak a számára, aki az összeget jövőbeli tranzakciókban szeretné használni. Egy tranzakció kimenetei egy új tranzakció bemeneteiként használhatók. Ily módon egy tulajdonosi lánc jön létre, amin az érték címről címre vándorol (lásd 2-4. ábra).

Transaction as Double-Entry Bookkeeping			
Inputs	Value	Outputs	Value
Input 1	0.10 BTC	Output 1	0.10 BTC
Input 2	0.20 BTC	Output 2	0.20 BTC
Input 3	0.10 BTC	Output 3	0.20 BTC
Input 4	0.15 BTC		
Total Inputs:	0.55 BTC	Total Outputs:	0.50 BTC
<i>Inputs</i>	<i>0.55 BTC</i>		
<i>Outputs</i>	<i>0.50 BTC</i>		
<i>Difference</i>	<i>0.05 BTC (implied transaction fee)</i>		

2-3. ábra: Tranzakciók kettős könyvelésként ábrázolva

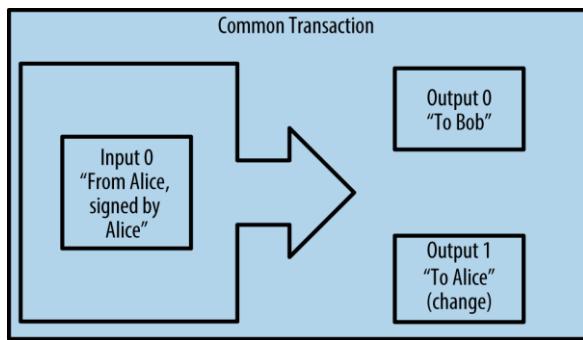


2-4. ábra: Egy tranzakciós lánc, ahol az egyik tranzakció kimenete alkotja a következő tranzakció bemenetét

Alíz a Bob kávéházának szóló fizetség során egy előző tranzakciót használ bemenetként. Az előző fejezetben Alíz a barátjától, Joe-tól készpénzért vett bitcoinot. Ez a tranzakció bizonyos számú bitcoin kötött hozzá (akkadályval) Alíz kulcsához. Alíz a Bob kávéháza számára létrehozott új tranzakció bemeneteként erre az előző tranzakcióról hivatkozik, és új kimeneteket hoz létre, a kávéért történő fizetség és a visszajáró pénz számára. A tranzakciók egy láncot alkotnak, amelyben a legutolsó tranzakciók bemenetei megfelelnek az előző tranzakciók kimeneteinek. Alíz kulcsa szolgáltatja azt az aláírást, amely felszabadítja az előző tranzakció kimeneteit, vagyis ily módon bizonyítja a bitcoin hálózat számára, hogy ő a pénzösszeg tulajdonosa. A kávéért történő fizetséget Bob címéhez rendeli hozzá, ezáltal „akkadályt állít” ezen a kimeneten, azzal a követelménnyel, hogy Bob aláírására van szükség, ha Bob szeretné elkölni ezt az összeget. Ez jelenti az érték továbbítást Alíz és Bob között. Az Alíz és Bob közötti tranzakciós láncot a 2-4. ábra szemlélteti.

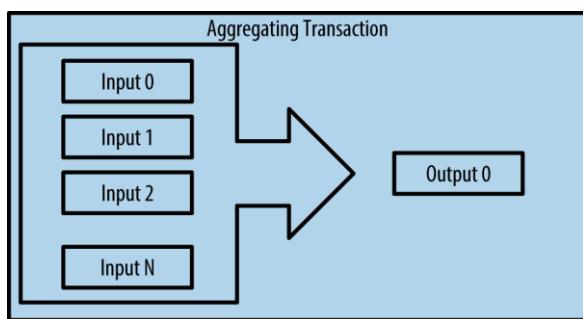
A leggyakrabban előforduló tranzakciók

A leggyakoribb tranzakció az egyik címről egy másik címre történő egyszerű fizetség, amely gyakran tartalmaz valamilyen „visszajáró” pénzt, mely az eredeti tulajdonoshoz jut vissza. Ennek a tranzakciótípusnak egy bemenete és két kimenete van, amint azt a 2-5. ábra mutatja:



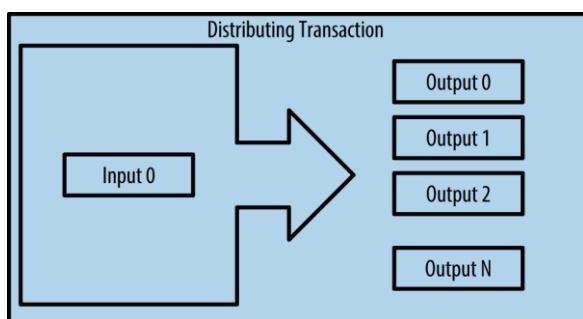
2-5. ábra: A leggyakoribb tranzakció

Egy másik, gyakori tranzakció több bemenetet egyetlen kimenetben összesít (lásd 2-6. ábra). Ez annak felel meg, amikor a valós világban egy csomó érméért és bankjegyért egyetlen nagyobb bankjegyet kapunk. A pénztárca alkalmazások néha azért hoznak létre ilyen tranzakciókat, hogy a számos kisebb visszajáró pénzt kitakarítsák.



2-6. ábra: Összegeket egyesítő tranzakció

Végül, a bitcoin főkönyv gyakori tranzakció típusa az is, amely egyetlen bemenetet több kimenettként oszt fel, ahol a kimenetek különböző személyekhez tartoznak (lásd 2-7. ábra). Ezt a tranzakciótípust az üzleti vállalkozások pénz elosztásra használják, pl. amikor egy fizetési lista alapján több alkalmazottnak küldenek fizetést.



2-7. ábra: Pénz elosztó tranzakció

Egy tranzakció létrehozása

Alíz pénztárca programja a megfelelő bemenetek és kimenetek kiválasztásával az Alíz előírásának megfelelő tranzakciót hozza létre. Alíznak csak a célszemélyt és az összeget kell megadnia, a többi a pénztárca program automatikusan elvégzi anélkül, hogy Alíznak törödnie kellene a részletekkel. Fontos, hogy egy pénztárca program még akkor is képes tranzakciók létrehozására, ha teljesen offline állapotú. Hasonlóan ahhoz, ahogy egy otthon megírt csekket is el lehet küldeni egy borítékban a banknak, egy tranzakció létrehozása és aláírása sem követeli meg, hogy a program kapcsolatban legyen a bitcoin hálózattal. A hálózatnak csak a legvégén kell a tranzakciót elküldeni, hogy megtörténhessen a végrehajtása.

A megfelelő bemenetek kiválasztása

Alíz pénztárca programjának először olyan bemeneteket kell találnia, amelyekkel lehetséges a Bobnak küldendő összeg kifizetése. A legtöbb pénztárca program egy kis adatbázist hoz létre az „el nem költött tranzakció kimenetek”-ből, melyek a pénztárca saját kulcsaival vannak zárolva („akkadályoztatva”). Ennek megfelelően, Alíz pénztárcájában ott lesz Joe tranzakciójából annak a kimenetnek a másolata, amely akkor jött létre, amikor Alíz bitcoint vett Joe-tól (lásd „[Az első bitcoin beszerzése](#)” részt). Azoknak a bitcoin pénztárca alkalmazásoknak, melyek teljes kliensként futnak, másolatuk van a blokklánc összes tranzakciójának elköltetlen kimeneteiről. Ez lehetővé teszi, hogy a pénztárca program tranzakció bemeneteket hozhasson létre, valamint hogy gyorsan ellenőrizze, hogy a bejövő tranzakcióknak helyesek-e a bemenetei. Mivel egy teljes kliens sok diszk helyet foglal, a legtöbb felhasználó „pehelysúlyú” klienseket futtat. Ezek a kliensek csak a felhasználó saját, el nem költött kimeneteit tartják nyilván.

Ha a pénztárca programban nincs meg az összes elköltetlen tranzakciós kimenet másolata, akkor a program a bitcoin hálózatból le tudja kérdezni ezt az adatot, vagy úgy, hogy a különféle szolgáltatók API-jait használja, vagy egy teljes csomópont bitcoin kliensének JSON RPC API-n keresztsüli lekérdezésével. A 2-1. példa egy RESTful API kérési példát szemléltet, melyet egy adott URL-re kiadott HTTP GET kéréssel hoztunk létre. Az URL visszaadja, hogy egy adott címhez milyen az el nem költött tranzakciós kimenetek tartoznak, vagyis megadja azokat az adatokat, melyek ahhoz szükségesek, hogy egy alkalmazás létrehozza a kimenetek elköltéséhez szükséges tranzakció bemeneteket. Egy parancssorból futtatható, egyszerű *cURL* HTTP klienssel kapjuk meg a választ:

2-1. példa: Az Alíz bitcoin címéhez tartozó el nem költött kimenetek megkeresése

```
$ curl  
https://blockchain.info/unspent?active=1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK
```

2-2. példa: A keresésre kapott válasz

```
{  
  
    "unspent_outputs": [  
  
        {  
            "tx_hash": "186f9f998a5...2836dd734d2804fe65fa35779",  
            "tx_index": 104810202,  
            "tx_output_n": 0,  
  
            "script": "76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",  
            "value": 10000000,  
            "value_hex": "00989680",  
            "confirmations": 0  
        }  
  
    ]  
}
```

A választ a 2-2. példa mutatja. Eszerint a bitcoin hálózat egyetlen egy el nem költött kimenetről tud (amely még nem lett felhasználva), és ez Alíz 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK címéhez tartozik. A válasz egy hivatkozást tartalmaz arra a tranzakcióra, amelyben ez az el nem költött kimenet (a Joe-tól érkező pénz) van. A kimenet értéke Satoshiban van megadva, a 10 millió Satoshi 0.10 bitcoinnak felel meg. Ezen információ birtokában Alíz pénztárca alkalmazása létre tud hozni egy tranzakciót, amely ezt az értéket az új tulajdonosok címeire továbbítja.



Lásd [Joe tranzakciója Alíz számára](#).

Mint látható, Alíz pénztárcájában elegendő bitcoin van az egyetlen egy el nem költött kimenetben ahhoz, hogy kifizesse a kávéját. Ha nem ez lenne a helyzet, akkor a pénztárca programnak „végig kellene bogarásznia” egy halom kisebb, el nem költött kimenetet, ahhoz hasonlóan, mint ha valaki egy fizikai pénztárcából újabb és újabb pénzéréméket venne elő, hogy ki tudja fizetni a kávéját. Mindkét esetben kezelni kell a visszajáró pénzt is. Ezt a következő részben fogjuk látni, amikor a pénztárca alkalmazás létrehozza a tranzakció kimeneteket (a kifizetések). (payments).

A kimenetek létrehozása

A tranzakció kimenete egy script formájában jön létre. Ez a script egy akadályt hoz létre, és az összeg csak úgy használható fel, ha a scripthez valaki ismeri a megoldást. Egyszerűbb szavakkal, az Alíz által létrehozott tranzakció kimenetében egy olyan script lesz, ami ezt mondja: „Ez a kimenet annak fizethető ki, aki be tud mutatni egy olyan aláírást, amely Bob

nyilvános címéhez tartozó kulccsal történt.” Mivel az a kulcs, amely ehhez a címhez tartozik, csak Bob pénztárcájában van meg, ezért csak Bob pénztárcája képes ilyen aláírásra, és ily módon a kimenet elköltésére. Alíz tehát azzal, hogy aláírást kér a Bobtól, „megakadályozza”, hogy más is elkölthesse a kimenet értékét.

A tranzakciónak lesz egy második kimenete is, mivel Alíz pénze egy 0.10 BTC értékű kimenetben áll rendelkezésre, ami túl sok a 0.015 BTC-be kerülő kávéért. Alíznak 0.085 BTC visszajár. A visszajáró pénzt Alíz pénztárca programja kezeli, ugyanabban a tranzakcióban, amelyben a Bobnak történő kifizetést. Lényegében Alíz pénztárcája a pénzt két kifizetésre bontja: egy Bobnak történő kifizetésre és egy saját magának történő visszafizetésre. Alíz a visszajáró pénzhez tartozó kimenetet egy későbbi tranzakcióban tudja felhasználni, vagyis el tudja majd költeni.

Végül, annak érdekében, hogy a hálózat gyorsan feldolgozza a tranzakciót, Alíz pénztárca programja egy kis díjat alkalmaz. A díj a tranzakcióban nem jelenik meg explicit módon, hanem a bemenetek és kimenetek különbsége. Ha Alíz a második kimenetben 0.085 helyett csak 0.0845-öt ad meg, akkor 0.0005 BTC (fél millibitcoin) marad. A bemenet 0.10 BTC-jét a két kimenet nem költi el teljesen, mivel a kimenetek összege kisebb lesz, mint 0.10. Az így keletkező különbség a tranzakciós díj, amely azé a bányászé lesz, aki a tranzakciót blokkba foglalja és a blokkot a blokkláncjal megalvájtott főkönyvben tárolja.

A bitcoin blokkláncban lévő tranzakció a következő URL-lel íratható ki, amint azt a 2-8. ábra mutatja:

Transaction View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2	
1Cd1d9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)	 1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA - (Unspent) 0.015 BTC 1Cd1d9KFAaatwczBwBttQcwXYCpvK8h7FK - (Unspent) 0.0845 BTC
	97 Confirmations 0.0995 BTC
Summary	Inputs and Outputs
Size 258 (bytes)	Total Input 0.1 BTC
Received Time 2013-12-27 23:03:05	Total Output 0.0995 BTC
Included In Blocks 277316 (2013-12-27 23:11:54 +9 minutes)	Fees 0.0005 BTC
	Estimated BTC Transacted 0.015 BTC

2-8. ábra: Alíz Bob kávézójával kapcsolatos tranzakciója



Alíz Bob kávézójával kapcsolatos tranzakciója a következő hivatkozás segítségével érhető el: [Alíz tranzakciója Bob kávéháza számára](#).

A tranzakció hozzáadása a nyilvántartáshoz

Alíz pénztárca programja egy 258 bajt hosszú tranzakciót hozott létre. A tranzakció minden tartalmaz, ami az összeg felett tulajdonjog bizonyításához szükséges, és az összeget egy új

tulajdonoshoz rendeli hozzá. Ez az a pont, amikor a tranzakciót el kell küldeni a bitcoin hálózatba, ahol az be fog épülni az elosztott nyilvántartásba, a blokkláncba. A következő részben látni fogjuk, hogyan válik a tranzakció egy új blokk részévé, és hogyan történik az új blokk „kibányászása”. Végül látni fogjuk, hogy miután az új blokk a blokklánc részévé vált, hogyan lesz a blokk egyre megbízhatóbb, amint a blokklánc egyre több blokkal bővül.

A tranzakció elküldése

Mivel a tranzakció tartalmazza a feldolgozásához szükséges összes információt, nem számít, hogyan vagy honnan küldjük el a bitcoin hálózatba. A bitcoin hálózat egy egyenrangú csomópontkból álló, ún. peer-to-peer hálózat, amelyben az egyes bitcoin kliensek számos más bitcoin klienshez kapcsolódnak. A bitcoin hálózat célja az, hogy az összes résztvevőnek továbbítsa a tranzakciókat és a blokkokat.

A tranzakció szétterjedése

Alíz pénztárca programja az új tranzakciót tetszőleges bitcoin kliensnek el tudja küldeni, ha azzal valamilyen Internet kapcsolata van. A kapcsolat lehet vezetékes, WiFi vagy mobil. Szükségtelen, hogy Alíz bitcoin pénztárcája Bob bitcoin pénztárcájával közvetlen kapcsolatban legyen, vagy hogy a kávéházban lévő Internet kapcsolatot használja, bár minden dolog lehetséges. Egy tetszőleges bitcoin hálózati csomópont (vagyis egy másik kliens), amely egy előzőleg még nem látott érvényes tranzakcióval találkozik, azonnal továbbítja azt vele kapcsolatban lévő többi csomópontnak. Emiatt a peer-to-peer hálózatban a tranzakció gyorsan szétterjed, és a csomópontok nagy részéhez néhány másodpercen belül eljut.

Hogyan látja mindezt Bob

Ha Bob bitcoin pénztárca programja közvetlenül Alíz pénztárca programjával van kapcsolatban, akkor Bob kliense lesz az első, amelyik a megkapja a tranzakciót. De ha Alíz pénztárcája más csomópontokon keresztül küldi el a tranzakciót, a tranzakció akkor is néhány másodpercen belül eljut Bob pénztárcájához. Bob pénztárcája Alíz tranzakcióját azonnal egy bejövő fizetésnek fogja látni, mivel egy olyan kimenetet tartalmaz, amely Bob kulcsával elkölhető. Bob pénztárca programja azt is ellenőrizni tudja, hogy a tranzakció jól formált-e, előzőleg elköltetlen bemeneteket használ-e és kellő nagyságú tranzakciós díjat tartalmaz-e ahhoz, hogy a befoglalják a következő blokkba. Ezek után Bob viszonylag kis kockázattal feltételezheti, hogy a tranzakció blokkba foglalása és megerősítése hamarosan megtörténik.



A bitcoin tranzakciókkal kapcsolatos gyakori félreértés az, hogy 10 percet kell várni a tranzakció „megerősítéséhez”, vagyis amíg bele nem kerül egy új blokkba, vagy 60 percet 6 teljes megerősítéshez. Noha a megerősítés biztosítja, hogy a tranzakciót az egész hálózat ugyanolyannak lássa, az olyan kis értékű tételek esetén, mint egy csésze kávé, felesleges a várakozás. Egy érvényes, kis értékű tranzakció megerősítés nélküli elfogadása nem jár nagyobb kockázattal, mint egy hitelkártyával történő fizetés azonosító okmány vagy aláírás nélküli elfogadása, márpedig ez gyakori manapság.

Bitcoin bányászat

A tranzakció tehát szétterjedt a bitcoin hálózatban. Addig azonban nem lesz az osztott főkönyv (a *blokklánc*) része, amíg egy *bányászatnak* nevezett folyamat le nem ellenőrzi és be nem foglalja egy blokkba. Részletesebb magyarázat a [8. fejezetben](#) található.

A bitcoin rendszer a bizalmat elvégzett számításokra alapozza. A tranzakciókat *blokkokba* rendezi, amihez rendkívül sok számításra van szükség, de a blokkok ellenőrzéséhez kevésre. Ez a folyamat a bányászat, és a bitcoin esetében két célt szolgál:

- A bányászat révén minden egyes blokkban új bitcoinok jönnek létre, majdnem úgy, ahogy egy központi bank új pénzt nyomtat. A létrejövő bitcoinok mennyisége állandó, és idővel csökkenő.
- A bányászat hozza létre a bizalmat oly módon, hogy a tranzakciók csak akkor kerülnek megerősítésre, ha elég feldolgozó kapacitást fordítottak az öket tartalmazó blokkra. A több blokk több elvégzett számítást, vagyis nagyobb bizalmat jelent.

A bányászat olyasféle dolog, mint egy hatalmas sudoku játék, melyet egymással párhuzamosan játszanak, és amely mindig újra indul, ha valaki talál egy megoldást. A játék nehézségét automatikusan úgy választják meg, hogy körülbelül 10 perc legyen a megoldáshoz szükséges idő. Képzeljünk el egy hatalmas sudoku rejtvényt, melyben néhány ezer a sorok és oszlopok száma. Egy kész megoldás nagyon gyorsan ellenőrizhető. De ha a rejtvény még nincs kitöltve, akkor a megoldásához nagyon sok munkára van szükség! A sudoku bonyolultsága a méretének a módosításával szabályozható (mennyi legyen a sorok és az oszlopok száma), de még akkor is nagyon egyszerű az ellenőrzése, ha nagyon nagy. A bitcoinnál használt „rejtvény” a hash-képző titkosítási algoritmuson alapul, és hasonló jellemzőkkel rendelkezik: aszimmetrikusan nehéz a megoldása, de könnyű az ellenőrzése és a nehézsége állítható.

[„A bitcoin használata a felhasználók szemszögéből”](#) részben bemutattuk Jinget, aki számítástechnikát tanul Sanghajban. Jing bányászként működik közre a bitcoin hálózatban. Kb. 10 percentként Jing és sok ezer más bányász versenyre kel egymással, hogy megoldást találjon egy tranzakciókból álló blokkhoz. Az ilyen megoldás neve: „munkabizonyíték”. A megoldáshoz másodpercenként több trillió hash (zanza) műveletet kell a teljes bitcoin hálózatban elvégezni. A „munkabizonyíték” algoritmusá abból áll, hogy a blokk fejéből és egy véletlen számból az SHA256 titkosítási algoritmussal egy hasht (zanzát) képez, és ezt mindenkor ismétli, amíg létre nem jön egy előre meghatározott minta. Az adott körben az a bányász nyeri meg a versenyt, aki elsőként talál egy ilyen megoldást, és publikálja a blokkot a blokkláncon.

Jing 2010-ben kezdett bányászni. Egy gyors asztali számítógéppel kereste az új blokkokhoz a megfelelő munkabizonyítékot. Ahogy egyre több bányász csatlakozott a bitcoin hálózathoz, a megoldandó probléma nehézsége gyorsan nőtt. Jingnek és a többi bányásznak hamarasan speciálisabb hardverekre kellett áttérnie, pl. a játékokban vagy a konzolokban használt grafikus kártyákra (GPU, Graphical Processing Unit). Ennek a könyvnek az írása idején a nehézség már olyan magas, hogy csak ASIC-ekkel (ASIC, Application Specific Integrated Circuit → BOÁK, Berendezés Orientált Integrált Áramkör) kifizetődő a bányászat. Az ASIC-ekben sok száz hash-képző egység van hardverrel megvalósítva. Ezek egy szilícium morzsán, egymással párhuzamosan futnak. Jing csatlakozott egy „bányatársasághoz” is, ami egy lottázó közösségekkel hasonlóan lehetővé teszi, hogy a résztvevők egyesítsék az erőforrásait és

osztozzanak a jutalmon. Jing most napi 24 órában két, USB-vel rendelkező ASIC géppel bányászik. A villanszámláját úgy fizeti, hogy eladja a bányászattal előállított bitcoinokat, és még némi nyereségre is szert tesz. A számítógépen a bitcoind referencia kliens egy példánya fut, ami a specializált bányász szoftver futtatásához szükséges.

Blokkok létrehozása a tranzakciókból

A hálózatba elküldött tranzakció csak akkor kerül ellenőrzésre, ha bekerül a globális elosztott nyilvántartásba, a blokkláncba. A bányászok minden 10 percben egy új blokkot állítanak elő, amelyik az utolsó blokk óta előállt összes tranzakciót tartalmazza. A felhasználók pénztárcáiból és egyéb alkalmazásokból folyamatosan érkeznek a hálózatba az új tranzakciók. A bitcoin hálózat csomópontjai ezeket egy ellenőrizetlen tranzakciókból álló, átmeneti „poolba” (gyűjtőterületre) helyezik. A bányászok egy új blokk felépítésének a megkezdésekor az ellenőrizetlen tranzakciókat erről a területről egy új blokkhoz adják hozzá, majd megpróbálnak megoldani egy nagyon nehéz problémát (a munkabizonyítékot), hogy így bizonyítsák az új blokk érvényességét. A bányászat folyamatát részletesen a „[Bányászat és konszenzus](#)” fejezet ismerteti.

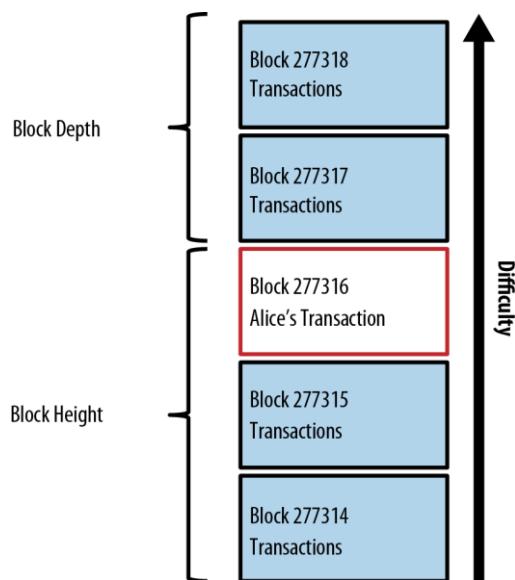
Azt, hogy mely tranzakciók kerülnek be a blokkba, a tranzakciós díj és néhány egyéb tényező befolyásolja. Mindegyik bányász egy új blokk bányászatához kezd, mielőtt megkapja a hálózattól az előző blokkot, mivel ebből tudja, hogy elvesztette a verseny előző fordulóját. Mindegyik bányász azonnal egy új blokkot hoz létre, feltölti tranzakciókkal és az előző blokk ujjlenyomatával, majd megkezdi az új blokkhoz a munkabizonyíték számításokat. Mindegyik bányász egy speciális tranzakciót foglal bele a blokkba, amely jutalomként újonnan előállított bitcoinokat (ez jelenleg 25 BTC blokkonként) fizet ki a bányász saját bitcoin címére. Ha a bányász talál egy megoldást, amely a blokkot érvényessé teszi, akkor „megnyeri” ezt a jutalmat, mivel a sikeresen létrehozott blokk a globális blokklánc részévé válik, és a blokkban lévő, jutalmat tartalmazó tranzakció elkölnhetővé válik. Jing, aki egy bányatársaság tagja, úgy állította be a szoftverét, hogy egy új blokk létrehozásakor a jutalom a bányatársaság címére kerüljön. Innen a jutalom egy részét a bányatársaság Jingnek és a többi bányásznak osztja szét, azzal arányosan, hogy ki-ki mennyi munkát végzett az utolsó körben.

Alíz tranzakciója szétterjedt a hálózaton, és bekerült az ellenőrizetlen tranzakciók pooljába. Mivel a tranzakcióban elégéges tranzakciós díj volt, a tranzakció bekerült a Jing bányatársasága által létrehozott új blokkba. Kb. 5 perccel azt követően, hogy a tranzakciót Alíz pénztárcája szétküldte, Jing ASIC bányagépe talált egy megoldást a blokkhoz, és a tranzakciót 419 másik tranzakcióval egyetemben a 277316. blokkban publikálta. A Jing által publikált új blokkot a többi bányász is ellenőrizte, majd egy újabb versenybe kezdett, hogy előállítsa a következő blokkot.

Az Alíz tranzakcióját tartalmazó blokk itt látható: [Alíz tranzakciója](#).

Néhány perccel ezután egy másik bányász egy újabb blokkot állított elő, a 277317-ik blokkot. Mivel ez a blokk az előző (277316.) blokkon alapul, amely tartalmazta Alíz tranzakcióját, a blokkban lévő számítások tovább erősítik az előző blokkban lévő tranzakciók iránti bizalmat. A tranzakciót tartalmazó blokk fölötti blokk egy újabb „megerősítést” jelent a tranzakció számára. Amint a blokkok egymásra halmozódnak, exponenciálisan egyre nehezebb a tranzakció megfordítása, emiatt egyre megbízhatóbbá válik.

A lenti 2-9. ábrán a 277316. blokkot láthatjuk, amely Alíz tranzakcióját tartalmazza. Alatta 277315 db blokk van, amely egy blokkláncként kapcsolódik egymáshoz, egészen a 0-ik blokkig visszamenőleg, amely az ún. *genезис blokk*. Idővel, ahogy a blokkok „magassága” egyre nő, úgy lesz a számítási nehézség az egyes blokkok és a lánc egész szempontjából is egyre nagyobb. Azok a blokkok, melyeket az Alíz tranzakcióját tartalmazó blokk után lettek kibányászva, további megerősítést jelentenek, mivel egy egyre hosszabb láncban egyre több és több számítást tesztenek meg. A tranzakciót tartalmazó blokk fölötti blokkok számítanak „megerősítésnek”. A 6-nál több megerősítéssel rendelkező blokkok visszavonhatatlannak tekinthetők, mivel 6 blokk érvénytelenítéséhez és újraszámításához hatalmas számítási kapacitásra lenne szükség. A bányászat folyamatát és szerepét a bizalom kialakulásában a [8. fejezetben](#) fogjuk részletesen megvizsgálni.



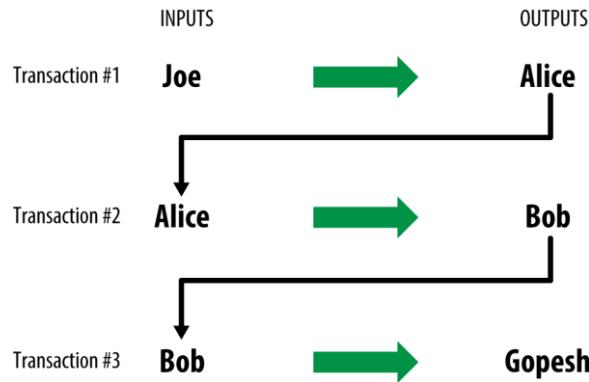
2-9. ábra: Alíz tranzakciója a 277316. blokkban

A tranzakció elköltése

Most, hogy Alíz tranzakciója egy blokk részeként be lett ágyazva a blokkláncba, része lett a bitcoin elosztott főkönyvének, és az összes bitcoin alkalmazás számára látható. Mindegyik bitcoin kliens külön-külön képes ellenőrizni, hogy a tranzakció érvényes és elkölthető-e. A teljes kliensek képesek nyomon követni a pénzmozgást attól a pillanattól kezdve, ahogy a bitcoinok először létrejöttek a blokkban, tranzakcióról, tranzakcióra, egészen addig, amíg el nem érnek Bob címéhez. A pehelysúlyú kliensek Egyszerűsített Fizetési Ellenőrzésre képesek (lásd [SPV], Simple Payment Verification), melynek során megállapítják, hogy a tranzakció része a blokkláncnak, és elég sok blokk lett-e már kibányászva utána, ami szavatolja, hogy a hálózat a tranzakciót érvényesnek tekinti (lásd „[SPV csomópontok](#)”).

Bob úgy tudja elkölteni ennek a tranzakciónak és egyéb tranzakcióknak a kimenetét, hogy létrehoz egy saját tranzakciót, amelynek bemenete ezekre a kimenetekre hivatkozik, és egy új tulajdonoshoz rendeli hozzá őket. Például Bob egy beszállítót úgy tud kifizetni, hogy Alíz kávéért történő fizetségét ennek az új tulajdonosnak utalja át. A legvalószínűbb eset az, hogy Bob bitcoin programja a sok kis fizetséget egy nagyobb fizetségen egyesíti, esetleg az egész napi bitcoin bevételt egyetlen egy tranzakcióba koncentrálja. A különféle befizetéseket ez a tranzakció egyetlen címre, a bolt általános „folyószámlájára” utalja. Az összesítő tranzakciókat lásd a 2-6. ábrán.

Amikor Bob elkölti az Alíztól és a többi ügyfélről kapott fizetséget, akkor ezzel a tranzakciós láncot bővíti, a tranzakció pedig hozzáadódik a blokkláncból álló globális nyilvántartáshoz, melyet mindenki lát, és amelyben mindenki megbízik. Tegyük fel, hogy Bob a web tervezőnek, Gopeshnek fizet egy új weblapért. Ekkor a tranzakciós lánc a következőképpen fog kinézni:



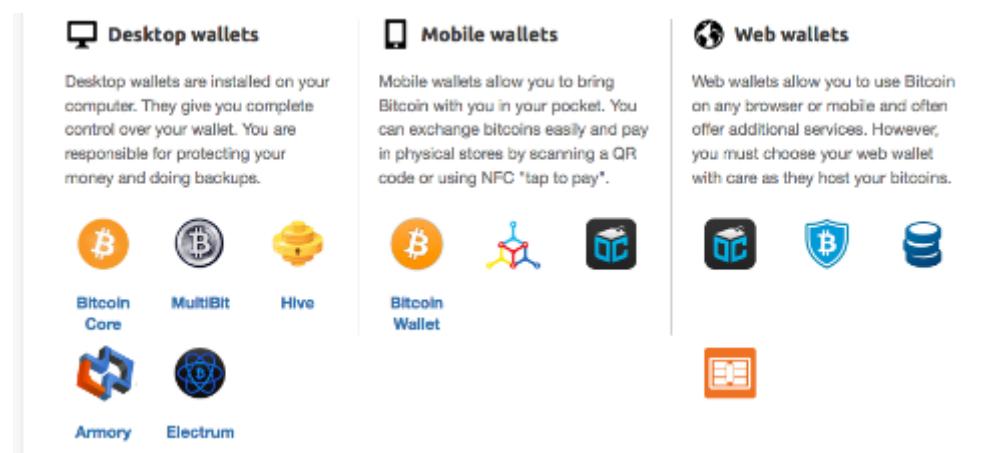
2-10. ábra: Alíz tranzakciója, mint a Joe-tól Gopeshig tartó tranzakciós lánc része

3. A bitcoin kliens

Bitcoin Core – a referencia implementáció

A *Bitcoin Core* referencia kliens, más néven a Satoshi kliens a bitcoin.org -ról töltethető le. A referencia kliens a bitcoin rendszer összes részét megvalósítja: van benne pénztárca, tranzakció ellenőrzés, mely a tranzakciós főkönyv (a blokklánc) teljes másolatára épül, és egy teljes értékű peer-to-peer hálózat csomópont.

A referencia kliens a <http://bitcoin.org/en/choose-your-wallet> web helyről, a „Bitcoin Core” választásával töltethető le. Az operációs rendszertől függően egy végrehajtható installáló program fog letölteni. A Windows esetén ez egy ZIP archívum vagy egy EXE végrehajtható program. A MAC OS esetén egy .dmg disk kép. A Linux változatok az Ubuntu esetén egy PPA csomagot vagy egy tar.gz archívumot tartalmaznak. Az ajánlott klienseket felsoroló bitcoin.org lap a 3-1. ábrán látható.

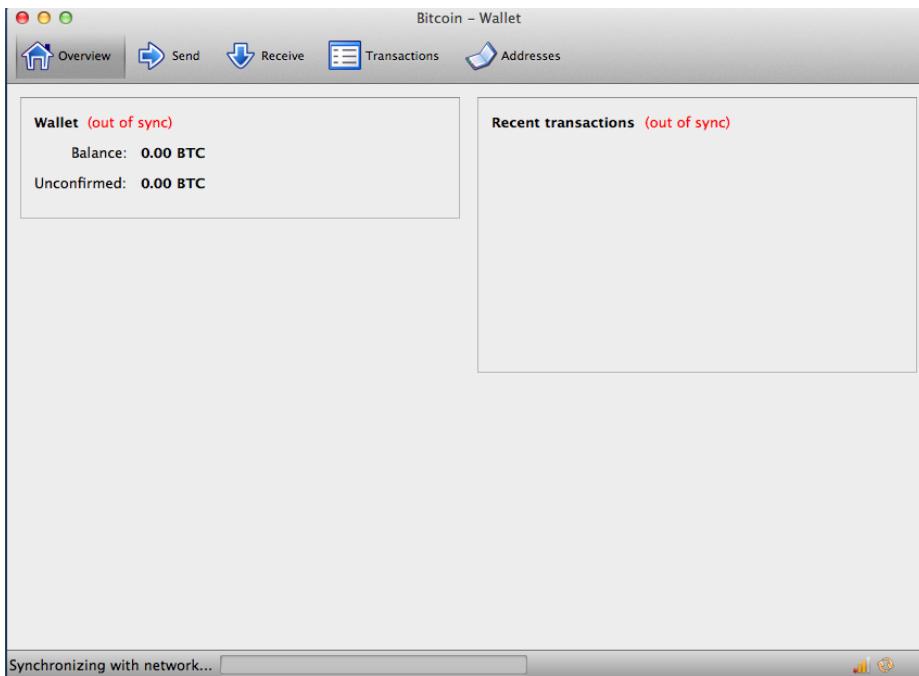


3-1. ábra: A bitcoin kliens kiválasztása a bitcoin.org webhelyen

A Bitcoin Core kliens első futtatása

Ha egy végrehajtható csomagot, például egy .exe, .dmg, vagy PPA csomagot töltöttünk le, akkor ugyanúgy installálhatjuk az operációs rendszerünkön, mint bármely más alkalmazást. A Windows esetében futtassuk az .exe állományt, és lépésről lépésre kövessük az utasításokat. A Mac OS esetében indítsuk el a .dmg-t, és húzzuk a Bitcoin-Qt ikont az *Alkalmazások* mappába. Az Ubuntu esetében a File Explorer-ben kattintsunk duplán a PPA-ra, ennek hatására megnyílik a package manager, amellyel installálható a csomag. Az installálás befejeződése után egy új „Bitcoin-Qt” alkalmazás jelenik meg az alkalmazások között. Az ikonon történő dupla kattintással indítsuk el a bitcoin klienst.

A Bitcoin Core első futtatásakor megkezdődik a blokklánc letöltése. A letöltési folyamat több napig tarthat (lásd 3-2. ábra). Hagyjuk, hadd fusson a háttérben, amíg meg nem jelenik rajta az egyenleg mellett, hogy „Szinkronban” („Synchronized”), és már nem azt jelzi ki, hogy „Nincs szinkronban” („Out of sync”).



3-2. ábra: A Bitcoin Core képernyője a blokklánc inicializálása során

A Bitcoin Core a tranzakciós nyilvántartás (blokklánc) egy teljes másolatát állítja elő, melyben a bitcoin hálózat elindulása, vagyis a 2009 óta lezajlott összes tranzakció megtalálható. Ez egy jó pár gigabájt méretű adathalmaz (2013 végén kb. 16 GB volt a mérete), és pár nap alatt, fokozatosan töltődik le. A kliens csak a blokklánc összes adatának letöltése után képes a tranzakciók feldolgozására vagy a számla egyenlegek módosítására. A letöltés során a kliens az egyenleg mellett az jelzi ki, hogy „Nincs szinkronban” („Out of sync”), az ablak alján pedig azt, hogy „Szinkronizálás a hálózattal” („Synchronizing”). Ellenőrizze, hogy van-e elég szabad hely a diszken, valamint elegendő sávszélesség és idő áll-e rendelkezésre a kezdeti szinkronizáláshoz.

A Bitcoin Core kliens lefordítása a forráskódóból

A fejlesztők azt is megtehetik, hogy ZIP archívumként letölthetik a teljes forrást, vagy a Githubról klónozzák a hiteles forráshalmazt. Menjen a [GitHub bitcoin web lapra](#), és válassza a „Download ZIP” gombot a jobb oldalról. Egy másik lehetőség az, ha a git parancssal létrehozza a forráskód egy helyi másolatát a rendszerén. A lenti példában egy Linux vagy Mac OS alatt kiadott Unix-szerű parancssal fogjuk klónozni a forrás kódot:

```
$ git clone https://github.com/bitcoin/bitcoin.git
Cloning into 'bitcoin'...
remote: Counting objects: 31864, done.
remote: Compressing objects: 100% (12007/12007), done.
remote: Total 31864 (delta 24480), reused 26530 (delta 19621)
Receiving objects: 100% (31864/31864), 18.47 MiB | 119 KiB/s, done.
Resolving deltas: 100% (24480/24480), done.
$
```



Az utasítások és azok kimenete verzióról verzióra változhat. Kövesse a forráskód melletti dokumentációt, még akkor is, ha az különbözik attól, amit itt lát, és ne lepődjön meg akkor sem, ha a képernyőjén megjelenő kimenet kicsit különbözik attól, mint amit az itteni példák tartalmaznak.

A klónozási művelet befejeződése után a forráskódról egy teljes másolat lesz a helyi *bitcoin* könyvtárban. Menjen ebbe a könyvtárba. Gépelje be a `cd bitcoin` parancsot:

```
$ cd bitcoin
```

Ha a `git clone` parancsban semmi sem volt megadva, akkor a helyi példány a legfrissebb kóddal lesz szinkronban, ami akár a bitcoin kliens egy nem stabil vagy „béta” verziója is lehet. A kód lefordítása előtt egy *release tag* (*címke*) megadásával egy adott verzió választható ki. A tag (*címke*) kulcsszó a helyi másolatot a kódtár egy adott pillanatképével szinkronizálja. A fejlesztők a címkék használatával tudják egy verziószámmal megjelölni a kód egy adott verzióját. Azt, hogy milyen címkék vannak, a `git tag` parancssal írathatjuk ki:

```
$ git tag
v0.1.5
v0.1.6test1
v0.2.0
v0.2.10
v0.2.11
v0.2.12

[... sok egyéb címke ...]

v0.8.4rc2
v0.8.5
v0.8.6
v0.8.6rc1
v0.9.0rc1
```

A címkék listája a bitcoin összes kibocsátott változatát tartalmazza. Megállapodás szerint azoknak a jelölteknek, melyek tesztelésre szolgálnak, „rc” az utótagja (ami a *release candidate* kezdőbetűinek felel meg). A stabil változatoknak, melyek éles rendszereken futtathatók, nincs utótagjuk. A fenti listából a legmagasabb verziójú változatot választjuk, ami az adott időpontban a v0.9.0rc1. Ahhoz, hogy a helyi kód ezzel a változattal legyen szinkronban, a `git checkout` parancsot használjuk:

```
$ git checkout v0.9.0rc1
Note: checking out 'v0.9.0rc1'.

HEAD is now at 15ec451... Merge pull request #3605
$
```

A forráskód tartalmaz némi dokumentációt is, melyek különféle állományokban találhatók. Nézze át a bitcoin könyvtár *README.md* állományában található dokumentációt. Ehhez gépelje be a prompt-on: `more README.md`, és lapozásra használja a szóköz billentyűt. Ebben a fejezetben egy parancssori bitcoin klienst fogunk újraépíteni, melyet a Linux-on `bitcoind`-nek hívnak. Nézze át, hogy hogyan kell az ön platformján a `bitcoind` parancssori klienst lefordítani, ehhez gépelje be: `more doc/build-unix.md`. A Mac OS X és a Windows

esetében az utasítások a *doc* könyvtárban, a *build-osx.md* vagy a *build-msw.md* állományokban vannak.

Gondosan nézze át az újraépítés előfeltételeit, melyek a dokumentáció első részében találhatók. Az itt felsorolt könyvtárknak a rendszerben léteznie kell, mielőtt megkezdené a fordítást. Ha az előfeltételek nem teljesülnek, akkor az újraépítési folyamat hibával ér véget. Ha azért képződött hiba, mert valamelyik előfeltétel nem teljesült, akkor a megfelelő könyvtár installálása után a build folyamat onnan folytatatható, ahol abbamarradt. Ha az előfeltételek teljesülnek, akkor az újraépítési folyamat úgy kezdhető el, hogy az *autogen.sh* scripttel létrehozza az újraépítésre szolgáló scripteket.



A bitcoind build folyamata a 0.9 változat óta az *autogen/configure/make* rendszert használja. A régebbi változatok egy egyszerű *Makefile*-t használnak, és a lenti példától kissé különböző módon működnek. Kövesse annak a változatnak az utasításait, amelyet szeretne lefordítani. Valószínűleg a 0.9-ben bevezetett *autogen/configure/make* lesz az összes jövőbeli kódváltoztnál használt rendszer, és a lenti példa ezt a rendszert mutatja be.

```
$ ./autogen.sh
configure.ac:12: installing `src/build-aux/config.guess'
configure.ac:12: installing `src/build-aux/config.sub'
configure.ac:37: installing `src/build-aux/install-sh'
configure.ac:37: installing `src/build-aux/missing'
src/Makefile.am: installing `src/build-aux/depcomp'
$
```

Az *autogen.sh* script automatikus konfiguráló scripteket hoz létre, melyek a rendszerből lekérdezik a helyes beállításokat, és biztosítják, hogy a fordításhoz szükséges összes könyvtár rendelkezésre álljon. Ezek közül a legfontosabb a *configure* script, amely számos különféle lehetőséget ajánl a build folyamat egyedivé tételere. A különféle lehetőségek megjelenítésére gépelje be: *./configure --help*

```
$ ./configure --help
`configure' configures Bitcoin Core 0.9.0 to adapt to many kinds of
systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE. See below for descriptions of some of the useful variables.
```

Defaults for the options are specified in brackets.

Configuration:

```
-h, --help display this help and exit
--help=short display options specific to this package
--help=recursive display the short help of all the included packages
-v, --version display version information and exit

[... sok egyéb opció és változó kilistázása ...]

Optional Features:
--disable-option-checking ignore unrecognized --enable/--with options
--disable-FEATURE do not include FEATURE (same as --enable-FEATURE=no)
--enable-FEATURE [=ARG] include FEATURE [ARG=yes]
```

[... további opciók ...]

Use these variables to override the choices made by `configure' or to help it to find libraries and programs with nonstandard names/locations.

Report bugs to <info@bitcoin.org>.

\$

A `configure` scripttel bizonyos jellemzők engedélyezése vagy tiltása lehetséges, az `--enable-FEATURE` és `--disable-FEATURE` használatával, ahol a `FEATURE` a fenti listában szereplő jellemző neve. Ebben a fejezetben egy olyan bitcoind klienst építünk, amelynek alapértelmezett jellemzői lesznek. Nem használunk egyetlen egy konfigurálási lehetőséget sem, de érdemes átnézni, hogy a kliensnek milyen egyéb opcionális részei lehetnek. Ezután a `configure` script futtatásával automatikusan feltérképezzük, hogy melyek a szükséges könyvtárak, és egy testre szabott build scriptet hozunk létre a rendszerünk számára:

```
$ ./configure
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
```

[... sok egyéb vizsgált rendszerjellemző ...]

```
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating src/test/Makefile
config.status: creating src/qt/Makefile
config.status: creating src/qt/test/Makefile
config.status: creating share/setup.nsi
config.status: creating share/qt/Info.plist
config.status: creating qa/pull-tester/run-bitcoind-for-test.sh
config.status: creating qa/pull-tester/build-tests.sh
config.status: creating src/bitcoin-config.h
config.status: executing depfiles commands
$
```

Ha minden jól megy, akkor a `configure` parancs úgy ér véget, hogy egy testre szabott build scriptet hoz létre, amellyel lefordítható a `bitcoind`. Ha hiányzó könyvtárak vagy hibák vannak, akkor a `configure` parancs hibával fog véget érni, és nem hozza létre a fenti példában látható build scripteket. Ha hiba történik, annak a legvalószínűbb oka egy hiányzó vagy nem kompatibilis könyvtár. Nézze át ismét az újraépítésre vonatkozó dokumentációt, és installálja a hiányzó előfeltételeket. Azután futtassa le ismét a `configure -t`, és nézze meg, hogy elmúlt-e a hiba. Ezután fordítsa le a forráskódot – ez a folyamat akár egy óráig is tarthat. A fordítás során néhány másodpercenként vagy néhány percenként megjelenik valami – vagy hibaüzenetet kap, ha valami baj van. A fordítási folyamat bármikor folytatható, ha félbeszakadt. A fordítás megkezdéséhez gépelje be, hogy `make`:

```

$ make
Making all in src
make[1]: Entering directory `/home/ubuntu/bitcoin/src'
make all-recursive
make[2]: Entering directory `/home/ubuntu/bitcoin/src'
Making all in .
make[3]: Entering directory `/home/ubuntu/bitcoin/src'
  CXX addrman.o
  CXX alert.o
  CXX rpcserver.o
  CXX bloom.o
  CXX chainparams.o

[... sok egyéb fordítási üzenet ...]

  CXX test_bitcoin-wallet_tests.o
  CXX test_bitcoin-rpc_wallet_tests.o
  CXXLD test_bitcoin
make[4]: Leaving directory `/home/ubuntu/bitcoin/src/test'
make[3]: Leaving directory `/home/ubuntu/bitcoin/src/test'
make[2]: Leaving directory `/home/ubuntu/bitcoin/src'
make[1]: Leaving directory `/home/ubuntu/bitcoin/src'
make[1]: Entering directory `/home/ubuntu/bitcoin'
make[1]: Nothing to be done for `all-am'.
make[1]: Leaving directory `/home/ubuntu/bitcoin'
$
```

A fordító megjegyzése: Ubuntu 12.04 LTS alatt a bitcoind fordításához szükséges parancsok:

```

$ sudo apt-get install autoconf
$ ./autogen.sh
$ sudo apt-get install git-core build-essential libssl-dev libboost-all-dev
libdb-dev libdb++-dev libgtk2.0-dev
$ ./configure --with-incompatible-bdb
$ make
```

Ha minden jól megy, akkor a bitcoind lefordul. Az utolsó lépés az, hogy a bitcoind végrehajtható programot a `make` parancssal a rendszer path-ba installáljuk:

```

$ sudo make install
Making install in src
Making install in .
 /bin/mkdir -p '/usr/local/bin'
 /usr/bin/install -c bitcoind bitcoin-cli '/usr/local/bin'
Making install in test
make install-am
 /bin/mkdir -p '/usr/local/bin'
 /usr/bin/install -c test_bitcoin '/usr/local/bin'
$
```

A következőképpen bizonyosodhatunk meg arról, hogy a bitcoind helyesen van installálva:

```

$ which bitcoind
/usr/local/bin/bitcoind

$ which bitcoin-cli
/usr/local/bin/bitcoin-cli
```

Az alapértelmezés szerinti installáláskor a bitcoind a `/usr/local/bin` könyvtárba kerül. Amikor a bitcoind-t először futtatjuk, akkor üzen, hogy egy konfigurációs állományt kell létrehoznunk, melyben a JSON-RPC interface számára egy erős jelszó van megadva. Futtassunk a bitcoind-t a `bitcoind` terminálon történő begépelésével:

```
$ bitcoind
Error: To use the "-server" option, you must set a rpcpassword in the
configuration file:
/home/ubuntu/.bitcoin/bitcoin.conf
It is recommended you use the following random password:
rpcuser=bitcoindrpc
rpcpassword=2XA4DuKNCbtZXsBQRRNDEwEY2nM6M4H9Tx5dFjoAVVbK
(you do not need to remember this password)
The username and password MUST NOT be the same.
If the file does not exist, create it with owner-readable-only file
permissions.
It is also recommended to set alertnotify so you are notified of problems;
for example: alertnotify=echo %s | mail -s "Bitcoin Alert" admin@foo.com
```

Egy általunk ismert szerkesztő programmal szerkeszük meg a konfigurációs állományt, jelszónak pedig adjunk meg egy erős jelszót, ahogyan azt a bitcoind javasolta. *Ne használjuk a fenti jelszót.* A `.bitcoin` könyvtáron belül hozzunk létre egy `bitcoin.conf` állományt, és adjuk meg benne a felhasználói nevet és jelszót:

```
rpcuser=bitcoindrpc
rpcpassword=2XA4DuKNCbtZXsBQRRNDEwEY2nM6M4H9Tx5dFjoAVVbK
```

A konfigurációs állomány szerkesztése során egyéb paraméterek is beállíthatók, pl. a `txindex` (lásd a „[Tranzakciós Adatbázis Index és a txindex opció](#)” részt). A használható opciók a `bitcoind --help` begépelésével listázhatók ki.

Indítsuk el a bitcoin klienst. Az első futtatáskor a kliens a blokkok letöltésével újra fogja építeni a teljes bitcoin blokkláncot. Ennek sok gigabájt a mérete, és a teljes letöltése átlagosan 2 napig tart. A blokklánc inicializálási idő lerövidíthető, ha a blokklánc egy részleges másolatát bittorrent segítségével a [SourceForge](#) helyről letöljük.

A `-daemon` opció megadásával futtassuk a bitcoind-t a háttérben:

```
$ bitcoind -daemon

Bitcoin version v0.9.0rc1-beta (2014-01-31 09:30:15 +0100)
Using OpenSSL version OpenSSL 1.0.1c 10 May 2012
Default data directory /home/bitcoin/.bitcoin
Using data directory /bitcoin/
Using at most 4 connections (1024 file descriptors available)
init message: Verifying wallet...
dbenv.open LogDir=/bitcoin/database ErrorFile=/bitcoin/db.log
Bound to [::]:8333
Bound to 0.0.0.0:8333
init message: Loading block index...
Opening LevelDB in /bitcoin/blocks/index
Opened LevelDB successfully
Opening LevelDB in /bitcoin/chainstate
Opened LevelDB successfully
[... további indulási üzenetek ...]
```

A Bitcoin Core JSON-RPC API-jának a használata a parancssorból

A Bitcoin Core kliensben van egy parancssorból elérhető JSON-RPC interfész, amely a `bitcoin-cli` segédprogrammal érhető el. A parancssor lehetővé teszi, hogy interaktívan kísérletezzünk azokkal a lehetőségekkel, melyek az API-n keresztül programokból is elérhetők. Indulásként, a `help` parancs kiadásával jelenítsük meg a használható bitcoin RPC parancsok listáját:

```
$ bitcoin-cli help
addmultisigaddress nrequired ["key",...] ( "account" )
addnode "node" "add|remove|onetry"
backupwallet "destination"
createmultisig nrequired ["key",...]
createrawtransaction [{"txid":"id","vout":n},...] {"address":amount,...}
decoderawtransaction "hexstring"
decodescript "hex"
dumpprivkey "bitcoinaddress"
dumpwallet "filename"
getaccount "bitcoinaddress"
getaccountaddress "account"
getaddednodeinfo dns ( "node" )
getaddressesbyaccount "account"
getbalance ( "account" minconf )
getbestblockhash
getblock "hash" ( verbose )
getblockchaininfo
getblockcount
getblockhash index
getblocktemplate ( "jsonrequestobject" )
getconnectioncount
getdifficulty
getgenerate
gethashespersec
getinfo
getmininginfo
getnettotals
getnetworkhashps ( blocks height )
getnetworkinfo
getnewaddress ( "account" )
getpeerinfo
getrawchangeaddress
getrawmempool ( verbose )
getrawtransaction "txid" ( verbose )
getreceivedbyaccount "account" ( minconf )
getreceivedbyaddress "bitcoinaddress" ( minconf )
gettransaction "txid"
gettxout "txid" n ( includemempool )
gettxoutsetinfo
getunconfirmedbalance
getwalletinfo
getwork ( "data" )
help ( "command" )
importprivkey "bitcoinprivkey" ( "label" rescan )
importwallet "filename"
keypoolrefill ( newsize )
listaccounts ( minconf )
listaddressgroupings
```

```

listlockunspent
listreceivedbyaccount ( minconf includeempty )
listreceivedbyaddress ( minconf includeempty )
listsinceblock ( "blockhash" target-confirmations )
listtransactions ( "account" count from )
listunspent ( minconf maxconf ["address",...] )
lockunspent unlock [{"txid":"txid","vout":n},...]
move "fromaccount" "toaccount" amount ( minconf "comment" )
ping
sendfrom "fromaccount" "tobitcoinaddress" amount ( minconf "comment"
"comment-to" )
sendmany "fromaccount" {"address":amount,...} ( minconf "comment" )
sendrawtransaction "hexstring" ( allowhighfees )
sendtoaddress "bitcoinaddress" amount ( "comment" "comment-to" )
setaccount "bitcoinaddress" "account"
setgenerate generate ( genproclimit )
settxfee amount
signmessage "bitcoinaddress" "message"
signrawtransaction "hexstring" (
[{"txid":"id","vout":n,"scriptPubKey":"hex","redeemScript":"hex"},...]
["privatekey1",...] sighashtype )
stop
submitblock "hexdata" ( "jsonparametersobject" )
validateaddress "bitcoinaddress"
verifychain ( checklevel numblocks )
verifymessage "bitcoinaddress" "signature" "message"
walletlock
wallet passphrase "passphrase" timeout
walletpassphrasetchange "oldpassphrase" "newpassphrase"

```

A Bitcoin Core kliens státuszának lekérdezése

Parancs: `getinfo`

A bitcoin `getinfo` RPC parancsával a bitcoin hálózati csomópontról, a pénztárcáról és a blokklánc adatbázisról írathatók ki a legfontosabb adatok. A parancs a `bitcoin-cli` használatával futtatható:

```

$ bitcoin-cli getinfo
{
    "version" : 90000,
    "protocolversion" : 70002,
    "walletversion" : 60000,
    "balance" : 0.00000000,
    "blocks" : 286216,
    "timeoffset" : -72,
    "connections" : 4,
    "proxy" : "",
    "difficulty" : 2621404453.06461525,
    "testnet" : false,
    "keypoololdest" : 1374553827,
    "keypoolsize" : 101,
    "paytxfee" : 0.00000000,
    "errors" : ""
}

```

Az adatokat JavaScript Object Notation (JSON) formátumban kapjuk vissza. Ezt a formátumot az összes programozási nyelv könnyen „megérti”, ugyanakkor emberek számra is egész olvasható. Az adatok között látható a bitcoin kliens szoftver verziója (90000), a

protokoll verziója (70002), és a pénztárca verziója (60000). Látható a pénztárcában lévő aktuális egyenleg, amely nulla. Látható a blokk magasság (286216), amely azt mutatja, hogy a hány blokkot ismer a kliens. Láthatunk még a bitcoin hálózatra és a kliens beállítására vonatkozó különféle statisztikákat. A beállításokat a fejezet hátrólévő részletesebben megvizsgáljuk.



Eltarthat némi ideig, akár több napig is, amíg a bitcoind kliens „beéri” az aktuális blokklánc magasságot, eközben blokkokat tölt le a többi bitcoin klienstől. A folyamat menete úgy ellenőrizhető, hogy a `getinfo`-val kiíratjuk az ismert blokkok számát.

A pénztárca beállítása és titkosítása

Parancsok: `encryptwallet`, `walletpassphrase`

Mielőtt a kulcsok létrehozásával és más parancsokkal folytatnánk a dolgot, először egy jelszóval titkosítanunk kell a pénztárcát. Ennél a példánál az `encryptwallet` parancsot és a „foo” jelszót használjuk. Természetesen a „foo” jelszavat az önök pénztárcája esetében egy erős és összetett jelszóval kell helyettesíteni!

```
$ bitcoin-cli encryptwallet foo
wallet encrypted; Bitcoin server stopping, restart to run with encrypted
wallet. The keypool has been flushed, you need to make a new backup.
$
```

Úgy tudjuk ellenőrizni, hogy titkosított lett-e a pénztárca, hogy ismét lefuttatjuk a `getinfo` parancsot. Ezúttal egy új sort is láthatunk, az `unlocked_until` sort, amely azt mutatja, hogy mennyi ideig lesz a pénztárca kikódoló jelszó a memóriában, és mennyi ideig fogja a pénztárcát nyitva tartani. Először nulla lesz az értéke, ami azt jelenti, hogy a pénztárca zárolva van:

```
$ bitcoin-cli getinfo
{
    "version" : 90000,
    # [... további információk...]
    "unlocked_until" : 0,
    "errors" : ""
}
$
```

A pénztárca zárolásához feloldásához adjuk ki a `walletpassphrase` parancsot, melynek két paramétere van: a jelszó, és egy szám, amely azt mutatja, hogy hány másodperc múlva záródik be automatikusan a pénztárca (idő számláló):

```
$ bitcoin-cli walletpassphrase foo 360
$
```

A `getinfo` ismételt futtatásával nézzük meg, hogy kinyílt-e a pénztárca, és mennyi a lejárat idő:

```
$ bitcoin-cli getinfo
{
    "version" : 90000,
    # [... egyéb információk ...]
    "unlocked_until" : 1392580909,
    "errors" : ""
}
```

A pénztárca biztonsági mentése, egyszerű szövegként történő kivitele és visszaállítása

Parancsok: backupwallet, importwallet, dumpwallet

Ezután azt fogjuk gyakorolni, hogy hogyan lehet a pénztárcáról biztonsági mentést készíteni, és hogyan lehet a biztonsági mentésből visszaállítani a pénztárcát. A mentéshez a `backupwallet` parancs használható, melynek paramétere egy állománynév. Például mentsük el a pénztárcát a `wallet.backup` állományba:

```
$ bitcoin-cli backupwallet wallet.backup
$
```

Most állítsuk helyre a pénztárcát az `importwallet` parancssal. Ha a pénztárca zárolva van, akkor a biztonsági mentés visszaimportálása előtt először meg kell szüntetni a zárolását (lásd a fenti `walletpassphrase` parancsot).

```
$ bitcoin-cli importwallet wallet.backup
$
```

A `dumpwallet` parancssal a pénztárca egy olvasható szöveges állományba vihető ki:

```
$ bitcoin-cli dumpwallet wallet.txt
$ more wallet.txt
# Wallet dump created by Bitcoin v0.9.0rc1-beta (2014-01-31 09:30:15 +0100)
# * Created on 2014-02- 8dT20:34:55Z
# * Best block at time of backup was 286234
(0000000000000000f74f0bc9d3c186267bc45c7b91c49a0386538ac24c0d3a44),
#     mined on 2014-02- 8dT20:24:01Z

KzTg2wn6Z8s7ai5NA9MVX4vstHRSqP26QKJCzLg4JvFrp6mMaGB9 2013-07- 4dT04:30:27Z
change=1 # addr=16pJ6XkwSQv5ma5FSXMRPaXEYrENCEg47F
Kz3dVz7R6mUpXzdZy4gJEVzxXJwA15f198eVui4CUivXotzLBDKY 2013-07- 4dT04:30:27Z
change=1 # addr=17oJds8kaN8LP8kuAkWTco6ZM7BGXFC3gk
[... sok egyéb kulcs ...]

$
```

Pénztárca címek és pénzt fogadó tranzakciók

Parancsok: getnewaddress, getreceivedbyaddress, listtransactions, getaddressesbyaccount, getbalance

A bitcoin referencia kliens nyilvántart egy címhalmazt, melynek mérete a `getinfo` parancs használatakor a `keypoolsize` -nál jelenik meg. Ezek a címek automatikusan jönnek létre, és

nyilvános pénz fogadó címként vagy a visszajáró pénz címeként használhatók. Egy ilyen cím a `getnewaddress` parancssal íratható ki:

```
$ bitcoin-cli getnewaddress  
1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL
```

Ennek a címnek a használatával küldjünk egy kis bitcoint egy külső pénztárcából a bitcoind pénztárcánkba (ha már van némi bitcoinunk egy pénzváltóban, egy web-es pénztárcában vagy egy másik bitcoind pénztárcában). Példánkban 50 mBTC-t (0.050 bitcoint) fogunk küldeni a fent visszaadott címre.

Ezután lekérdezhetjük a bitcoind klienstől, hogy mennyi pénz jött erre a címre, és megadhatjuk, hogy hány darab megerősítésre van szükség ahhoz, hogy a pénzt beszámítsa az egyenlegbe. Példánkban nulla megerősítést fogunk megadni. Néhány másodperccel azt követően, hogy a másik pénztárcából elküldtük a bitcoint, már látjuk, hogy megjelent a pénztárcában. A `getreceivedbyaddress` parancsot fogjuk használni a fenti címmel, és nulla (0) megerősítési számmal:

```
$ bitcoin-cli getreceivedbyaddress 1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL 0  
0.05000000
```

Ha a nullát elhagyjuk a parancs végéről, akkor csak azokat az összegeket fogjuk látni, melyeknek legalább `minconf` megerősítése van. A `minconf` beállítás a bitcoind konfigurációs állományában található. Mivel a bitcoin küldő tranzakció csak az utolsó pár másodpercen érkezett meg, még nincs megerősítve, és emiatt a parancs nulla egyenleget fog kiírni:

```
$ bitcoin-cli getreceivedbyaddress 1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL  
0.00000000
```

A pénztárca által fogadott összes tranzakció is kijelzhető, a `listtransactions` parancssal:

```
$ bitcoin-cli listtransactions  
[  
  {  
    "account" : "",  
    "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",  
    "category" : "receive",  
    "amount" : 0.05000000,  
    "confirmations" : 0,  
    "txid" :  
"9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",  
    "time" : 1392660908,  
    "timereceived" : 1392660908  
  }  
]
```

A `getaddressesbyaccount` parancssal kilistázhatjuk a pénztárcához tartozó összes címét is:

```
$ bitcoin-cli getaddressesbyaccount ""  
[  
  "1LQoTPYy1TyERbNV4zZbhEmgyfAipC6eqL",  
  "17vrg8uwMQUibkvS2ECRX4zpcVJ78iFaZS",  
  "1FvRHWhHBBZA8cGRRsGiAeqEzUmjJkJQWR",  
  "1NVJK3JsL41BF1KyxrUyJW5XHjunjfP2jz",  
]
```

```

"14MZqqzCxjc99M5ipsQSRfiT7qPZcM7Df",
"1BhrGvtKFjTAhGdPGbrEwP3xvFjkJBuFCa",
"15nem8CX91XtQE8B1Hdv97jE8X44H3DQMT",
"1Q3q6taTsUiv3mMemEuQQJ9sGLEGaSjo81",
"1HoSiTg8sb16oE6SrmaZQEWcGEv8obv9ns",
"13fE8BGhBvnOy68yZKuWJ2hheYKovSDjqM",
"1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
"1KHUmVfCJteJ21LmRXHSpPoe23rXKifAb2",
"1LqJZz1D9yHxG4cLkdujnqG5jNNGmPeAMD"
]

```

Végül a `getbalance` parancssal a pénztárca egészének egyenlegét írathatjuk ki, vagyis az összes olyan tranzakció összegét, amely legalább `minconf` megerősítéssel rendelkezik:

```
$ bitcoin-cli getbalance
0.05000000
```



Ha a tranzakció még nincs megerősítve, akkor a `getbalance` által visszaadott egyenleg nulla lesz. A „`minconf`” konfigurációs beállítás határozza meg, hogy legalább hány darab megerősítésre van szükség ahhoz, hogy a tranzakció megjelenjen az egyenlegben.

Tranzakciók vizsgálata és dekódolása

Parancsok: `gettransaction`, `getrawtransaction`, `decoderawtransaction`

Most a `gettransactions` parancssal megvizsgáljuk azt a bejövő tranzakciót, amelyet előzőleg klistáltunk. A tranzakciót úgy tudjuk klistázni, hogy a `gettransaction` parancsban a `txid` helyén megadjuk a tranzakció hash értékét (zanzáját):

```
$ bitcoin-cli gettransaction
9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3
{
    "amount" : 0.05000000,
    "confirmations" : 0,
    "txid" :
"9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
    "time" : 1392660908,
    "timereceived" : 1392660908,
    "details" : [
        {
            "account" : "",
            "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
            "category" : "receive",
            "amount" : 0.05000000
        }
    ]
}
```



A tranzakció azonosítók mindaddig nem hitelesek, amíg a tranzakció megerősítésre nem került. Ha a blokkláncon hiányzik a tranzakció azonosító, az nem jelenti azt, hogy a tranzakció nem lett feldolgozva. Ez az ún. „tranzakció képlékenység” („transaction malleability”), amelynek az az oka, hogy egy tranzakció hashe az előtt, mielőtt megerősítésre kerül a blokkban, még módosítható. A megerősítés után a `txid` megváltoztathatatlan és hiteles.

A `gettransaction` által fent kijelzett tranzakció egy egyszerűsített alak. A teljes tranzakció visszányeréséhez és dekódolásához két parancsot fogunk használni, a `getrawtransaction` és a `decoderawtransaction` parancsot. Először, a `getrawtransaction` parancsnak paraméterként megadjuk a *tranzakció hash-t* (*txid-t*), a parancs pedig „nyers” hexa stringként a teljes tranzakciót visszaadja, pontosan úgy, ahogyan azt a bitcoin hálózat ismeri :

```
$ bitcoin-cli getrawtransaction
9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3
```

```
0100000001d717279515f88e2f56ce4e8a31e2ae3e9f00ba1d0add648e80c480ea22e0c7d30
00000008b483045022100a4ebbeec83225dedead659bbde7da3d026c8b8e12e61a2df0dd075
8e227383b302203301768ef878007e9ef7c304f70ffaf1f2c975b192d34c5b9b2ac1bd193df
ba2014104793ac8a58ea751f9710e39aad2e296cc14daa44fa59248be58ede65e4c4b884ac5
b5b6dede05ba84727e34c8fd3ee1d6929d7a44b6e111d41cc79e05dbfe5ceafffffff02404
b4c000000000001976a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac1f31290600
0000001976a914107b7086b31518935c8d28703d66d09b3623134388ac000000000
```

Ennek a hexa stringnek a dekódolására a `decoderawtransaction` parancs használható. A teljes tartalom JSON adatstruktúraként történő megjeleníteni másoljuk a hexa stringet a `decoderawtransaction` első paraméterének a helyébe:

```
$ bitcoin-cli decoderawtransaction
0100000001d717279515f88e2f56ce4e8a31e2ae3e9f00ba1d0add648e80c480ea22e0c7d30
00000008b483045022100a4ebbeec83225dedead659bbde7da3d026c8b8e12e61a2df0dd075
8e227383b302203301768ef878007e9ef7c304f70ffaf1f2c975b192d34c5b9b2ac1bd193df
ba2014104793ac8a58ea751f9710e39aad2e296cc14daa44fa59248be58ede65e4c4b884ac5
b5b6dede05ba84727e34c8fd3ee1d6929d7a44b6e111d41cc79e05dbfe5ceafffffff02404
b4c000000000001976a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac1f31290600
0000001976a914107b7086b31518935c8d28703d66d09b3623134388ac000000000
{
    "txid" :
"9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "txid" :
"d3c7e022ea80c4808e64dd0a1dba009f3eaee2318a4ece562f8ef815952717d7",
            "vout" : 0,
            "scriptSig" : {
                "asm" :
"3045022100a4ebbeec83225dedead659bbde7da3d026c8b8e12e61a2df0dd0758e227383b3
02203301768ef878007e9ef7c304f70ffaf1f2c975b192d34c5b9b2ac1bd193dfba20104793
ac8a58ea751f9710e39aad2e296cc14daa44fa59248be58ede65e4c4b884ac5b5b6dede05ba
84727e34c8fd3ee1d6929d7a44b6e111d41cc79e05dbfe5cea",
                "hex" :
"483045022100a4ebbeec83225dedead659bbde7da3d026c8b8e12e61a2df0dd0758e227383
b302203301768ef878007e9ef7c304f70ffaf1f2c975b192d34c5b9b2ac1bd193dfba201410
4793ac8a58ea751f9710e39aad2e296cc14daa44fa59248be58ede65e4c4b884ac5b5b6dede
05ba84727e34c8fd3ee1d6929d7a44b6e111d41cc79e05dbfe5cea"
            },
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 0.05000000,
            "n" : 0,
            "scriptPubKey" :

```

```

        "asm" : "OP_DUP OP_HASH160
07bdb518fa2e6089fd810235cf1100c9c13d1fd2 OP_EQUALVERIFY OP_CHECKSIG",
        "hex" :
"76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
        "reqSigs" : 1,
        "type" : "pubkeyhash",
        "addresses" : [
            "1hvzSofGwt8cjB8JU7nBsCSfEVQX5u9CL"
        ]
    }
},
{
    "value" : 1.03362847,
    "n" : 1,
    "scriptPubKey" : {
        "asm" : "OP_DUP OP_HASH160
107b7086b31518935c8d28703d66d09b36231343 OP_EQUALVERIFY OP_CHECKSIG",
        "hex" :
"76a914107b7086b31518935c8d28703d66d09b3623134388ac",
        "reqSigs" : 1,
        "type" : "pubkeyhash",
        "addresses" : [
            "12W9goQ3P7Waw5JH8fRVs1e2rVAKoGnvoy"
        ]
    }
}
]
}

```

A tranzakció dekódolás a tranzakció összes részét, többek között a tranzakció bemeneteket és kimeneteket is megjeleníti. Ebben az esetben azt látjuk, hogy a tranzakció, amely 50 mBTC-t írt jóvá az új címünkre, egy bemenetet és két kimenetet használt. A tranzakció bemenete egy előzőleg megerősített tranzakció kimenete volt (amely fent a `d3c7` kezdetű vin txid-ként szerepel). A két kimenet megfelel az 50 mBTC jóváírásnak és a visszajáró pénznek.

A blokkláncot tovább tudjuk vizsgálni, ha ugyanezzel a parancssal (vagyis a `gettransaction` parancssal) a tranzakcióban szereplő előző tranzakciókat listázzuk ki. Tranzakcióról tranzakcióra lépve nyomon követhetjük a tranzakciótánban, hogy az egyes tulajdonosok között hogyan mozogtak az érmék.

Ha az általunk fogadott tranzakció már be lett foglalva egy blokkba és megerősítésre került, akkor a `gettransaction` parancs további információkat is szolgáltat, többek között megmutatja azt a *blockhash* (*azonosítót*), amelybe bele lett foglalva a tranzakció:

```

$ bitcoin-cli gettransaction
9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3
{
    "amount" : 0.05000000,
    "confirmations" : 1,
    "blockhash" :
"00000000000000051d2e759c63a26e247f185ecb7926ed7a6624bc31c2a717b",
    "blockindex" : 18,
    "blocktime" : 1392660808,
    "txid" :
"9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
    "time" : 1392660908,
    "timereceived" : 1392660908,
    "details" : [

```

```
        {
            "account" : "",
            "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
            "category" : "receive",
            "amount" : 0.05000000
        }
    ]
}
```

Az egyik új információ a `blockhash`, ami annak a blocknak a hashe (zanzája), amibe a tranzakció be lett foglalva, a másik pedig a `blockindex`, melynek értéke (18) azt mutatja, hogy a tranzakcióból a blokk 18. tranzakciója.

A Tranzakciós Adatbázis Index és a txindex opció

Alapértelmezésben a Bitcoin Core által felépített adatbázis *kizárolag* a felhasználó pénztárcájához tartozó tranzakciókat tartalmazza. Ha a `gettransaction` parancccsal szeretnénk *akármelyik* tranzakciót kilistázni, akkor a Bitcoin Core-t úgy kell beállítanunk, hogy egy teljes adatbázis indexet építsen föl, amely a `txindex` opcionális paraméterrel lehetséges. Állítsa be a `txindex=1` opciót a Bitcoin Core konfigurációs állományában. (A konfigurációs állomány (általában) a felhasználó home könyvtárában van, a `.bitcoin/bitcoin.conf` állományban). A paraméter beállítását követően újra kell indítani a bitcoind-t és meg kell várni az index újraépülését.

Blokkok vizsgálata

Parancsok: getblock, getblockhash

Most, hogy tudjuk, melyik blokkba lett befoglalva a tranzakciónk, le tudjuk kérdezni ezt a blokkot. A `getblock` parancsot használjuk, és paraméterként a blokk hashét (zanzáját) adjuk meg:

A blokk 367 db tranzakciót tartalmaz, és a 18. kilistázott tranzakció (9ca8f9...) az a txid (tranzakció azonosító), amely 50 mBTC-t írt jóvá a címünkre. A height sor szerint az adott blokk a 286384. blokk volt a blokkláncban.

Egy blokkot a magassága alapján a `getblockhash` parancssal tudjuk elérni, a parancs paramétere a blokk magasság, és a blokkhoz tartozó blokk haszt (zanzát) adja vissza:

```
$ bitcoin-cli getblockhash 0  
0000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Fent a „genезис блок” блок хешт írtuk ki. Генезис блок volt az első, Satoshi Nakamoto által kibányászott blokk, és nulla a magassága. A blokk a következőket tartalmazza:

}

A `getblock`, `getblockhash` és `gettransaction` parancsokkal a blokklánc adatbázisát egy program segítségével is meg tudjuk vizsgálni.

Tranzakciók létrehozása, aláírása és feladása az el nem költött kimenetek alapján

Parancsok: listunspent, gettxout, createrawtransaction, decoderawtransaction, signrawtransaction, sendrawtransaction

A bitcoin tranzakciókban az előző tranzakciók „kimeneteinek” az elköltése történik. Ezáltal egy olyan tranzakciós lánc jön létre, amely a tulajdonjogot az egyik címről a másikra ruházza át. A pénztárcánk épp most fogadott egy olyan tranzakciót, amely egy ilyen kimenetet a mi címünkhez rendelt hozzá. A tranzakció megerősítése után el tudjuk költeni ezt a kimenetet.

Először is a `listunspent` parancssal a pénztárcánkban lévő összes elköltetlen, *megerősített* kimenetet íratjuk ki:

```
$ bitcoin-cli listunspent
[
    {
        "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
        "vout" : 0,
        "address" : "1hvzSofGwT8cjB8JU7nBsCSfEVQX5u9CL",
        "account" : "",
        "scriptPubKey" :
"76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
        "amount" : 0.05000000,
        "confirmations" : 7
    }
]
```

Láthatjuk, hogy a `9ca8f9...` tranzakció létrehozott egy kimenetet (0 vout index-szel), amely az `1hvzSo...` címhez van rendelve, a nagysága 50 mBTC, és már 7 megerősítést kaptunk róla. A tranzakciók az előzőleg létrehozott kimeneteket használják bemenetként oly módon, hogy a tranzakciókra az azonosítójukkal és vout indexükkel hivatkoznak. Most létre fogunk hozni egy tranzakciót, amely a bemenetén egy új címhez rendeli, és ezáltal elkölti a `9ca8f9...` tranzakció azonosító (txid) 0-ik kimenetét.

Először vizsgáljuk meg részletesebben az adott kimentet. A `gettxout` parancssal a fenti elköltetlen kimenet részleteit írathatjuk ki. A tranzakciós kimenetekre minden `txid` és `vout` alapján lehet hivatkozni, és ezeket a paramétereket adjuk meg a `gettxout`-nak:

```

        "hex" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
        "reqSigs" : 1,
        "type" : "pubkeyhash",
        "addresses" : [
            "1hVzSofGwT8cjb8JU7nBsCSfEVQX5u9CL"
        ]
    },
    "version" : 1,
    "coinbase" : false
}

```

Azt látjuk, hogy a kimenet 50 mBTC-t rendelt hozzá az `1hVz...` címünkhez. A kimenet elköltéséhez egy új tranzakciót hozunk létre. Először állítsunk elő egy címet, ahová elküldhetjük a pénzt:

```
$ bitcoin-cli getnewaddress
1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb
```

25 mBTC-t fogunk az újonnan létrehozott `1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb` címre elküldeni. Az új tranzakciónkban elköltjük az 50 mBTC-s kimenetet, és 25 mBTC-t küldünk erre az új címre. Mivel az előző tranzakció *egész* kimenetét el kell költenünk, a visszajáró pénzt is kezelnünk kell. A visszajáró pénzt az `1hVz...` címre fogjuk visszaküldeni, vagyis ugyanarra a címre, amelyről a pénz származott. Végül tranzakciós díjat is kell fizetnünk a tranzakcióért. A díjat úgy fogjuk megfizetni, hogy a visszajáró összeget 0.5 mBTC-vel csökkentjük, és csak 24.5 mBTC-t küldünk vissza. Az új kimenetek összegének ($25 \text{ mBTC} + 24.5 \text{ mBTC} = 49.5 \text{ mBTC}$) és a bemenetnek (50 mBTC) a különbségét tranzakciós díjként a bányászok kapják.

A `createrawtransaction` parancsot használjuk a fenti tranzakció létrehozására. A `createrawtransaction`-nak paraméterként a tranzakció bemenetet adjuk meg (vagyis a megerősített tranzakciónkból az 50 mBTC elköltetlen kimenetet), valamint a két tranzakció kimenetet (az új címre küldött pénzt, és az előző címre visszaküldött visszajáró pénzt):

```
$ bitcoin-cli createrawtransaction '[{"txid" :
"9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3", "vout" :
0}', '{"1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb": 0.025,
"1hVzSofGwT8cjb8JU7nBsCSfEVQX5u9CL": 0.0245}]'
```

```
0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c0
000000000fffffff02a0252600000000001976a914d90d36e98f62968d2bc9bbd68107564a
156a9bcf88ac50622500000000001976a91407bdb518fa2e6089fd810235cf1100c9c13d1fd
288ac00000000
```

A `createrawtransaction` parancs egy nyers hexadecimális stringet hoz létre, amely az általunk megadott tranzakció részleteit kódolja. Ellenőrizzük, hogy minden rendben van-e! Dekódoljuk ezt a nyers stringet a `decoderawtransaction` parancssal:

```
$ bitcoin-cli decoderawtransaction
0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c0
000000000fffffff02a0252600000000001976a914d90d36e98f62968d2bc9bbd68107564a
156a9bcf88ac50622500000000001976a91407bdb518fa2e6089fd810235cf1100c9c13d1fd
288ac00000000
{
    "txid" :
"0793299cb26246a8d24e468ec285a9520a1c30fc5b6125a102e3fc05d4f3cba",
    "version" : 1,
```

```

    "locktime" : 0,
    "vin" : [
        {
            "txid" :
"9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
            "vout" : 0,
            "scriptSig" : {
                "asm" : "",
                "hex" : ""
            },
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 0.02500000,
            "n" : 0,
            "scriptPubKey" : {
                "asm" : "OP_DUP OP_HASH160
d90d36e98f62968d2bc9bbd68107564a156a9bcf OP_EQUALVERIFY OP_CHECKSIG",
                "hex" :
"76a914d90d36e98f62968d2bc9bbd68107564a156a9bcf88ac",
                "reqSigs" : 1,
                "type" : "pubkeyhash",
                "addresses" : [
                    "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb"
                ]
            }
        },
        {
            "value" : 0.02450000,
            "n" : 1,
            "scriptPubKey" : {
                "asm" : "OP_DUP OP_HASH160
07bdb518fa2e6089fd810235cf1100c9c13d1fd2 OP_EQUALVERIFY OP_CHECKSIG",
                "hex" :
"76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
                "reqSigs" : 1,
                "type" : "pubkeyhash",
                "addresses" : [
                    "1hvzSofGwt8cjb8JU7nBsCSfEVQX5u9CL"
                ]
            }
        }
    ]
}

```

A tranzakció helyesnek látszik! Az új tranzakció „elfogyasztja” a megerősített tranzakció elköltetlen kimenetét, majd elkölti azt két kimenetben, ezek egyike 25 mBTC az új címre, míg a másik 24.5 mBTC visszajáró pénz az eredeti címre. A 0.5 mBTC-s különbség jelenti a tranzakciós díjat, mely annak a bányásznak lesz jóváírva, aki rábukkan a tranzakciót is magába foglaló blokkra.

Mint azt észrevehették, a tranzakció egy üres `scriptSig`-et tartalmaz, mivel még nincs aláírva. Aláírás nélkül a tranzakció értelmetlen, még nem bizonyítottuk, hogy a *miénk* az a cím, amelyből az elköltetlen kimenet származik. Az aláírás révén eltávolítjuk a kimeneten lévő akadályt, és bizonyítjuk, hogy a kimenet a mi birtokunkban van és el tudjuk költeni. A

`signrawtransaction` parancsot használjuk a tranzakció aláírására. Paraméterként a nyers tranzakció hexadecimális stringjét adjuk meg neki.



A titkosított pénztárcákat az aláírás előtt ki kell nyitni, mivel az aláíráshoz szükség van a pénztárcában lévő titkos kulcsokra.

```
$ bitcoin-cli wallet passphrase foo 360
$ bitcoin-cli signrawtransaction
0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c0
00000000fffffff02a0252600000000001976a914d90d36e98f62968d2bc9bbd68107564a
156a9bcf88ac50622500000000001976a91407bdb518fa2e6089fd810235cf1100c9c13d1fd
288ac00000000
{
    "hex" :
"0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c
000000006a47304402203e8a16522da80cef66bacfb0c800c6d52c4a26d1d86a54e0a1b76d
661f020c9022010397f00149f2a8fb2bc5bca52f2d7a7f87e3897a273ef54b277e4af52051a
06012103c9700559f690c4a9182faa8bed88ad8a0c563777ac1d3f00fd44ea6c71dc5127ffff
fffff02a0252600000000001976a914d90d36e98f62968d2bc9bbd68107564a156a9bcf88ac
50622500000000001976a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac0000000
0",
    "complete" : true
}
```

A `signrawtransaction` parancs egy másik hexadecimálisan kódolt nyers tranzakciót ad vissza. Ha látni szeretnénk, hogy mi változott, a `decoderawtransaction`-nal kódolható vissza:

```
$ bitcoin-cli decoderawtransaction
0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c0
0000006a47304402203e8a16522da80cef66bacfb0c800c6d52c4a26d1d86a54e0a1b76d6
61f020c9022010397f00149f2a8fb2bc5bca52f2d7a7f87e3897a273ef54b277e4af52051a0
6012103c9700559f690c4a9182faa8bed88ad8a0c563777ac1d3f00fd44ea6c71dc5127ffff
fffff02a0252600000000001976a914d90d36e98f62968d2bc9bbd68107564a156a9bcf88ac5
06225000000000001976a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac00000000
{
    "txid" :
"ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584ac17b346",
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "txid" :
"9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
            "vout" : 0,
            "scriptSig" : {
                "asm" :
"304402203e8a16522da80cef66bacfb0c800c6d52c4a26d1d86a54e0a1b76d661f020c902
2010397f00149f2a8fb2bc5bca52f2d7a7f87e3897a273ef54b277e4af52051a0601
03c9700559f690c4a9182faa8bed88ad8a0c563777ac1d3f00fd44ea6c71dc5127",
                "hex" :
"47304402203e8a16522da80cef66bacfb0c800c6d52c4a26d1d86a54e0a1b76d661f020c9
022010397f00149f2a8fb2bc5bca52f2d7a7f87e3897a273ef54b277e4af52051a06012103c
9700559f690c4a9182faa8bed88ad8a0c563777ac1d3f00fd44ea6c71dc5127"
            },
            "sequence" : 4294967295
        }
    }
```

```

        ],
        "vout" : [
            {
                "value" : 0.02500000,
                "n" : 0,
                "scriptPubKey" : {
                    "asm" : "OP_DUP OP_HASH160
d90d36e98f62968d2bc9bbd68107564a156a9bcf OP_EQUALVERIFY OP_CHECKSIG",
                    "hex" :
"76a914d90d36e98f62968d2bc9bbd68107564a156a9bcf88ac",
                    "reqSigs" : 1,
                    "type" : "pubkeyhash",
                    "addresses" : [
                        "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb"
                    ]
                }
            },
            {
                "value" : 0.02450000,
                "n" : 1,
                "scriptPubKey" : {
                    "asm" : "OP_DUP OP_HASH160
07bdb518fa2e6089fd810235cf1100c9c13d1fd2 OP_EQUALVERIFY OP_CHECKSIG",
                    "hex" :
"76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
                    "reqSigs" : 1,
                    "type" : "pubkeyhash",
                    "addresses" : [
                        "1hvzSofGwT8cjB8JU7nBsCSfEVQX5u9CL"
                    ]
                }
            }
        ]
    }
}

```

Most a tranzakcióban használt bemenetek egy `scriptSig`-et is tartalmaznak. A `scriptSig` egy digitális aláírás, ami bizonyítja az `1hzv...` cím tulajdonjogát, és megszünteti a kimeneten lévő akadályt, ami ezáltal elkölhetővé válik. Az aláírás a tranzakciót a bitcoin hálózat bármely csomópontja által ellenőrizhetővé teszi.

Most küldjük el ezt az újonnan létrehozott tranzakciót a hálózatnak. Ezt a `sendrawtransaction` parancssal fogjuk megenni, amelynek a paramétere a `signrawtransaction` által létrehozott nyers hexadecimális string lesz. Ez ugyanaz a string, amit épp most dekódoltunk:

```
$ bitcoin-cli sendrawtransaction
0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c0
0000006a47304402203e8a16522da80cef66bacfb0c800c6d52c4a26d1d86a54e0a1b76d6
61f020c9022010397f00149f2a8fb2bc5bca52f2d7a7f87e3897a273ef54b277e4af52051a0
6012103c9700559f690c4a9182faa8bed88ad8a0c563777ac1d3f00fd44ea6c71dc5127ffff
ffff02a0252600000000001976a914d90d36e98f62968d2bc9bbd68107564a156a9bcf88ac5
06225000000000001976a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac00000000
ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584ac17b346
```

A `sendrawtransaction` parancs egy *tranzakció hash-t* (*txid-t*) ad vissza, miután feladta a tranzakciót a hálózatnak. Ekkor a `gettransaction` parancssal le tudjuk kérdezni ezt a tranzakciót:

```

$ bitcoin-cli gettransaction
ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584ac17b346
{
    "amount" : 0.00000000,
    "fee" : -0.00050000,
    "confirmations" : 0,
    "txid" :
"ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584ac17b346",
    "time" : 1392666702,
    "timereceived" : 1392666702,
    "details" : [
        {
            "account" : "",
            "address" : "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb",
            "category" : "send",
            "amount" : -0.02500000,
            "fee" : -0.00050000
        },
        {
            "account" : "",
            "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
            "category" : "send",
            "amount" : -0.02450000,
            "fee" : -0.00050000
        },
        {
            "account" : "",
            "address" : "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb",
            "category" : "receive",
            "amount" : 0.02500000
        },
        {
            "account" : "",
            "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
            "category" : "receive",
            "amount" : 0.02450000
        }
    ]
}

```

Mint korábban, most is részletesebben megvizsgálhatjuk ezt a tranzakciót a `getrawtransaction` és a `decoderawtransaction` parancsokkal. Ezek a parancsok pontosan ugyanazt a hexadecimális stringet adják vissza, mint amit az előtt állítottunk elő és dekódoltunk, mielőtt elküldtük volna a tranzakciót a hálózatnak.

Alternatív kliensek, könyvtárak és eszközök

A bitcoind referencia kliensen kívül vannak más kliensek és könyvtárak is, melyekkel kapcsolatba léphetünk a bitcoin hálózattal és az adatstruktúrákkal. Ezek különféle programozási nyelveken lettek megvalósítva, így a programozóknak az általuk használt programozási nyelven kínálnak natív interfészeket.

Alternatív megvalósítás többek között:

[libbitcoin](#)

Bitcoin Cross-Platform C++ Development Toolkit

[bitcoin explorer](#)

Bitcoin parancssori eszköz

[bitcoin server](#)

Bitcoin teljes csomópont és lekérdező szerver

[bitcoinj](#)

Teljes csomópontot megvalósító Java kliens könyvtár

[btcd](#)

Go nyelvű, teljes csomópontot megvalósító bitcoin kliens

[Bits of Proof \(BOP\)](#)

A bitcoin Java enterprise-zal történő megvalósítása

[picocoin](#)

Egy C-ben megvalósított pehelysúlyú bitcoin kliens könyvtár

[pybitcointools](#)

Python bitcoin könyvtár

[pycoin](#)

Egy másik Python bitcoin könyvtár

Számos programozási nyelven sok további könyvtár létezik, és állandóan újabbak születnek.

A Libbitcoin és a Bitcoin Explorer

The libbitcoin könyvtár többféle platformon használható C++ fejlesztő eszköz, amely a egy teljes libbitcoin szerver csomópontot és a Bitcoin Explorer (bx) parancssori eszközt valósítja meg.

A bx parancsoknak nagyon sok olyan tulajdonságuk van, mint a fejezetben szemléltetett bitcoind parancsoknak. A bx parancsok között vannak olyan kulcs kezelési és kulcs átalakító eszközök, melyek a bitcoind-ból hiányoznak, pl. a 2-es típusú determinisztikus kulcsok kezelése, a mnemonikus kulcsok kódolása, a lopakodó címek kezelése, fizetési és lekérdezési támogatás.

A Bitcoin Explorer installálása

A Bitcoin Explorer installálásához [töltse le a megfelelő operációs rendszerhez tartozó aláírt végrehajtható programot](#). Az éles és a teszt hálózatot használó Linux, OS X, és Windows programok találhatók itt.

A bx paraméterek nélkül begépelésével tudja kiíratni a rendelkezésre álló parancsokat (lásd [D. függelék](#)).

A Bitcoin Explorer installálával [a Linux és OS X rendszereken forrásból tudunk fordítani](#), [Windowsban pedig Visual Studio projektekkel](#). A források az Autotools használatával kézzel is lefordíthatók. Ezek a libbitcoin könyvtári függőséget is installálják.



A Bitcoin Explorer-ben sok hasznos parancs van a címek kódolására és dekódolására, valamint különféle formátumok és ábrázolások közötti átalakításokra. Segítségükkel a különféle formátumok, pl. a Base16 (hexadecimális), Base58, Base58Check, Base64, stb. vizsgálhatók.

A Libbitcoin installálása

A libbitcoin könyvtár installálával [a Linux és OS X rendszereken forrásból tudunk fordítani](#), [Windows-ban pedig Visual Studio projektekkel](#). A források az Autotools használatával kézzel is lefordíthatók.



A Bitcoin Explorer a bx-et és a libbitcoin könyvtárat egyaránt installálja, ezért ha a bx-et forrásból fordítottuk újra, akkor ezt a lépést átugorhatjuk.

pycoin

A [pycoin](#) Python könyvtárat eredetileg Richard Kiss írta és tartotta karban. A könyvtár a bitcoin kulcsok és tranzakciók kezelését támogatja, és még a script nyelvet támogatja annyira, hogy a nem szabványos tranzakciók kezelése is lehetséges benne.

A pycoin könyvtár mind a Python 2 (2.7.x), mind a Python 3 (3.3 utáni verziók) támogatására képes, és hasznos parancssori segédprogramjai vannak, pl. a ku és a tx. Ha szeretnénk a pycoin 0.42-t Python 3 alatt, virtuális környezetben (venv) installálni, használjuk a következőket:

```
$ python3 -m venv /tmp/pycoin
$ . /tmp/pycoin/bin/activate
$ pip install pycoin==0.42
Downloading/unpacking pycoin==0.42
  Downloading pycoin-0.42.tar.gz (66kB): 66kB downloaded
  Running setup.py (path:/tmp/pycoin/build/pycoin/setup.py) egg_info for
  package pycoin

Installing collected packages: pycoin
  Running setup.py install for pycoin
```

```

Installing tx script to /tmp/pycoin/bin
Installing cache_tx script to /tmp/pycoin/bin
Installing bu script to /tmp/pycoin/bin
Installing fetch_unspent script to /tmp/pycoin/bin
Installing block script to /tmp/pycoin/bin
Installing spend script to /tmp/pycoin/bin
Installing ku script to /tmp/pycoin/bin
Installing genwallet script to /tmp/pycoin/bin
Successfully installed pycoin
Cleaning up...
$
```

Az alábbiakban egy minta Python script látható, amely a pycoin könyvtár segítségével küld bitcoinokat:

```

$ cat pycoin_example.py
#!/usr/bin/env python

from pycoin.key import Key

from pycoin.key.validate import is_address_valid, is_wif_valid
from pycoin.services import spendables_for_address
from pycoin.tx.tx_utils import create_signed_tx

def get_address(which):
    while 1:
        print("enter the %s address=> " % which, end=' ')
        address = input()
        is_valid = is_address_valid(address)
        if is_valid:
            return address
        print("invalid address, please try again")

src_address = get_address("source")
spendables = spendables_for_address(src_address)
print(spendables)

while 1:
    print("enter the WIF for %s=> " % src_address, end=' ')
    wif = input()
    is_valid = is_wif_valid(wif)
    if is_valid:
        break
    print("invalid wif, please try again")

key = Key.from_text(wif)
if src_address not in (key.address(use_uncompressed=False),
key.address(use_uncompressed=True)):
    print("** WIF doesn't correspond to %s" % src_address)
print("The secret exponent is %d" % key.secret_exponent())

dst_address = get_address("destination")

tx = create_signed_tx(spendables, payables=[dst_address], wifs=[wif])

print("here is the signed output transaction")

print(tx.as_hex())
```

A ku és tx parancssori eszközök használatát lásd a „[C. függelék: pycoin, ku és tx](#)” részben.

btcd

A btcd egy Goban megírt, teljes csomópontot megvalósító bitcoin implementáció. Jelenleg helyesen letölти, ellenőri és kiszolgálja a blokkláncot. A blokkok elfogadása a referencia implementáció, a bitcoind pontos szabályai szerint történik (beleértve a hibákat is). Helyesen továbbítja az újonnan kibányászott blokkokat is, karban tartja a tranzakciós készletet és továbbítja azokat a tranzakciókat, melyek még nem lettek blokkba foglalva. Biztosítja, hogy a tranzakciós készletbe (poolba) befogadott összes tranzakció teljesítse a blokklánc által megkövetelt szabályokat, és azoknak a szigorú ellenőrzéseknek a túlnyomó többségét is tartalmazza, amelyek a bányászat követelményei szerint szűrik a tranzakciókat („szabványos” tranzakciók).

A btcd és a bitcoind közötti egyik lényeges különbség az, hogy a btcdben tudatos tervezési döntés eredményeképpen nincs pénztárca funkció. Ez azt jelenti, hogy közvetlenül a btcdvel nem lehet fizetni vagy pénzt fogadni. Ezt a funkciót a btcwallet és a btcgui projektek biztosítják, mindenkor aktív fejlesztés alatt áll. Egy másik figyelemre méltó különbség az, hogy a btcd nem csak a HTTP POST kéréseket támogatja (a bitcoind-hez hasonlóan), hanem a Websocketeket is (ezt részesíti előnyben), és a btcd RPC összeköttetéseinek a TLS alapértelemben engedélyezve van.

A btcd installálása

A btcd Windows alatti installálása az alábbi msi letöltésével és futtatásával lehetséges: [GitHub](#). Linux alatt a következő parancs futtatható, feltéve, hogy a Go nyelv már installálva van:

```
$ go get github.com/conformal/btcd/...
```

Ha a btcd-t szeretné a legfrissebb verzóira frissíteni, futtassa a következőt:

```
$ go get -u -v github.com/conformal/btcd/...
```

A btcd használata

A btcdnek számos beállítási lehetősége van. Ezeket a következőképpen lehet megnézni:

```
$ btcd --help
```

A btcd egy btcctl parancssori segédprogrammal is rendelkezik. Ennek segítségével a btcd RPC-n kereszthü beállítása vagy lekérdező parancsai használhatók. A btcd alapértelemben nem engedélyezi az RPC szerverét, emiatt minimálisan egy RPC felhasználó név és jelszó beállítására van szükség:

- *btcd.conf*:

```
[Application Options]
rpcuser=myuser
rpcpass=SomeDecentp4ssw0rd
```

- *btcctl.conf*:

```
[Application Options]
rpcuser=myuser
rpcpass=SomeDecentp4ssw0rd
```

Ha a parancssorból szeretné felülbírálni a konfigurációs állományban lévő értékeket, akkor:

```
$ btcctl -u myuser -P SomeDecentp4ssw0rd
$ btcctl -u myuser -P SomeDecentp4ssw0rd
```

A rendelkezésre álló lehetőségek a következőképpen kilistázhatók ki:

```
$ btcctl --help
```

4. Kulcsok, címek, pénztárcák

Bevezetés

A bitcoinban a tulajdonjogot a *digitális kulcsok*, a *bitcoin címek* és a *digitális aláírások* teremtik meg. A digitális kulcsokat nem a hálózat tárolja, hanem a végfelhasználók hozzák létre és tárolják őket – vagy egy állományban vagy egy egyszerű adatbázisban, melynek *pénztárca* a neve. A felhasználó pénztárcájában lévő digitális kulcsok teljesen függetlenek a bitcoin protokolltól, a kulcsok a felhasználó pénztárca szoftverével a blokkláncra történő hivatkozás vagy Internet hozzáférés nélkül állíthatók elő és kezelhetők. A kulcsok valósítják meg a bitcoin sok érdekes tulajdonságát, többek között a decentralizált bizalmat és felügyeletet, a tulajdonjog igazolását és a kriptográfiai helyes biztonsági modellt.

Minden egyes bitcoin tranzakciónak érvényes aláírással kell rendelkeznie, csak ekkor lesz befoglalva a blokkláncba. Az aláírások viszont csak érvényes digitális kulcsokkal állíthatók elő, ezért ha valaki rendelkezik ezekkel a kulcsokkal, akkor rendelkezik a számlán lévő bitcoinokkal is. A kulcsok párokból állnak, az egyik a titkos (privát) kulcs, a másik a nyilvános kulcs. A nyilvános kulcsra gondoljanak úgy, mint egy bankszámla számra, a titkos kulcsra pedig úgy, mint egy titkos PIN kódra, vagy egy csekken lévő aláírástra, amely megteremti a számla feletti felügyeletet. Ezeket a digitális kulcsokat a bitcoin felhasználók ritkán látják. Többnyire a pénztárca állományban vannak tárolva, és a pénztárca szoftver kezeli őket.

A bitcoin tranzakciók kifizetésekhez tartozó részében a címzett nyilvános kulcsát egy *bitcoin címnek* nevezett digitális ujjlenyomat képviseli, amelynek ugyanaz a szerepe, mint egy csekken a kedvezményezett nevének. A legtöbb esetben a bitcoin cím egy nyilvános kulcsból jön létre, és megfelel a nyilvános kulcsnak. De nem minden bitcoin cím felel meg egy nyilvános kulcsnak, a címek más kedvezményezetteket, pl. scripteket is képviselhetnek, amint azt a fejezet későbbi részében látni fogjuk. Ily módon a bitcoin címek a pénzösszeg címzettjeit személyesítik meg, és a papír alapú csekkekhez hasonlóan a tranzakciókat rugalmassá teszik: ugyanaz a fizetési eszköz használható magánszemélyek vagy cégek számláinál, be- vagy kifizetésre. A bitcoin cím a kulcsok egyetlen olyan megjelenési formája, amivel a felhasználók rendszeresen találkoznak, mivel ezt kell megosztaniuk a nagyvilággal.

Ebben a fejezetben a kulcsokkal és az őket tartalmazó pénztárcákkal ismerkedünk meg. Megvizsgáljuk, hogyan történik a kulcsok létrehozása, tárolása és kezelése. Áttekintjük a különféle kódolási formátumokat, melyek a titkos és nyilvános kulcsok, címke és script címek ábrázolására szolgálnak. Végül a kulcsok különleges felhasználásait tekintjük át, melyek a következők: üzenetek aláírása, a tulajdonjog bizonyítása, kérkedő címek (vanity address) és papír alapú pénztárcák létrehozása.

A nyilvános kulcsú titkosítás és a digitális pénz

A nyilvános kulcsú titkosítást az 1970-es években fedezték fel. A számítógép- és információbiztonság matematikai alapjait a nyilvános kulcsú titkosítás képzi.

A nyilvános kulcsú titkosítás felfedezése óta számos, e célra alkalmas matematikai függvényt fedeztek föl, pl. a prímszámok hatványozását vagy az elliptikus görbéken történő szorzást. Ezek a matematikai függvények gyakorlatilag megfordíthatatlanok, ami azt jelenti, hogy

könnyű őket az egyik irányban kiszámítani, de ez a másik irányban gyakorlatilag lehetetlen. A titkosítás ezekre a matematikai függvényekre épül, és lehetővé teszi a digitális titkok és a nem hamisítható digitális aláírások létrehozását. A bitcoin elliptikus görbén történő szorzást használ a nyilvános kulcsú titkosításra.

A bitcoinnál nyilvános kulcsú titkosítást használunk egy olyan kulcspár létrehozására, amely a bitcoinokhoz történő hozzáférést szabályozza. A kulcspár egy titkos kulcsból és a belőle származó egyedi nyilvános kulcsból áll. A nyilvános kulcs szolgál a bitcoinok fogadására, a titkos kulcs szolgál a tranzakciók aláírására és a bitcoinok elköltésére.

A nyilvános és titkos kulcs között van egy matematikai összefüggés, ami lehetővé teszi, hogy a titkos kulcsokkal üzenetek aláírását állítsuk elő. Ez az aláírás a nyilvános kulccsal úgy ellenőrizhető, hogy közben nincs szükség a titkos kulcs felfedésére.

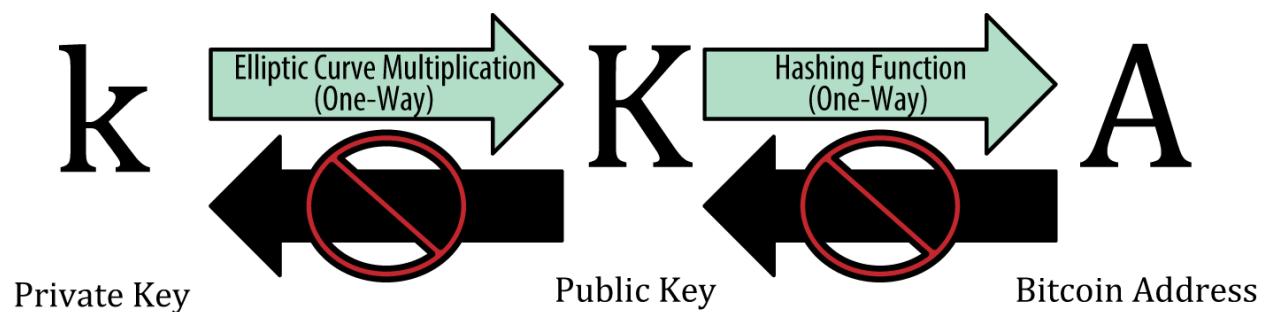
Ha egy bitcoin tulajdonos el akarja költeni a bitcoinjait, akkor egy tranzakcióban bemutatja a nyilvános kulcsát és egy aláírást (ami minden egyes alkalommal különböző, de ugyanabból a titkos kulcsból áll elő). A nyilvános kulcs és az aláírás révén a bitcoin hálózat bármelyik tagja ellenőrizni tudja a tranzakciót, meg tudja állapítani, hogy érvényes-e, és meg tud bizonyosodni arról, hogy a bitcoinokat küldő személy valóban birtokolta-e őket a küldés idején.



A legtöbb megvalósítás a titkos és nyilvános kulcsokat az egyszerűség kedvéért együtt, egy *kulcspárként* tárolja. Mivel a nyilvános kulcs előállítása a titkos kulcs ismeretében triviális feladat, ezért az is előfordulhat, hogy a pénztárcában csupán a titkos kulcs van tárolva.

Titkos és nyilvános kulcsok

A bitcoin pénztárca kulcspárok halmazát tartalmazza. Mindegyik kulcspár egy titkos és egy nyilvános kulcsból áll. A (k) titkos kulcs általában egy véletlenszerűen választott szám. A titkos kulcsból elliptikus görbén történő szorzással, ami egy egyirányú titkosító függvény, egy (K) nyilvános kulcsot állítunk elő. A (K) nyilvános kulcsból egy egyirányú titkosító hash függvényel egy (A) bitcoin címet állítunk elő. Ebben a részben először egy titkos kulcsot fogunk előállítani, majd megnézzük, hogy az elliptikus görbén milyen matematikai műveletekkel lehet a titkos kulcsot nyilvános kulcsá átalakítani, és végül a nyilvános kulcsból egy bitcoin címet állítunk elő. A titkos kulcs, nyilvános kulcs és bitcoin cím közötti összefüggést a 4-1. ábra mutatja:



4-1. ábra: Titkos kulcs, nyilvános kulcs és bitcoin cím

Titkos kulcsok

A titkos kulcs egyszerűen egy véletlenszerűen választott szám. A titkos kulcs birtoklása az alapja annak, hogy a felhasználó rendelkezni tudjon az összes olyan pénz fölött, amely a titkos kulcsnak megfelelő bitcoin címhez tartozik. A titkos kulcs szolgál aláírások létrehozására. Az aláírás azt a célt szolgálja, hogy a felhasználó bizonyítani tudja a tranzakcióban szereplő összegek tulajdonjogát, mielőtt elkölti őket. A titkos kulcsnak egész idő alatt titokban kell maradnia, mert felfedése egy harmadik fél számára azzal lenne egyenértékű, mint ha hozzáférést adnánk neki azokhoz a bitcoinokhoz, melyeket ez a kulcs biztosít. A titkos kulcsról biztonsági másolatot kell készíteni, és védeni kell, nehogy véletlenül elveszítsük, mert ha elvész, akkor nem tudjuk semmi másból visszaállítani, és az általa biztosított összegek is örökre elvesznek.



A bitcoin titkos kulcs csupán egy szám. A titkos kulcs véletlenszerűen, pl. egy kockával, egy darab papírral és ceruzával is előállítható. Dobunk fel egy pénzérmét 256-szor, és írjuk le a dobások eredményét egy bináris szám formájában. A nyilvános kulcs ezt követően a titkos kulcsból állítható elő.

A titkos kulcs előállítása egy véletlen számból

A kulcsok előállításának első és legfontosabb lépése, hogy egy biztonságos entrópiaforrást, másképpen véletlenszerű forrást találunk. Egy bitcoin kulcs előállítása lényegében egyenértékű azzal, hogy „Válasszunk egy számot 1 és 2^{256} között”. Hogy pontosan hogyan választjuk ezt a számot, az nem számít, feltéve, hogy a választás nem megjósolható vagy nem megismételhető. A bitcoin szoftver a mögöttes operációs rendszer véletlenszám generátorát használja 256 bit entrópia (véletlenszerűség) előállítására. Az OS véletlenszám generátorát általában egy emberi eredetű entrópiaforrással inicializálják, ezért van szükség pl. arra, hogy mozgassuk az egeret néhány másodpercig. Az igazán paranoiások számára a dobókockánál, a papírnál és a ceruzánál nincs jobb módszer.

Pontosabban, a titkos kulcs egy $1 \dots n - 1$ közötti tetszőleges szám lehet, ahol n konstans ($n = 1.158 * 10^{77}$, vagyis egy kicsit kevesebb, mint 2^{256}), és a bitcoinnal használt elliptikus görbe rendszámával egyenlő (lásd az „[Elliptikus görbükkkel történő titkosítás](#)” részt). Egy ilyen kulcs előállításához véletlenszerűen válasszunk egy 256 bites számot, és ellenőrizzük, hogy kisebb-e $n - 1$ -nél. Programozási szempontból ezt általában úgy valósítható meg, hogy kriptográfiailag biztonságos véletlen forrásból származó bitek egy nagyobb halmazát egy SHA256 hash algoritmussal összetömörítjük, és ezzel egyszerűen létrehozunk egy 256 bites számot. Ha az eredmény kisebb, mint $n - 1$, akkor a titkos kulcs megfelelő. Ha nem, akkor egy másik véletlen számmal próbálkozunk.



Ne próbálkozzanak azzal, hogy saját pszeudó-véletlenszám generátort (PRNG, pseudo random number generator) írnak, vagy a kedvenc programozási nyelvük által felkínált „egyszerű” véletleszám generátort használják. Használjanak egy kriptográfiailag biztonságos pszeudó-véletlenszám generátort (CSPRNG, cryptographically-secure pseudo-random number generator), melynek magja megfelelő entrópiaforrásból származik. A CSPRNG helyes megvalósítása a kulcsok biztonsága szempontjából kritikus fontosságú.

Alább egy véletlenszerűen előállított (k) titkos kulcs látható hexadecimális formátumban (256 bináris számjegy, 64 hexadecimális számjeggyel ábrázolva, ahol minden egyik hexadecimális számjegy 4 bitnek felel meg):

```
1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
```



A titkos kulcsok száma a bitcoin esetén 2^{256} , ami egy elképzelhetetlenül nagy szám. Decimálisan kb. 10^{77} . A látható világgyetemben becslések szerint kb. 10^{80} atom van.

A Bitcoin Core kliensben (lásd [3. fejezet](#)) egy új kulcs a `getnewaddress` parancssal állítható elő. Biztonsági okokból a kliens csak a nyilvános kulcsot jelzi ki, a titkos kulcsot nem. Ha azt szeretnénk, hogy `bitcoind` írja ki a titkos kulcsot, használjuk a `dumpprivkey` parancsot. A `dumpprivkey` a titkos kulcs kijelzésére ellenőrző összeggel kiegészített 58-as számrendszeri formátumot használ, melyet tárca import formátumnak (*WIF, wallet import format*) hívunk. A WIF formátumot a „[Titkos kulcs formátumok](#)” részben fogjuk részletesebben megvizsgálni. Az előző két parancssal a következőképpen lehet egy titkos kulcsot előállítani és kijelezni:

```
$ bitcoind getnewaddress  
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy  
$ bitcoind dumpprivkey 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy  
KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

A `dumpprivkey` kinyitja a pénztárcát és kiveszi belőle a `getnewaddress` parancs által előállított titkos kulcsot. A `bitcoind` csak akkor képes a nyilvános kulcsnak tartozó titkos kulcs kiírására, ha a pénztárcában minden kettő tárolva van.



A `dumpprivkey` parancs a nyilvános kulcsból nem képes előállítani a titkos kulcsot, mivel ez lehetetlen. A parancs egyszerűen csak felfedi azt a titkos kulcsot, amelyet a pénztárca már ismer, és amely a `getnewaddress` parancssal lett előállítva.

Titkos kulcsok előállítása és kijelzése a Bitcoin Explorer parancssori eszközzel is lehetséges, (lásd „[A Libbitcoin és a Bitcoin Explorer](#)”). Az ehhez szükséges parancsok: `seed`, `ec-new` és `ec-to-wif`:

```
$ bx seed | bx ec-new | bx ec-to-wif  
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

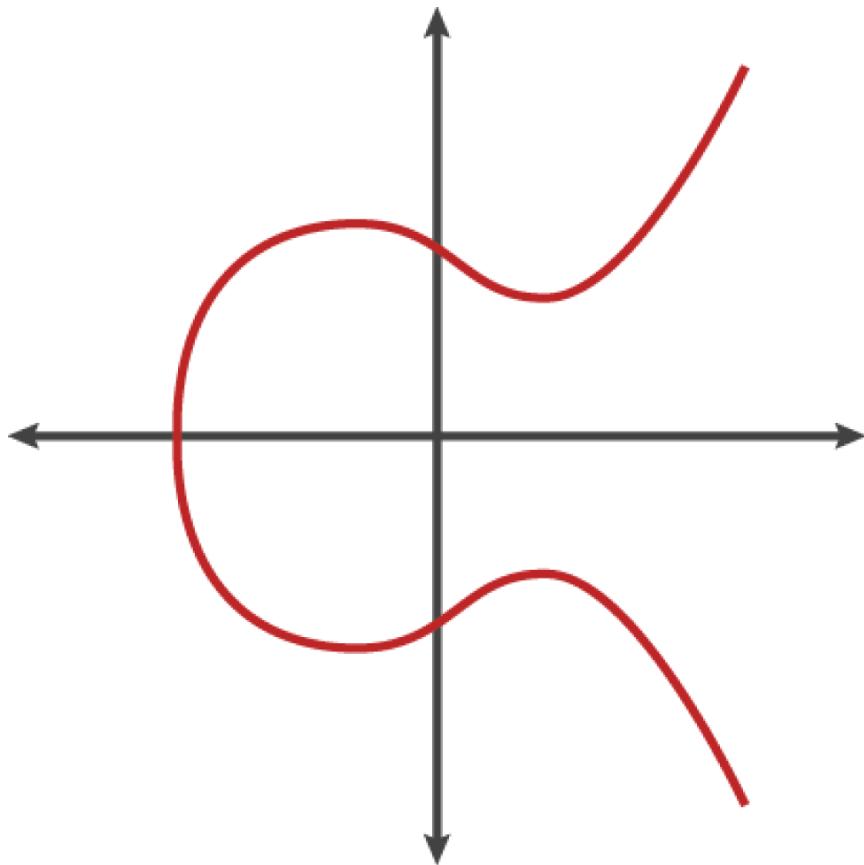
Nyilvános kulcsok

A nyilvános kulcs a titkos kulcsból, az elliptikus görbén történő szorzással számítható ki: $K = k * G$, ahol k a titkos kulcs, G az ún. *generátor pont*, és K az eredményként kapott nyilvános kulcs. Az ellentétes művelet, az ún. „diszkrét logaritmus meghatározása” – vagyis a k kiszámítása, ha a K ismert – annyira nehéz, hogy egyenértékű azzal, mint ha a k összes lehetséges értékét végigpróbáltunk, vagyis olyan, mint egy nyers erőn alapuló keresés. Mielőtt szemléltetnénk, hogyan lehet a titkos kulcsból a nyilvános kulcsot előállítani, vizsgáljuk meg kicsit részletesebben az elliptikus görbékkel történő titkosítást.

Elliptikus görbekkel történő titkosítás

Az elliptikus görbekkel történő titkosítás egyfajta aszimmetrikus azaz nyilvános kulcsú titkosítás, amely egy elliptikus görbe pontjain végzett összeadás és szorzás diszkrét logaritmus problémáján alapul.

Alább egy elliptikus görbe látható, hasonló ahhoz, mint amit a bitcoin használ:



4-2. ábra: Egy elliptikus görbe

A bitcoin az Amerikai Szabványügyi Hivatal (NIST, National Institute of Standards and Technology) által, a secp256k1 szabványban definiált elliptikus görbét és matematikai konstansokat használja. A secp256k1 elliptikus görbét a következő függvény definiálja:

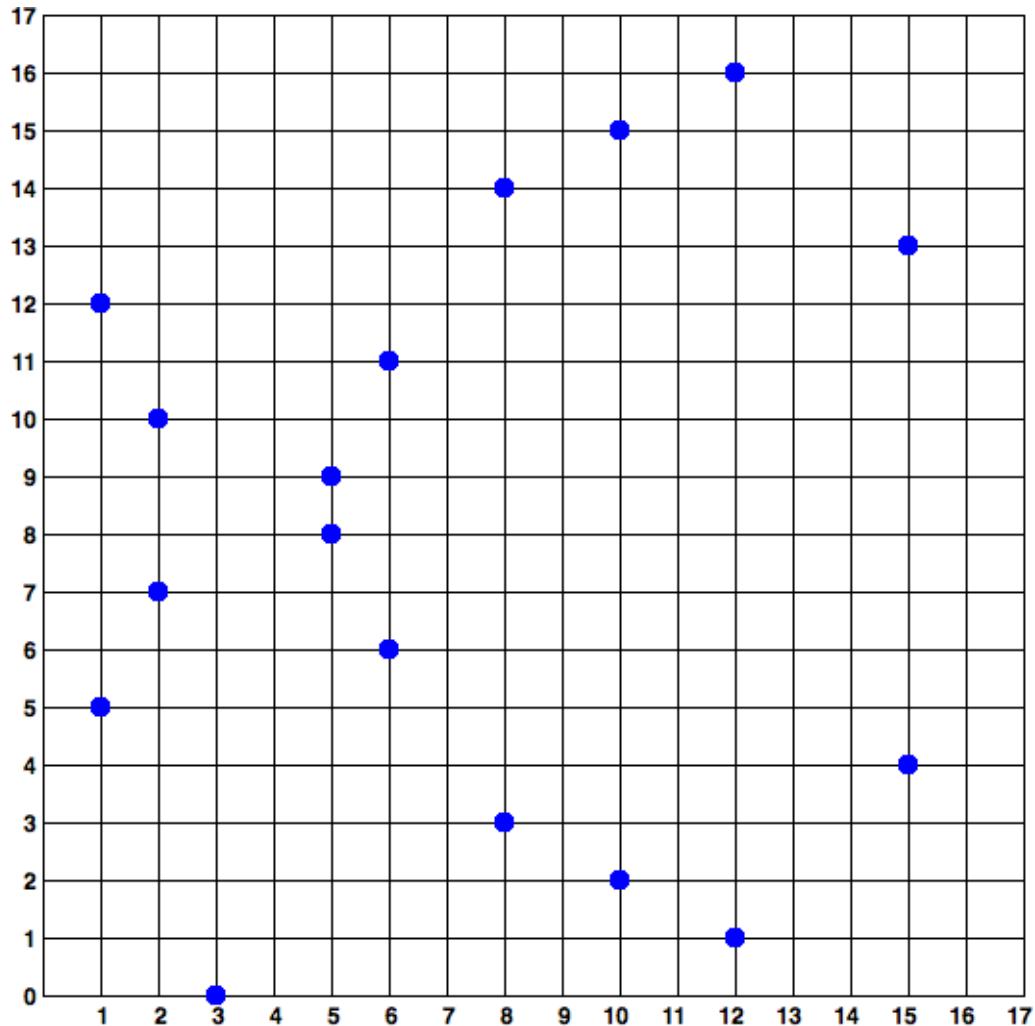
$$y^2 = (x^3 + 7) \pmod{F_p}$$

vagy

$$y^2 = (x^3 + 7) \pmod{p}$$

A \pmod{p} (p prímszám szerinti modulus) azt jelzi, hogy egy p rendszámú véges mező fölött definiált görbéről van szó, ami úgy is írható, hogy F_p , ahol $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$, egy nagyon nagy prímszám.

Mivel ez a görbe a valós számok halmaza helyett egy prím rendszámú véges mező fölött lett definiálva, úgy néz ki, mint két dimenzióban szétszórt pontok halmaza, ami nagyon nehézzé teszi a megjelenítését. A matematikája azonban megegyezik a fenti, valós számok fölött definiált elliptikus görbüvel. Például a 4-3. ábrán ugyanez az elliptikus görbe látható egy sokkal kisebb, 17 rendszámú véges mező fölött, ahol a pont-minták egy rácson lettek megjelenítve. A bitcoin secp256k1 elliptikus görbéje úgy képzelhető el, mint egy sokkal összetettebb pontminta, egy mérhetetlenül nagy rácson.



4-3. ábra: Titkosítás elliptikus görbüvel: egy $F(p)$ elliptikus görbe megjelenítése, $p=17$ esetén

Lent például egy $P(x, y)$ pont látható, amely a secp256k1 görbén van. Ezt egy Python programmal önök is ellenőrizhetik:

```
P =
(50566263022277343669578718895168534326250603453777594175500187360389116729
240,
326705100207588169780830851305070431844712733806592432759389043357573374824
24)
Python 3.4.0 (default, Mar 30 2014, 19:23:13)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

```

>>> p =
115792089237316195423570985008687907853269984665640564039457584007908834671
663
>>> x =
550662630222773436695787188951685343262506034537775941755001873603891167292
40
>>> y =
326705100207588169780830851305070431844712733806592432759389043357573374824
24
>>> (x ** 3 + 7 - y**2) % p
0

```

Az elliptikus görbék matematikája tartalmaz egy „végtelenben lévő” pontot, amely durván a 0-nak felel meg az összeadásban. A számítógépeken néha az $x = y = 0$ segítségével ábrázolják (amely nem elégít ki az elliptikus görbék egyenletét, de könnyen ellenőrizhető külön esetként kezelhető).

Van továbbá egy „összeadásnak” nevezett + művelet, amelynek néhány sajátossága hasonlít az iskolában tanult valós számok összeadásához. Ha az elliptikus görbén van két pont, P_1 és P_2 , akkor létezik egy harmadik pont, P_3 , amely szintén az elliptikus görbén van, és amelyre $P_3 = P_1 + P_2$.

Geometriailag ez a harmadik pont, a P_3 úgy számítható ki, hogy húzunk egy egyeneset a P_1 és P_2 között. Ez az egyenes az elliptikus görbét pontosan egy további helyen fogja metszeni. Nevezzük ezt a pontot P_3 -nak: $P_3 = (x, y)$. A P_3 pont ennek a pontnak az x tengelyre történő tükrözésével kapható meg: $P_3 = (x, -y)$.

Van néhány különleges eset, amely megvilágítja, miért van szükség a „végtelenben lévő pontra”.

Ha a P_1 és a P_2 pont megegyezik, akkor a P_1 és P_2 „közötti” egyenes a görbe P_1 pontbeli érintője lesz. Az érintő pontosan egy pontban fogja metszeni a görbét. A differenciáliszámítás segítségével meghatározható az érintő meredeksége. Ezek a módszerek érdekes módon még akkor is működnek, ha csak azok a görbén lévő pontok érdekelnek minket, melyeknek minden két koordinátája egész szám!

Bizonyos esetekben (pl. ha a P_1 és a P_2 x koordinátája azonos, de az y koordinátája különböző), akkor az érintő függőleges lesz, és ebben az esetben a $P_3 =$ „a végtelenben lévő pont”.

Ha P_1 a „végtelenben lévő pont”, akkor $P_1 + P_2 = P_2$. Hasonlóképpen, ha a P_2 a végtelenben lévő pont, akkor $P_1 + P_2 = P_1$. Ez mutatja, hogy a végtelenben lévő pont a 0 szerepét játssza.

$A + \text{asszociatív}$, vagyis $(A + B) + C = A + (B + C)$. Ez azt jelenti, hogy $A + B + C$ zárójelezés nélkül is egyértelmű.

Az összeadás definiálása után az szorzás a szokásos módon, az összeadás kiterjesztéseként definiálható. Az elliptikus görbén lévő P pontra, ha k egész szám, akkor $kP = P + P + \dots + P$ (k -szor). Megjegyezzük, hogy a k -t néha zavaró módon „kitevőnek” hívják.

Egy nyilvános kulcs előállítása

Kiindulópontunk egy titkos kulcs, amely egy véletlenszerűen előállított k szám, majd ezt megszorozzuk a görbe egy előre meghatározott G pontjával, a *generátor ponttal*, és ezzel egy másik pontot állítunk elő valahol a görbén, ami megfelel a K nyilvános kulcsnak. A generátor pontot a `secp256k1` szabvány definiálja, és mindegyik bitcoin kulcs esetén ugyanaz.

$$K = k * G$$

ahol k a titkos kulcs, G a generátor pont, és K az eredményként kapott nyilvános kulcs, azaz a görbe egy másik pontja. Mivel a generátor pont az összes bitcoin felhasználó esetén ugyanaz, egy k titkos kulcs G -vel vett szorzata mindenkor ugyanazt a K nyilvános kulcsot eredményezi. A k és K közötti kapcsolat rögzített, de csak az egyik irányban lehet könnyen kiszámítani, k -tól K irányában. A bitcoin címet (amely K -ból van leszármaztatva) emiatt lehet bárkivel megosztani, és emiatt nem fedi fel a felhasználó titkos kulcsát (k).



Egy titkos kulcsból kiszámítható a nyilvános kulcs, de egy nyilvános kulcsot nem lehet titkos kulccsá visszaalakítani, mert a számítás csak egy irányban működik.

Az elliptikus görbén történő szorzást úgy valósítjuk meg, hogy az előzőleg előállított k titkos kulcsot megszorozzuk a G generátor ponttal, ami a K nyilvános kulcsot eredményezi:

$$K = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD * G$$

A K nyilvános kulcs definíció szerint egy pont: $K = (x, y)$:

$$K = (x, y)$$

ahol

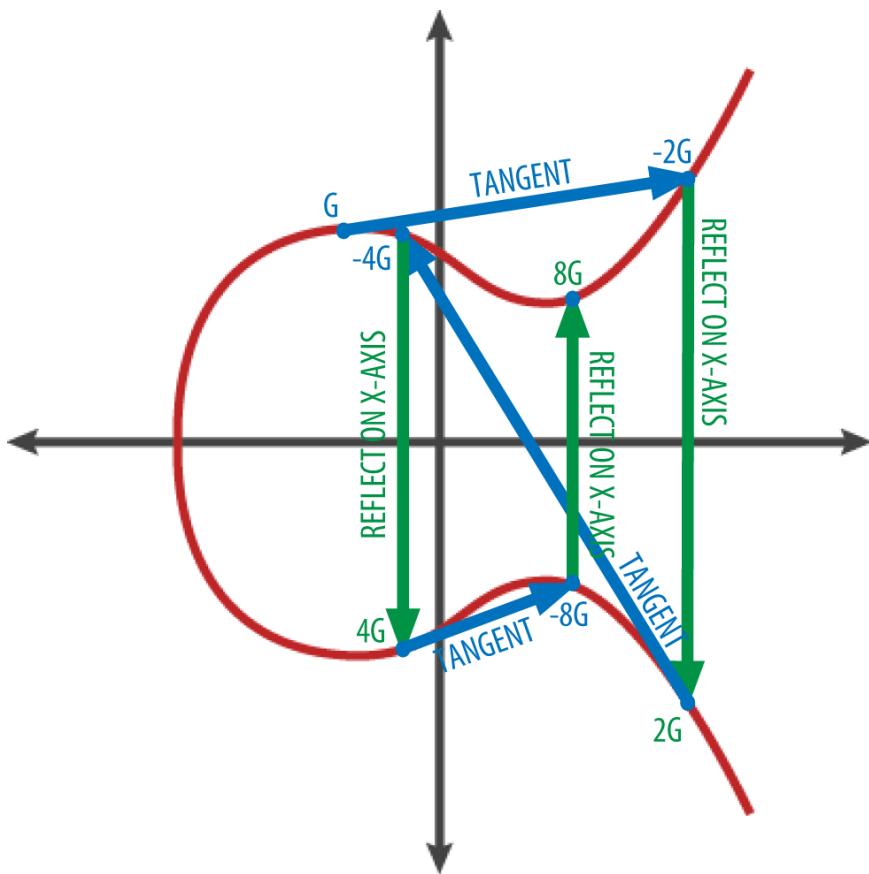
$$\begin{aligned} x &= F028892BAD7ED57D2FB57BF33081D5CF6F9ED3D3D7F159C2E2FFF579DC341A \\ y &= 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB \end{aligned}$$

Egy pont és egy egész szorzatának megjelenítésére egy egyszerűbb, valós számokon definiált elliptikus görbét fogjuk használni – a matematika ugyanaz. A célunk az, hogy előállítsuk a G generátor pont kG többszörösét. Ez ugyanaz, mint ha a G -t k -szor összeadnánk. Az elliptikus görbek esetén egy pont önmagával történő összeadása egyenértékű azzal, hogy egy érintő húzunk az adott pontban, és megkeressük, hogy hol metszi az érintő a görbét, majd ezt a pontot tükrözük az x -tengelyen.

A 4-4. ábra a G , $2G$, $4G$ előállításának folyamatát mutatja, a görbén végzett geometriai műveletek formájában.



A legtöbb bitcoin implementáció az [OpenSSL könyvtárat](#) használja az elliptikus görbekkel történő titkosításra. Például a nyilvános kulcs előállítása az `EC_PONT_mul()` függvénytelével lehetséges.



4-4. ábra: Elliptikus görbével történő titkosítás: a G pont és egy egész szorzatának megjelenítése egy elliptikus görbén

Bitcoin címek

A bitcoin cím egy számokból és betűkből álló string, amely bárkivel megosztható, aki pénz akar önknek küldeni. A nyilvános kulcsból előállított címek betűket és számokat tartalmaznak, és az „1” számjeggyel kezdődnek. Példa egy bitcoin címre:

1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZY

A tranzakciókban a bitcoin cím leggyakrabban a pénz „címzettjét” azonosítja. Ha összehasonlítjuk a bitcoin tranzakciót egy papír csekkel, akkor a bitcoin cím felel meg a kedvezményezettnek, vagyis ezt írjuk a „Kinek fizetendő” sor után. Papír csekk esetén a kedvezményezett néha egy bankszámlaszám, de lehet cégi, intézmény vagy akár pénzt is felvehetünk vele. Mivel a papír csekkekben nem kell számlaszámot megadni, csak egy absztrakt személyt, aki a pénz címzettje, ezért a papír csekkek nagyon rugalmas fizetési eszközt jelentenek. A bitcoin tranzakciók hasonló absztraktiós használnak, a bitcoin címet, ami nagyon rugalmassá teszi őket. A bitcoin cím képviselheti egy nyilvános/titkos kulcspar tulajdonosát, vagy valami mást, pl. egy scriptet, amint azt a „[Fizetés script hashnek \(P2SH\)](#)” részben látni fogjuk. Egyelőre vizsgáljuk meg azt az egyszerű esetet, amikor a bitcoin cím egy nyilvános kulcsból származik és azt képviseli.

A bitcoin cím a nyilvános kulcsból egy egyirányú kriptográfiai tömörítés (hashing) használatával áll elő. A „tömörítő algoritmus” vagy egyszerűen „hash algoritmus” egy egyirányú függvény, amely egy tetszőleges méretű bemenet esetén egy ujjlenyomatot vagy

„zanzát” (hasht) állít elő. A kriptográfiai hash függvényeket a bitcoin a bitcoin címekben, a script címekben és a bányászat „munkabizonyíték” algoritmusában széleskörűen használja. A nyilvános kulcsból a bitcoin cím előállítása a következő algoritmusokkal történik: az SHA (Secure Hash Algorithm) és a RIPEMD (RACE Integrity Primitives Evaluation Message Digest), konkrétabban az SHA256 és a RIPEMD160 segítségével.

A K nyilvános kulcsból kiindulva kisszámítjuk a kulcs SHA256 tömörítését, majd az eredmény RIPEMD160 tömörítését. Így egy 160 bites (20 bájtos) számot kapunk:

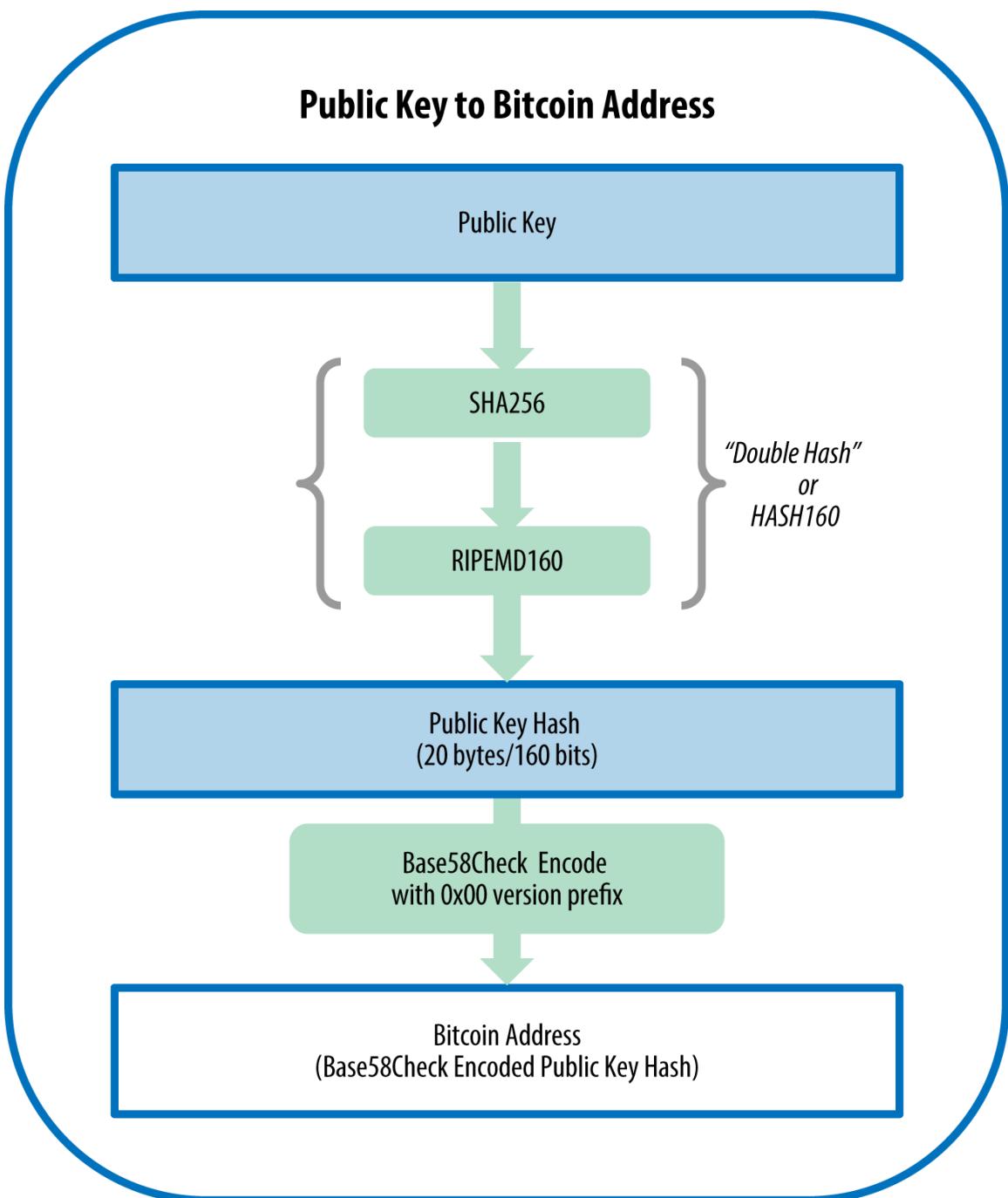
$$A = \text{RIPEMD160}(\text{SHA256}(K))$$

ahol K a nyilvános kulcs és A az eredményként kapott bitcoin cím.



Egy bitcoin cím *nem* azonos a nyilvános kulccsal. A bitcoin címek a nyilvános kulcsból származnak, egy egyirányú függvény alkalmazásával.

A bitcoin címeket a felhasználók majdnem minden „Base58Check” kódolásban látják (lásd „[Base58 és Base58Check kódolás](#)”). Ez a kódolás 58 karaktert (58-as számrendszer) használ, és egy ellenőrző összeggel van kiegészítve, ami segíti az olvashatóságot, és véd a cím beviteli és továbbítási hibák ellen. A Base58Check sok más módon is szerephez jut a bitcoinban, ha egy szám, pl. egy bitcoin cím, egy titkos kulcs, egy titkosított kulcs vagy egy script tömörítésének pontos beírására van szükség. A következő részben megvizsgáljuk a Base58Check kódolás és dekódolás működését, és az így előálló alakokat. A 4-5. ábra egy nyilvános kulcs bitcoin címmé történő átalakítását szemlélteti.



4-5. ábra: Nyilvános kulcsból bitcoin cím: egy nyilvános kulcs átalakítása bitcoin címmé

Base58 és Base58Check kódolás

Azért, hogy a hosszú számok tömören, kevesebb szimbólummal legyenek ábrázolhatók, sok számítógérendszer vegyes alfanumerikus ábrázolást használ, ahol a számrendszer alapja 10-nél nagyobb. Például míg a szokásos tízes alapú számrendszer 0-tól 9-ig 10 számjegyet használ, a hexadecimális számrendszer 16-ot, amelyben az A és F közötti betűk jelentik a további hat szimbólumot. Egy hexadecimális formátumban ábrázolt szám rövidebb, mint a neki megfelelő tízes számrendszerbeli szám. Még tömörebb a Base-64 ábrázolás, amely a 26 kisbetűt, a 26 nagybetűt, a 10 számjegyet és két további karaktert, a „+” és a „/” karaktereket használja bináris adatok szövegként, pl. e-levélben történő továbbítására. A Base-64-et

leggyakrabban e-levelek bináris csatolmányainál használják. A Base-58 formátum egy olyan szöveges formátum, melyet a bitcoin és sok más digitális pénz használ. Egyensúlyt teremt a tömör ábrázolás, az olvashatóság, a hiba ellenőrzés és a hiba megelőzés között. A Base-58 a Base-64 egy részhalmaza: a kis- és nagybetűket valamint a számokat használja, de elhagy közülük néhányat, amelyeket gyakran összecserélnek egymással, vagy amelyek nemelyik betűtípus esetén egyformának látszanak. A Base-58 olyan Base-64, melyből hiányzik a 0 (a nulla szám), az O (a nagy O betű), az l (a kis L), az I (a nagy i), valamint a „+” és „/”. Vagy egyszerűbben, a Base-58 a kis- és nagybetűk valamint a számok halmaza, melyből hiányzik az előbb említett négy karakter (0, O, l, I).

4-1. példa: A bitcoin Base-58 ábécéje

```
123456789ABCDEFGHIJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

A Base58Check olyan Base-58 kódolási formátum, amely az elírások és továbbítási hibák elleni védelemként beépített hiba ellenőrző kóddal rendelkezik. Az ellenőrző összeg további négy bájt, amely a kódolt adat végén áll. Az ellenőrző összeg a kódolt adat tömörítéséből származik, emiatt gépelési hibák felfedésére és megelőzésére használható. A dekódoló szoftver egy Base58Check kód esetén kiszámítja az adat ellenőrző összegét, és összehasonlítja a kódban lévő ellenőrző összeggel. Ha a kettő nem egyezik meg, akkor ez azt mutatja, hogy hiba van, és a Base58Check adat érvénytelen. Pl. ezen a módon megelőzhető, hogy egy elgépelt bitcoin címet a pénztárca alkalmazás érvényes címként fogadjon el. Az ellenőrzés hiányában egy gépelési hiba a pénz elvesztéséhez vezetne.

Egy tetszőleges adat (szám) Base58Check formátumba történő átalakítása úgy történik, hogy az adathoz egy előtagot adunk hozzá, az úgynvezett „verzió bájt”-ot, ami a kódolt adat adattípusának egyszerű azonosítására szolgál. Például a bitcoin címek esetében ez az előtag nulla (0x00 hexadecimálisan), míg a titkos kulcsok esetében 128 (0x80 hexadecimálisan). A leggyakoribb előtagokat a 4-1. táblázat mutatja.

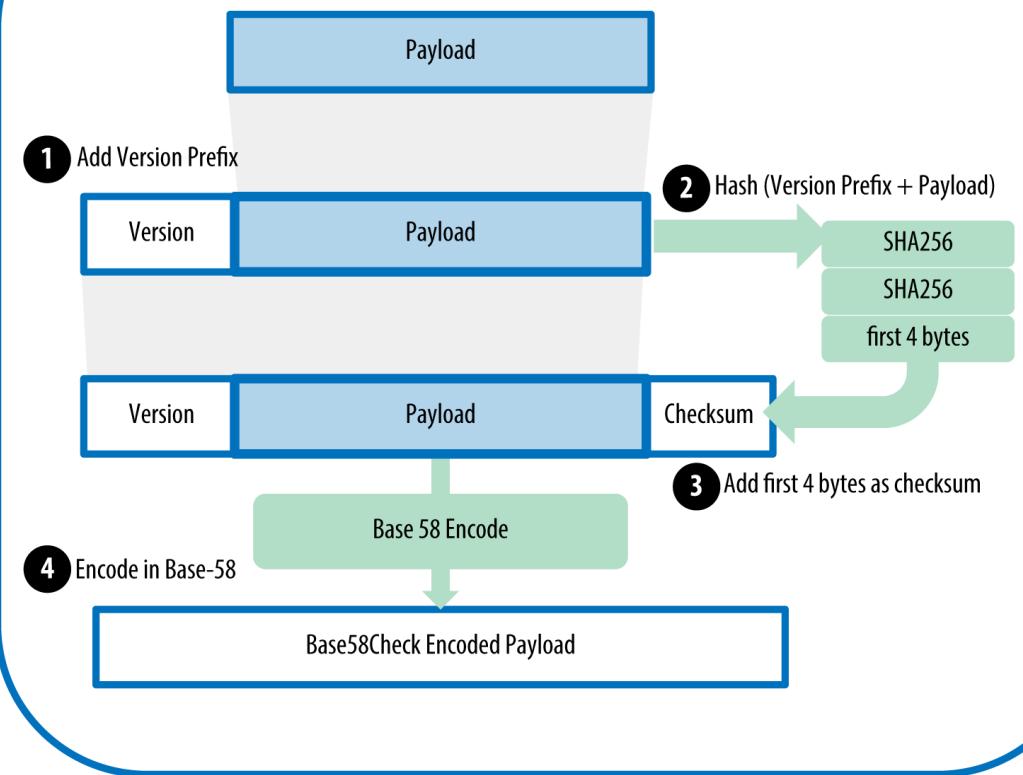
Ezután kiszámítjuk a „kettős-SHA” ellenőrző összeget, vagyis az SHA256 hash algoritmust az előző eredményen kétszer alkalmazzuk:

```
checksum = SHA256(SHA256(prefix+data))
```

Az eredményként kapott 32 bájtos hashból (a hash hashéből) csak az elő négy bájtot használjuk. Ez a négy bájt szolgál hibaellenőrző kódként vagy ellenőrző összegként. Az ellenőrző összeget hozzáadjuk a cím végéhez.

Az eredmény három részből tevődik össze: egy előtagból, az adatból és az ellenőrző összegből. Az eredményt az előzőleg leírt Base58 ábécével kódoljuk. A 4-6. ábra a Base58Check kódolási folyamatát szemlélteti.

Base58Check Encoding



4-6. ábra: Base58Check kódolás: bitcoin adatok egyértelmű kódolása 58-as számrendszerben, verziószámmal és ellenőrző összeggel

A bitcoin esetén a felhasználó számára megjelenített legtöbb adat Base58Check kódolású, mert így az adatok tömörek, könnyen olvashatók és a hibák szempontjából könnyen ellenőrizhetők. A Base58Check kódolásban használt verzió előtag lehetővé teszi, hogy egymástól könnyen megkülönböztethető formátumokat hozzunk létre. Az előtag Base-58-ban kódolva a Base58Check kódolt formátum egy adott karaktere lesz, ami az emberek számára is könnyen felismerhetővé teszi a különféle adattípusokat. Ez különbözteti meg például az „1”-essel kezdődő Base58Check formátumban kódolt bitcoin címet a „5”-sel kezdődő WIF formátumú titkos kulcsról. Néhány minta előtag és az eredményként kapott Base-58 karakter a 4-1. táblázatban látható.

4-1. táblázat: Base58Check verzió előtagok és a kódolt eredmények

Típus	Verzió előtag (hexa)	A Base58 eredmény előtagja
Bitcoin cím	0x00	1
Fliszéts-Script-Hashnek cím	0x05	3
Bitcoin Testnet cím	0x6F	m vagy n
WIF titkos kulcs	0x80	5, K vagy L
BIP38 kódolt titkos kulcs	0x0142	6P
BIP32 kiterjesztett nyilvános kulcs	0x0488B21E	xpub

Tekintsük át a bitcoin cím előállítás teljes folyamatát, a titkos kulcstól a nyilvános kulcsra keresztül a kettősen hashelt címig, és végül a Base58Check kódolásig. A 4-2. példában látható C++ kód lépésről lépésre a teljes folyamatot bemutatja, a privát kulcstól a Base58Check kódolású bitcoin címig. A példa az „[Alternatív kliensek, könyvtárak és eszközök](#)” részben bevezetett libbitcoin könyvtár segédfüggvényeit használja.

4-2. példa: A titkos kulcsból egy Base58Check kódolású bitcoin cím létrehozása

```
$ cat addr.cpp
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Private secret key.
    bc::ec_secret secret;
    bool success = bc::decode_base16(secret,
"038109007313a5807b2ecc082c8c3fbb988a973cacf1a7df9ce725c31b14776");
    assert(success);
    // Get public key.
    bc::ec_point public_key = bc::secret_to_public_key(secret);
    std::cout << "Public key: " << bc::encode_hex(public_key) << std::endl;

    // Create Bitcoin address.
    // Normally you can use:
    //   bc::payment_address payaddr;
    //   bc::set_public_key(payaddr, public_key);
    //   const std::string address = payaddr.encoded();

    // Compute hash of public key for P2PKH address.
    const bc::short_hash hash = bc::bitcoin_short_hash(public_key);

    bc::data_chunk unencoded_address;
    // Reserve 25 bytes
    // [ version:1 ]
    // [ hash:20 ]
    // [ checksum:4 ]
    unencoded_address.reserve(25);
    // Version byte, 0 is normal BTC address (P2PKH).
    unencoded_address.push_back(0);
    // Hash data
    bc::extend_data(unencoded_address, hash);
    // Checksum is computed by hashing data, and adding 4 bytes from hash.
    bc::append_checksum(unencoded_address);
    // Finally we must encode the result in Bitcoin's base58 encoding
    assert(unencoded_address.size() == 25);
    const std::string address = bc::encode_base58(unencoded_address);

    std::cout << "Address: " << address << std::endl;
    return 0;
}
```

A kód egy előre definiált titkos kulcsot használ, emiatt minden egyes futáskor ugyanazt a címet hozza létre, amint azt 4-3. példa mutatja.

4-3. példa: A bitcoin címet előállító mintapélda lefordítása és futtatása

```
$ # Az addr.cpp kód lefordítása
$ g++ -o addr addr.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Az addr végrehajtható program futtatása
$ ./addr
Nyilvános kulcs:
0202a406624211f2abbdc68da3df929f938c3399dd79fac1b51b0e4ad1d26a47aa
Cím: 1PRTaJesdNovgne6EhcaulfpEdX7913CK
```

Kulcs formátumok

Mind a titkos, mind a nyilvános kulcs számos különböző formátumban ábrázolható. A különféle ábrázolási módok ugyanazt a számot ábrázolják, még ha különbözőnek látszanak is. Ezeket a formátumok főként arra használatosak, hogy megkönnyítsék a kulcsok leírását és megadását, és védjenek a hibák ellen.

Titkos kulcs formátumok

Egy titkos kulcs számos különböző formátumban ábrázolható. Ezek mindegyike ugyanannak a 256-bites számnak felel meg. A 4-2. táblázatban a titkos kulcsok ábrázolására szolgáló három leggyakrabban használt formátum látható.

4-2. táblázat: Egy titkos kulcs ábrázolási módjai (kódolási formátumok)

Típus	Előtag	Leírás
Hexa	Nincs	64 hexadecimális számjegy
WIF	5	Base58Check kódolás: Base58, verzió előtaggal (128) és egy 32 bites ellenőrző összeggel
tömörített WIF	K vagy L	Mint előbb, de a kódolás előtt a 0x01 utótag hozzáfűzése

A 4-3. táblázat ebben a három formátumban tartalmazza ugyanazt a privát kulcsot.

4-3. táblázat: Példa: Ugyanaz a kulcs, különböző formátumok

Formátum	Titkos kulcs
Hexa	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aed
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyfsYB1Jcn
tömörített WIF	KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Az összes fenti alak ugyanazt a számot, ugyanazt a titkos kulcsot ábrázolja. Ezek különbözőnek látszanak ugyan, de bármelyik formátum könnyen átalakítható bármelyik másik formátumra.

A Bitcoin Explorer `wif-to-ec` parancsával tudjuk megmutatni, hogy minden WIF kulcs ugyanannak a titkos kulcsnak felel meg:

```
$ bx wif-to-ec 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn  
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
```

```
$ bx wif-to-ec KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ  
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
```

A Base58Check formátum dekódolása

A Bitcoin Explorer parancsaival (lásd „[A Libbitcoin és a Bitcoin Explorer](#)”) könnyen tudunk bitcoin kulcsokat, címeket és tranzakciókat kezelő shell scripteket és „pipe”-okat írni. A Bitcoin Explorer-rel a következőképpen lehet a parancssorban dekódolni a Base58Check formátumot:

A `base58check-decode` parancsot használjuk a tömörítetlen kulcs dekódolására:

```
$ bx base58check-decode 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn  
wrapper  
{  
    checksum 4286807748  
    payload  
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd  
    version 128  
}
```

Az eredmény a kulcsot (hasznos tartalom, payload), a Pénztárca Import Formátum (Wallet Import Format, WIF) előtagját (128) és az ellenőrző összeget tartalmazza.

Figyelje meg, hogy a tömörített kulcs „hasznos tartalmához” a 01 utótag lett hozzáfűzve, ami azt jelzi, hogy tömörített kulcsot szeretnénk előállítani.

```
$ bx base58check-decode  
KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ  
wrapper  
{  
    checksum 2339607926  
    payload  
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01  
    version 128  
}
```

Hexadecimális formátum átalakítása Base58Check formátumba

Ha hexadecimális formátumból Base58Check formátumba szeretnénk átalakítást végezni (az előző parancs ellentettje), akkor a Bitcoin Explorer `base58check-encode` parancsát használhatjuk (lásd „[A Libbitcoin és a Bitcoin Explorer](#)”). A hexadecimális titkos kulcs után a Wallet Import Format (WIF) előtagot, a 128-at kell megadni:

```
$ bx base58check-encode  
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd --version  
128  
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

Tömörített hexadecimális kulcs kódolása Base58Check formátumba

Ha „tömörített” titkos kulcsként (lásd „[Tömörített titkos kulcsok](#)”) szeretnénk a kulcsot Base58Check kódolással előállítani, akkor hozzáadjuk a 01 utótagot a hexa kulcschoz, majd a fentiekhez hasonlóan elvégezzük a kódolást:

```
$ bx base58check-encode  
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01 --  
version 128  
KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

Az eredményként kapott WIF tömörített formátum „K”-val kezdődik, ami azt jelzi, hogy a titkos kulcsnak egy „01” utótagja van, és csak tömörített nyilvános kulcsok hozhatók létre belőle (lásd „[Tömörített nyilvános kulcsok](#)”).

Nyilvános kulcs formátumok

A nyilvános kulcsok szintén többféle formátumban ábrázolhatók, a legfontosabbak a *tömörített* és a *nem tömörített* nyilvános kulcsok.

Mint azt előzőleg láttuk, a nyilvános kulcs az elliptikus görbe egy pontja, amely egy (x , y) koordinátpárból áll. Általában a 04 előtaggal ábrázolják, melyet két 256-bites szám követ, az egyik a pont x -koordinátája, a másik az y -koordinátája. A 04 előtag különbözteti meg a nem tömörített nyilvános kulcsokat a tömörített nyilvános kulcsoktól, melyek 02-vel vagy 03-mal kezdődnek.

Alább a fenti titkos kulcsból előállított nyilvános kulcs x és y koordinátája látható.

```
x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A  
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Ugyanez a nyilvános kulcs egy 520-bites számként (130 hexa számjegyként), a 04 előtaggal, melyet az x és az y koordináta követ:

```
K =  
04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A07CF33DA1  
8BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

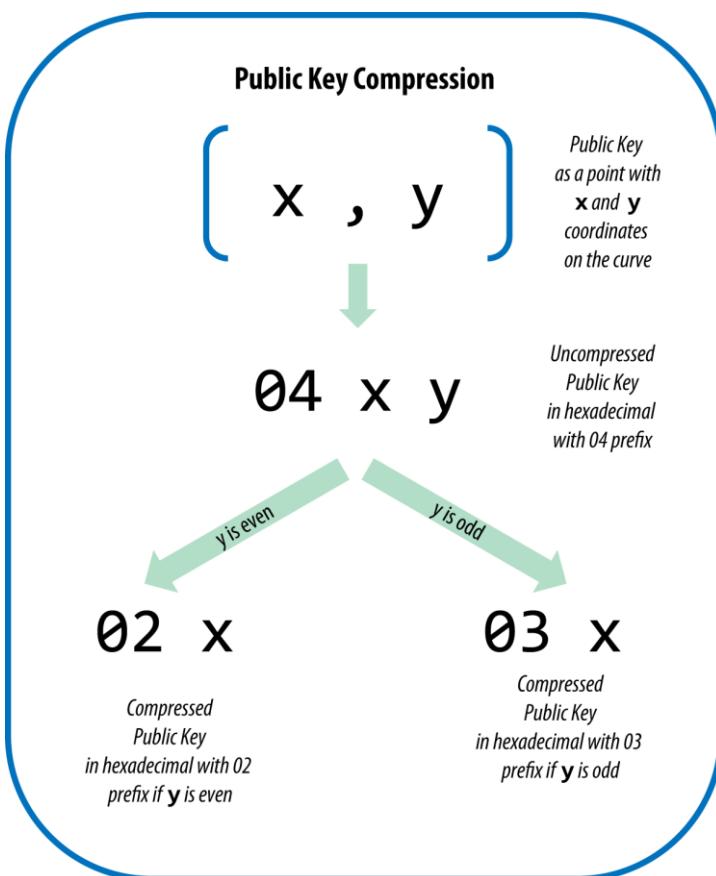
Tömörített nyilvános kulcsok

A tömörített nyilvános kulcsokat azért vezették be a bitcoinban, hogy csökkentsék a tranzakciók méretét és diszk helyet takarítsanak meg a teljes bitcoin blokklánc adatbázist tároló csomópontokon. A legtöbb tranzakciótban szerepel a nyilvános kulcs, amely a tulajdonos személyazonosságának tanúsítására és a bitcoin elköltésére szolgál. Mindegyik nyilvános kulcs 520 bit hosszú (előtag + x + y), ami összeszorozva a blokkban lévő több száz tranzakcióval, vagy a napi több tízezer tranzakcióval jelentős adatmennyiséget tesz ki a blokkláncon.

Amint azt a „[Nyilvános kulcsok](#)” részben láttuk, a nyilvános kulcs egy (x , y) pont az elliptikus görbén. Mivel a görbe egy matematikai függvénynek felel meg, a görbén lévő pont a görbe egyenletének egy megoldását jelenti. Ezért ha ismerjük az x -koordinátát, akkor az y -

koordinátát az $y^2 \bmod p = (x^3 + 7) \bmod p$ egyenlet megoldásával számíthatjuk ki. Ez lehetővé teszi, hogy a nyilvános kulcsban csak az x -koordinátáját tároljuk, és elhagyhassuk az y -koordinátát. Ily módon 256 bittel csökkenthető a tároláshoz szükséges hely. Ezzel majdnem 50%-kal csökken minden tranzakció mérete, ami idővel nagyon nagy helymegtakarításhoz vezet.

Míg a nem tömörített nyilvános kulcsoknak 04 az előtagja, a tömörített kulcsok 02-vel vagy 03-mal kezdődnek. Vizsgáljuk meg, miért van két lehetséges előtag! Mivel az egyenlet bal oldalán y^2 áll, az y megoldás pozitív vagy negatív lehet. Képiled ez azt jelenti, hogy az y -koordináta az x -tengely felett vagy az x -tengely alatt lehet. Amint azt az 4-2. ábrán az elliptikus görbe ábrázolásán láthatjuk, a görbe szimmetrikus, ami azt jelenti, hogy az x -tengelyre tükrös. Emiatt, ha el is hagyhatjuk az y koordinátát, az y előjelét (pozitív vagy negatív) tárolnunk kell, más szóval, tudnunk kell, hogy az x -tengely felett vagy alatt volt-e, mivel minden két lehetséghez egy különböző pont és egy különböző nyilvános kulcs tartozik. Ha az elliptikus görbét a p -rendű véges mezőn számítjuk ki, az y koordináta páros vagy páratlan lehet, ami megfelel a fenti pozitív vagy negatív előjelnek. Ezért aztán ha szeretnénk megkülönböztetni az y lehetséges értékeit, akkor a tömörített nyilvános kulcsot 02 előtaggal tároljuk, ha az y páros, és 03-mal, ha páratlan, ami lehetővé teszi, hogy egy program az x -koordinátából helyesen meg tudja állapítani az y -koordináta értékét, és a tömörített nyilvános kulcsból a pont minden koordinátáját előállítsa. A nyilvános kulcs tömörítését a 4-7. ábra szemlélteti.



4-7. ábra: A nyilvános kulcs tömörítése

Íme, ugyanaz a nyilvános kulcs, melyet előzőleg láttunk, tömörített nyilvános kulcsként, 264 biten (66 hexa számjeggyel) tárolva. A 03 előtag azt jelzi, hogy az y koordináta páratlan:

K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A

Ez a tömörített nyilvános kulcs ugyanannak a titkos kulcsnak felel meg, ami azt jelenti, hogy ugyanabból a titkos kulcsból lett előállítva. Mégis különbözőnek látszik a nem tömörített nyilvános kulcstól. Még fontosabb, hogy ha ezt a tömörített nyilvános kulesot a kétszeres hash függvényvel ([RIPEMD160 \(SHA256 \(K\)\)](#)) bitcoin címmé alakítjuk át, akkor egy másik bitcoin címet kapunk. Ez zavaró lehet, mert azt jelenti, hogy ugyanabból a titkos kulcsból két *különböző* nyilvános kulcs állítható elő, mely két különböző formátumban ábrázolható (tömörítve és nem tömörítve), ami két különböző bitcoin címet eredményez. Ugyanakkor a titkos kulcs minden bitcoin cím esetén azonos.

A tömörített nyilvános kulcsok lassanként alapértelmezettek lesznek a különféle bitcoin klienseken belül, ami jelentős hatással van a tranzakciók méretének csökkentésére, és emiatt a blokkláncra. De még nem mindegyik kliens támogatja a tömörített nyilvános kulcsokat. Az újabb klienseknek, melyek támogatják a tömörített nyilvános kulcsokat, számolniuk kell a tömörített nyilvános kulcsokat nem támogató, régebbi kliensekből származó tranzakciókkal. Ez különösen fontos akkor, ha egy pénztárca alkalmazás titkos kulcsokat importál egy másik pénztárca alkalmazásból, mert az új pénztárcának végig kell pásztáznia a blokkláncot, ha szeretné megtalálni az importált kulcsokhoz tartozó tranzakciókat. Melyik bitcoin címet kell a bitcoin pénztárcának végigpásztáznia? A nem tömörített nyilvános kulcs által előállított bitcoin címet, vagy a tömörített nyilvános kulcsokhoz tartozó bitcoin címet? Mindkettő érvényes bitcoin cím, és minden két külön címről van szó!

A kérdés megoldása érdekében a titkos kulcsok pénztárcából történő kiexportálásakor a titkos kulcsokat ábrázoló WIF formátum (Wallet Import Format, pénztárca import formátum) az újabb pénztárcák esetében eltérő módon lett megvalósítva, hogy azt is jelezze, ha a titkos kulcsok *tömörített* nyilvános kulcsok előállítására szolgálnak, és ennek megfelelően tömörített bitcoin címek tartoznak hozzájuk. Ez lehetővé teszi, hogy az importálást végző pénztárca különbséget tudjon tenni a régebbi vagy újabb pénztárcákból származó titkos kulcsok között, és a blokkláncon azokat a tranzakciókat keresse meg, melyek a megfelelő nem tömörített vagy tömörített nyilvános kulcsokhoz tartozó bitcoin címeknek felelnek meg. Nézzük meg részletesebben, hogyan megy minden végbe.

Tömörített titkos kulcsok

A „tömörített titkos kulcs” elnevezés elég félrevezető, mert a titkos kulcs kiexportálása WIF-tömörített titkos kulcsként történik, és valójában egy bájtal *hosszabb*, mint a „tömörítetlen” titkos kulcs. Ennek az az oka, hogy 01 utótaggal végződik, ami azt jelzi, hogy egy újabb, modern pénztárcából származik, és csak tömörített nyilvános kulcsok előállítására szabad használni. A titkos kulcsok nincsenek tömörítve és nem tömöríthetők. A „tömörített titkos kulcs” kifejezés valójában azt jelenti, hogy „olyan titkos kulcs, melyből tömörített nyilvános kulcsot kell előállítani”, míg a „nem tömörített titkos kulcs” azt jelenti, hogy „olyan titkos kulcs, melyből nem tömörített nyilvános kulcsot kell előállítani”. Az export formátumra „WIF-tömörített” vagy „WIF” formátumként érdemes hivatkozni, és a titkos kulcsnál a további félreértesek elkerülése érdekében el kell felejteni a „tömörítés” szót.

Megjegyezzük, hogy a kétféle formátum *nem* cserélhető fel egymással. Egy modern pénztárcában, amely képes a tömörített nyilvános kulcsok kezelésére, a titkos kulcsok minden WIF-tömörített alakban lesznek kiexportálva (K/L előtag). Ha a pénztárca régebbi, és nem használja a tömörített nyilvános kulcsokat, a titkos kulcs minden WIF formátumban lesz kiexportálva (5 előtag). A cél az, hogy jelezzen a titkos kulcsokat beimportáló pénztárca számára, hogy tömörített vagy tömörítetlen nyilvános kulcsokat és címeket kell-e keresnie a blokkláncban.

Ha a bitcoin pénztárca képes a tömörített nyilvános kulcsok kezelésére, akkor az összes tranzakcióban ezeket fogja használni. A pénztárcában lévő titkos kulcsokból levezethetők a görbén lévő nyilvános pontok, majd megtörténik ezek tömörítése. A pénztárca a tömörített nyilvános kulcsokat fogja bitcoin címek előállítására használni, és ezek szerepelnek majd a tranzakciókban. Ha titkos kulcsokat exportálunk ki egy új pénztárcából, amely támogatja a tömörített nyilvános kulcsokat, akkor a WIF formátum úgy módosul, hogy a titkos kulcs egy 1 bájtos utótaggal (01) egészül ki. Ennek a Base58Check kódolásával kapott titkos kulcsot nevezzük „tömörített WIF”-nek, és ez a „K” vagy az „L” betűvel kezdődik, ellentétben a régebbi pénztárcákból származó, WIF kódolt (nem tömörített) kulcsokkal, melyek „5”-tel kezdődnek.

A 4-4. táblázat ugyanazt a kulcsot mutatja, WIF és WIF-tömörített formátumban.

4-4. táblázat: Példa: Ugyanaz a kulcs, különböző formátumok

Formátum	Titkos kulcs
Hexa	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Tömörített hexa	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD01
tömörített WIF	KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ



A „tömörített titkos kulcs” teljesen helytelen elnevezés! A titkos kules nincs tömörítve. A WIF-tömörített formátum azt jelenti, hogy a titkos kulcsból csak tömörített nyilvános kulcsot, és az ehhez tartozó bitcoin címet szabad előállítani. A „WIF-tömörített” titkos kulcs egy bájttal hosszabb, mert a 01 utótaggal rendelkezik, amely megkülönbözteti a „tömörítetlen” titkos kulcstól.

Kulcsok és címek kezelése Pythonban

A legátfogóbb Pythonban megírt bitcoin könyvtár Vitalik Buterin [pybitcointools](#) könyvtára. A 4-4. példában a „bitcoin”-ként beimportált pybitcointools könyvtárral fogjunk különféle formátumú kulcsokat és címeket előállítani:

4-4. példa: Kulcs és cím előállítás és formattálás a pybitcointools könyvtárral

```
$ cat key-to-address-ecc-example.py
import bitcoin

# Generate a random private key
valid_private_key = False
while not valid_private_key:
    private_key = bitcoin.random_key()
    decoded_private_key = bitcoin.decode_privkey(private_key, 'hex')
    valid_private_key = 0 < decoded_private_key < bitcoin.N

print "Private Key (hex) is: ", private_key
print "Private Key (decimal) is: ", decoded_private_key

# Convert private key to WIF format
wif_encoded_private_key = bitcoin.encode_privkey(decoded_private_key,
'wif')
print "Private Key (WIF) is: ", wif_encoded_private_key

# Add suffix "01" to indicate a compressed private key
compressed_private_key = private_key + '01'
print "Private Key Compressed (hex) is: ", compressed_private_key

# Generate a WIF format from the compressed private key (WIF-compressed)
wif_compressed_private_key = bitcoin.encode_privkey(
    bitcoin.decode_privkey(compressed_private_key, 'hex'), 'wif')
print "Private Key (WIF-Compressed) is: ", wif_compressed_private_key

# Multiply the EC generator point G with the private key to get a public
# key point
public_key = bitcoin.fast_multiply(bitcoin.G, decoded_private_key)
print "Public Key (x,y) coordinates is:", public_key

# Encode as hex, prefix 04
hex_encoded_public_key = bitcoin.encode_pubkey(public_key, 'hex')
print "Public Key (hex) is:", hex_encoded_public_key

# Compress public key, adjust prefix depending on whether y is even or odd
(public_key_x, public_key_y) = public_key
if (public_key_y % 2) == 0:
    compressed_prefix = '02'
else:
    compressed_prefix = '03'
hex_compressed_public_key = compressed_prefix +
    bitcoin.encode(public_key_x, 16)
print "Compressed Public Key (hex) is:", hex_compressed_public_key

# Generate bitcoin address from public key
print "Bitcoin Address (b58check) is:",
bitcoin.pubkey_to_address(public_key)

# Generate compressed bitcoin address from compressed public key
print "Compressed Bitcoin Address (b58check) is:", \
    bitcoin.pubkey_to_address(hex_compressed_public_key)
```

A 4-4.1 példa a kód futtatásakor kapott kimenet mutatja:

4-4.1 példa: A key-to-address-ecc-example.py futtatása

```
$ python key-to-address-ecc-example.py
Private Key (hex) is:
3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa6
Private Key (decimal) is:

265632300484379575922325538266636964406067566859201174768322996732930137688
70
Private Key (WIF) is:
5JG9hT3beGTJuUAmCQEEmNaxAuMacCTfXuw1R3FCXig23RQHMr4K
Private Key Compressed (hex) is:
3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa601
Private Key (WIF-Compressed) is:
KyBsPXXtUVd82av65KZkrGrWi5qLMah5SdNq6uftawDbgKa2wv6S
Public Key (x,y) coordinates is:

(41637322786646325214887832269588396900663353932545912953362782457239403430
124L,
163889351287812384055267104667247415937610851208643314490666586224003393621
66L)
Public Key (hex) is:

045c0de3b9c8ab18dd04e3511243ec2952002dbfad864b9628910169d9b9b00ec243bccefdd
4347074d44bd7356d6a53c495737dd96295e2a9374bf5f02ebfc176
Compressed Public Key (hex) is:
025c0de3b9c8ab18dd04e3511243ec2952002dbfad864b9628910169d9b9b00ec
Bitcoin Address (b58check) is:
1thMirt546nngXqyPEz532S8fLwbozud8
Compressed Bitcoin Address (b58check) is:
14cxpo3MBCYYWCqF74SWTdcmxipnGUsPw3
```

A 4-5. példa egy másik példa, amely a Python ECDSA könyvtárat használja az elliptikus görbén történő számításokhoz, és nem használ semmilyen egyéb speciális bitcoin könyvtárat.

4-5. példa: A bitcoin kulcsoknál használt, elliptikus görbén végezett számítások szemléltetése

```

def random_secret():
    convert_to_int = lambda array: int("".join(array).encode("hex")), 16

    # Collect 256 bits of random data from the OS's cryptographically
    # secure random generator
    byte_array = os.urandom(32)

    return convert_to_int(byte_array)

def get_point_pubkey(point):
    if point.y() & 1:
        key = '03' + '%064x' % point.x()
    else:
        key = '02' + '%064x' % point.x()
    return key.decode('hex')

def get_point_pubkey_uncompressed(point):
    key = '04' +
        '%064x' % point.x() +
        '%064x' % point.y()
    return key.decode('hex')

# Generate a new private key.
secret = random_secret()
print "Secret: ", secret

# Get the public key point.
point = secret * generator
print "EC point:", point

print "BTC public key:", get_point_pubkey(point).encode("hex")

# Given the point (x, y) we can create the object using:
point1 = ecdsa.ellipticcurve.Point(curve, point.x(), point.y(), ec_order)
assert point1 == point

```

A 4-6. példa a script futtatásakor kapott kimenetet mutatja.



A fenti példa az `os.urandom` véletlenszám generátort használja, amely kriptográfiailag biztonságos véletlenszám generátor (cryptographically secure random number generator (CSRNG)), amely a scriptet futtató operációs rendszerből származik. Az UNIX-szerű operációs rendszerek, például a Linux esetén a script a `/dev/urandom` forrást használja, a Windows esetén pedig a `CryptGenRandom()` függvényt hívja. Ha a script nem talál megfelelő véletlen forrást, akkor a `NotImplementedError` hibajelzést adja. Az itt használt véletlenszám generátor csupán szemléltetésre szolgál, és *NEM* alkalmas éles bitcoin kulcsok előállítására, mivel nem rendelkezik elégsges biztonsággal.

4-6. példa: A Python ECDSA könyvtár installálása és az ec_math.py script futtatása

```
$ # Install Python PIP package manager
$ sudo apt-get install python-pip
$ # Install the Python ECDSA library
$ sudo pip install ecdsa
$ # Run the script
$ python ec-math.py
Secret:
380908350159543588624811326288874439059062049959123782780601687035806602940
00
EC point:
(70048853531867179489857750497606966272382583471322935454624595540007269312
627,
105262206478686743191060800263479589329920209527285803935736021686045542353
380)
BTC public key:
029ade3effb0a67d5c8609850d797366af428f4a0d5194cb221d807770a1522873
```

Pénztárcák

A pénztárcák a titkos kulcsok tárolására szolgálnak. Általában strukturált adatállományokkal vagy egyszerű adatbázisokkal vannak megvalósítva. A kulcs előállításának egy másik módszere a *determinisztikus kulcs előállítás*. Ennél minden egyik új titkos kulcs egy egyirányú hash függvény használatával, az előző titkos kulcsból áll elő, és egy sorozatot képez. A sorozat újból létrehozásához csak az első kulcsra van szükség (ennek *mag* vagy *mesterkulcs* a neve). Ebben a részben megvizsgáljuk a kulcsgenerálás különféle módszereit, és a köréjük épített pénztárca szerkezeteket.



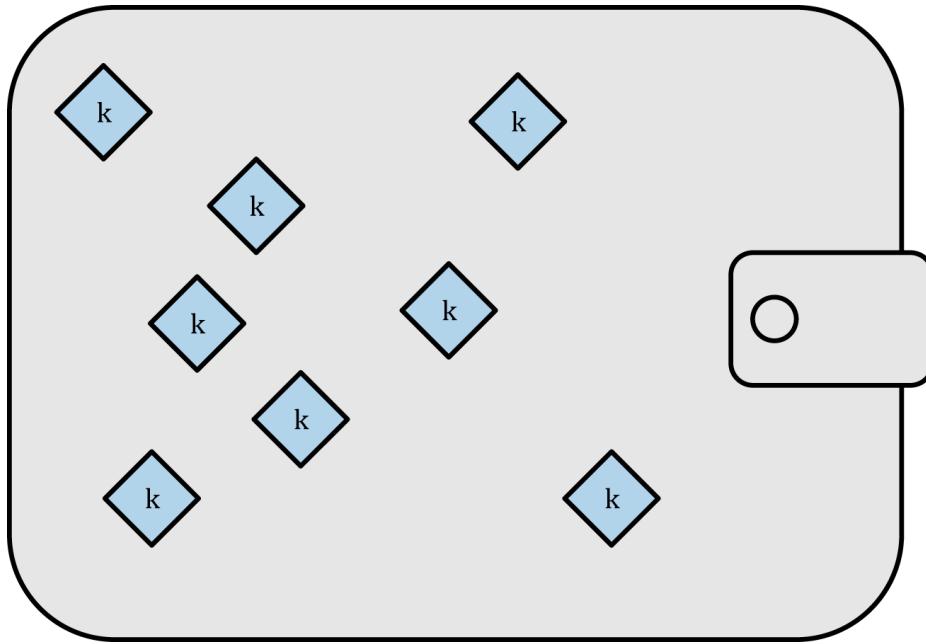
A pénztárcában kulcsok vannak, nem pedig érmék. Mindegyik felhasználónak van egy kulcsokat tartalmazó pénztárcája. A pénztárcák valójában kulcskarikák, melyeken nyilvános/titkos kulcspárok vannak (lásd a „[Titkos és nyilvános kulcsok](#)” részt). A felhasználók a kulcsokkal írják alá a tranzakciókat, így bizonyítva, hogy a birtokukban vannak az aláírt tranzakció kimenetek (az érmék). Az érméket a blokklánc tárolja, tranzakció kimenetek formájában (ezeket gyakran úgy jelölik, hogy *vout* vagy *txout*).

Nem-determinisztikus (véletlen) pénztárcák

Az első bitcoin kliensekben a pénztárca egyszerűen egy halom véletlenszerűen generált titkos kules volt. Az ilyen pénztárcákat *0. típusú, nem determinisztikus pénztárcának* hívjuk. Például a Bitcoin Core kliens az első indításakor előre generál 100 db véletlenszerű titkos kulcsot, és szükség esetén további kulcsokat generál. Mindegyik kulcsot csak egyszer használja. Az ilyen pénztárcát úgy is hívják, hogy „csak egy halom kulcs”. A determinisztikus pénztárcák váltják föl őket, mert nagyon nehézkes a kezelésük, a kulcsok mentése és beimportálása. A véletlenszerűen generált kulcsoknak az a hátránya, hogy ha sok ilyet állítunk elő, akkor mindegyikről másolatot kell készítenünk, ami azt jelenti, hogy a pénztárcát gyakran kell mentenünk. Mindegyik kulcsról biztonsági másolatot kell készítenünk, mert ha a pénztárca hozzáférhetetlenné válik, akkor a kules által kontrollált pénz örökre elvész. Ez közvetlenül ellentmond annak az alapelveknek, hogy a címeket ne használjuk föl újra, vagyis hogy minden egyik bitcoin címet csak egy tranzakcióhoz használunk. A cím újból felhasználása csökkenti a titkosságot, mivel kapcsolatba hozza egymással a tranzakciókat és a címeket. A 0. típusú pénztárca emiatt gyenge választás, különösen akkor, ha szeretnénk elkerülni a címek újrafelhasználását, ami azt jelenti, hogy sok kulcs kezelésére és emiatt gyakori mentésre van szükség. A Bitcoin Core kliensben lévő pénztárca 0. típusú, de ennek a használatát a Bitcoin Core fejlesztők aktívan ellenjavallják. A 4-8. ábra egy nem determinisztikus pénztárcát ábrázol, amely véletlenszerűen generált kulcsok gyűjteménye.

Determinisztikus (magot használó) pénztárcák

A determinisztikus, vagy másképpen „magot használó” pénztárcák olyan pénztárcák, melyekben a titkos kulcsokat egy egyirányú hash függvényteljesítéssel egy közös magból állítják elő. A mag egy véletlenszerűen generált szám, melyből más adatokkal, pl egy indexszámmal vagy „lánc kódossal” kombinálva állítják elő a titkos kulcsokat (lásd „[Hierarchikus determinisztikus pénztárcák \(BIP0032/BIP0044\)](#)”). Egy determinisztikus pénztárca esetén a mag ismeretében az összes származtatott kulcs visszanyerhető, emiatt csupán egyetlen egy biztonsági másolat készítésére van szükség. A mag a pénztárca exportjához vagy importjához is elégéges, ezért a felhasználó összes kulcsa könnyen átköltöztethető egy tetszőleges másik pénztárcába.



4-8. ábra: 0. típusú, nem determinisztikus (véletlen) pénztárca: véletlenszerűen generált kulcsok gyűjteménye

Mnemonikok

A mnemonikok olyan szóláncok, melyek egy determinisztikus pénztárca magját alkotó véletlen számnak felelnek meg. A szólánc elégseges a mag újbóli előállításához, ezáltal a pénztárca és a származtatott kulcsok újbóli létrehozásához. Az olyan pénztárca program, amely mnemonikokat használ, a pénztárca létrehozásakor 12 – 24 szót jelenít meg a felhasználónak. Ezek a szavak jelentik a pénztárca mentését, és segítségükkel egy azonos típusú vagy egy kompatibilis pénztárca programban az összes kulcs visszaállítható. A mnemonikok megkönnyítik a pénztárcák mentését, mert sokkal könnyebben olvashatók és rögzíthetők, mint egy véletlen szám.

A mnemonikokat a Bitcoin Improvement Proposal 39 definiálja (lásd [\[bip0039\]](#)), ami jelenleg még csak „tervezet”. A BIP0039 csak javaslat, nem szabvány. Például az Electrum pénztárca egy BIP0039 előtti másik mnemonik halmazt és egy másik szabványt használ. A Trezor és néhány másik pénztárca BIP0039-et használja, de ez nem kompatibilis az Electrummal.

A BIP0039 a mnemonikok és a mag létrehozását a következőképpen definiálja:

1. Hozzunk létre egy 128 .. 256 bites véletlen sorozatot (entrópiát)
2. Készítsük el a véletlen sorozat ellenőrző összegét oly módon, hogy vesszük az SHA256 hashének első néhány bitjét
3. Adjuk hozzá ezt az ellenőrző összeget a véletlen sorozat végéhez
4. Osszuk a sorozatot 11 bites részekre, melyek egy 2048 szavas, előre definiált szótár indexelésére szolgálnak.
5. Állítsuk elő a 12 – 24 szóból álló mnemonikot.

A 4-5. táblázat a mnemonikok hossza és az entrópia mérete közötti összefüggést szemlélteti

4-5. táblázat: Mnemonikok: Entrópia és szóhossz

Entrópia (bitek)	Ellenőrző összeg (bitek)	Entrópia+Ellenőrző összeg	Szóhossz
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

A mnemonikok a 128 .. 256 bitnek felelnek meg. Ezekből egy PBKDF2 kulcs-kiszélesítő függvény egy hosszabb (512 bites) magot állít elő. Az így kapott magot használja a determinisztikus pénztárca az összes származtatott kúlcslétrehozására.

A 4-6. és a 4-7. táblázatok a mnemonikokra és az általuk előállított magokra mutatnak néhány példát.

4-6. táblázat: 128 bites entrópiájú mnemonik és a belőle kapott mag

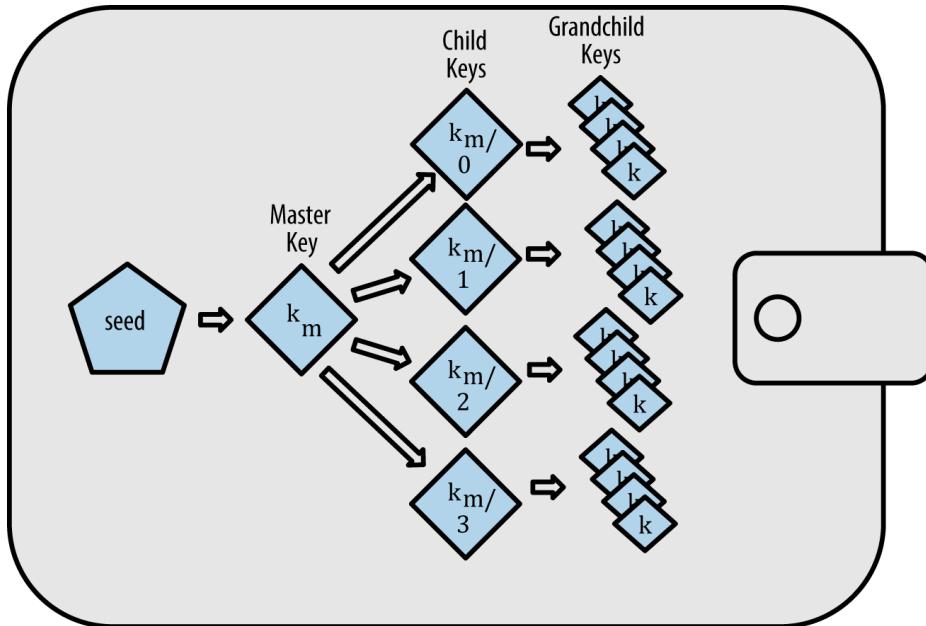
Entrópia (128 bit)	0c1e24e5917779d297e14d45f14e1a1a
Mnemonik (12 szó)	army van defense carry jealous true garbage claim echo media make crunch
Mag (512 bit)	3338a6d2ee71c7f28eb5b882159634cd46a898463e9d2d0980f8e80dfbba5b0fa0 291e5fb88 8a599b44b93187be6ee3ab5fd3ead7dd646341b2cdb8d08d13bf7

4-7. táblázat: 256 bites entrópiájú mnemonik és a belőle kapott mag

Entrópia (256 bit)	2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c
Mnemonik (24 szó)	cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige
Mag (512 bit)	3972e432e99040f75ebe13a660110c3e29d131a2c808c7ee5f1631d0a977fcf473 bee22 fce540af281bf7cdeade0dd2c1c795bd02f1e4049e205a0158906c343

Hierarchikus determinisztikus pénztárcák (BIP0032/BIP0044)

A determinisztikus pénztárcákat azért fejlesztették ki, hogy könnyű legyen egy „magból” sok kulcsot előállítani. A determinisztikus pénztárcák legfejlettebb fajtája a *hierarchikus determinisztikus pénztárca*, azaz a *HD pénztárca*, melyet a BIP0032 szabvány definiál. A hierarchikus determinisztikus pénztárcákban a kulcsok fa szerkezetet alkotnak, ahol egy szülő kulcsból számos gyermek kulcs állítható elő. A gyermek kulcsok mindegyikéből „unoka” kulcsok, és így tovább, a végtelenségig. Ez a fa szerkezet látható a 4-9. ábrán.



4-9. ábra: 2. típusú hierarchikus determinisztikus pénztárca: egyetlen magból kulcsok fája áll elő



Az újonnan kifejlesztett bitcoin pénztárcák HD pénztárcák, melyek megfelelnek a BIP0032 és BIP0044 szabványoknak.

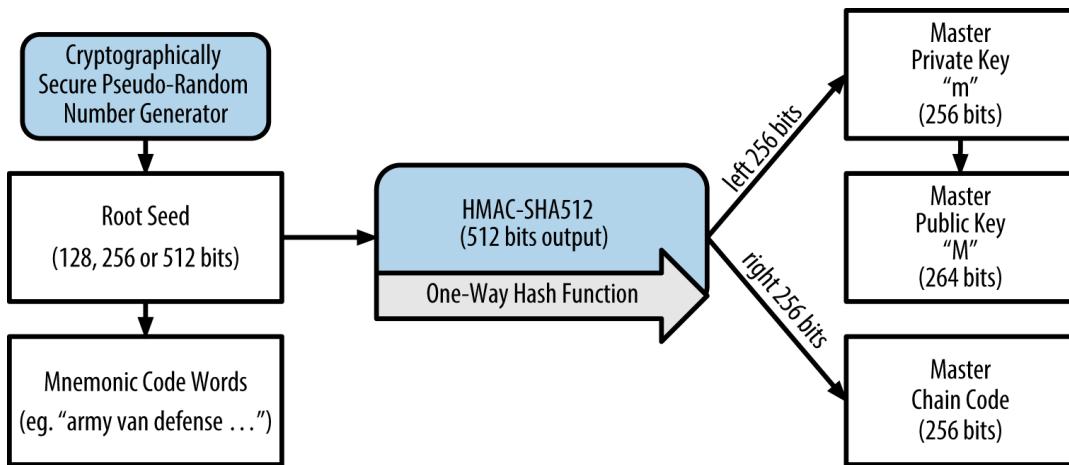
A HD pénztárcáknak két nagy előnyük van a véletlenszerű (nem determinisztikus) kulcsokkal szemben. Az első az, hogy a fa szerkezethez további jelentés rendelhető hozzá, pl. az egyik ágon lévő kulcsok használhatók a bejövő fizetségekhez, míg egy másik a kimenő fizetségekhez tartozó visszajáró pénz kezelésére. A kulcscsoporthoz egy vállalon belül különféle osztályoknak, részlegeknek vagy könyvelési kategóriáknak feleltethetők meg,

A HD pénztárcák másik előnye az, hogy a felhasználó anélkül tud nyilvános kulcsokat létrehozni, hogy ehhez szükség lenne a hozzájuk tartozó titkos kulcsokra. Ez lehetővé teszi, hogy a HD pénztárcákat nem biztonságos szervereken is használhassuk pénz fogadásra, és minden egyes tranzakcióhoz egy saját nyilvános kulcsot hozzunk létre. A nyilvános kulcsot nem kell előre betölteni vagy kiszámítani, ugyanakkor a szerveren nem kell, hogy ott legyen a pénz elköltésére szolgáló titkos kulcs.

HD pénztárca létrehozása egy magból

A HD pénztárcák egyetlen egy *kiinduló magból* állíthatók elő. Ez a mag egy 128, 256 vagy 512 bites véletlenszám. A HD pénztárcában minden más ebből a kiinduló magból, determinisztikusan származik, ami lehetővé teszi, hogy bármelyik másik kompatibilis HD pénztárca programban az egész HD pénztárcát újraépítsük. Ez megkönnyíti a kulcsok ezreit vagy millióit tartalmazó HD pénztárcák exportját és importját, mivel egyszerűen csak a kiinduló magot kell exportálni ill. importálni. A kiinduló magot a könnyű kezelhetőség miatt a leggyakrabban *mnemonikkal* ábrázolják, lásd az előző „[Mnemonikok](#)” részben leírtakat.

Egy HD pénztárca mesterkulcsainak és lánc kódjainak előállítási folyamatát a 4-10. ábra mutatja.



4-10. ábra: Mesterkódok és lánc kód előállítása a kiinduló magból

A kiinduló magot a HMAC-SHA512 algoritmus bemeneteként használják, és az eredményként kapott hasht használják a *titkos mesterkulcs* (*m*) és a *lánckód* előállítására. A nyilvános mesterkulcsot (*M*) a titkos mesterkulcs (*m*) segítségével, hagyományos elliptikus szorzással áll elő: $m * G$, ahogyan azt a fejezet korábbi részében láttuk. A lánckód arra a célra szolgál, hogy entrópiát vigyen a szülő kulcsokból a gyermek kulcsokat előállító függvénybe, amint azt a következő részben látni fogjuk.

Titkos gyermek kulcsok előállítása

A hierarchikus determinisztikus pénztárcák egy *gyermek kulcsok levezetésére szolgáló CKD (child key derivation)* függvényt használnak a leszármaztatott kulcsok szülő kulcsokból történő előállítására.

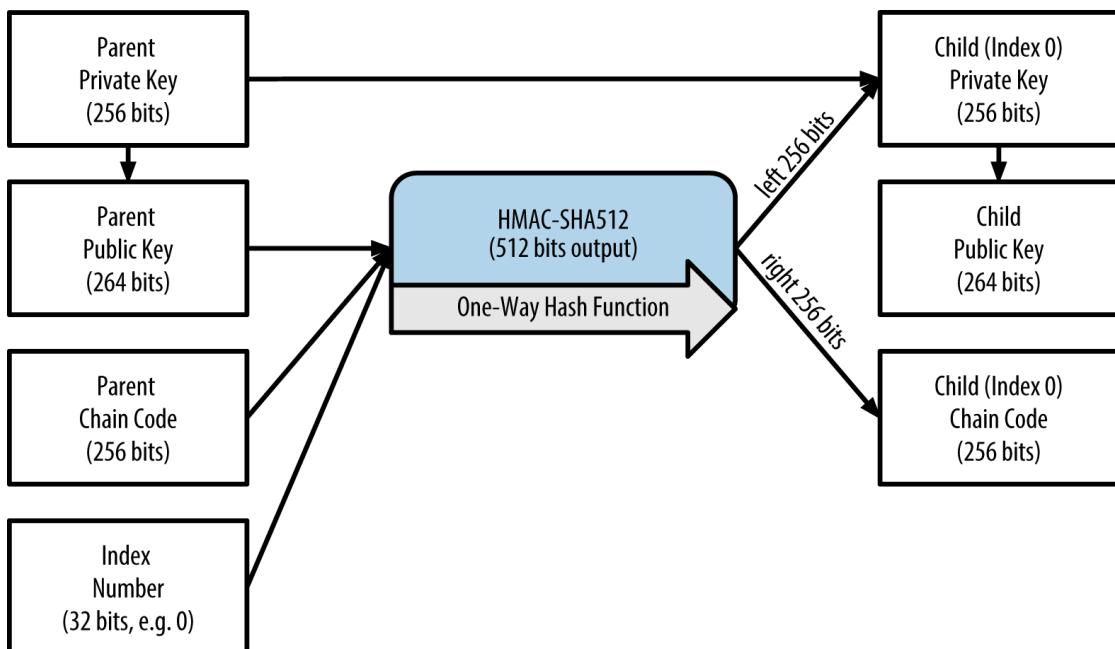
A leszármaztatott kulcsok előállítására szolgáló függvény egy egyirányú hashen alapul, amelyben a következők hashelése történik:

- A szülő titkos vagy nyilvános kulcsa (ECDSA tömörítetlen kulcs)
- Egy lánckódnak nevezett mag (256 bites)
- Egy indexszám (32 bites)

A lánckód arra a célra szolgál, hogy az eljárásba látszólag véletlen adatot vigyen, vagyis hogy önmagában az index ne legyen elégsges a leszármaztatott kulcsok előállítására. Emiatt ha van egy leszármaztatott kulcsunk, akkor ebből csak akkor tudjuk a további leszármazottakat előállítani, ha a lánckóddal is rendelkezünk. A lánckód kezdeti magja (a fa gyökerénél) véletlen adatból származik, míg az egyes további lánckódok a szülő lánckódjából származnak.

A fenti három térel összekapsolása és hashelése a következőképpen történik:

A szülő nyilvános kulcsának, lánckódjának és indexszámának összefűzése után a HMAC-SHA512 algoritmussal egy 512 bites hasht áll elő. A hash értékét két részre vágják. A hash jobb oldali 256 bitje lesz a leszármazott lánckódja. A hash bal oldali 256 bitjét és az indexszámot hozzáadják a szülő titkos kulcsához, és így létrejön a leszármazott titkos kulcsa. A 4-11. ábra mutatja, hogy az index 0-ra állításával hogyan lehet a szülő 0-ik (index szerint első) leszármaztatott kulcsát előállítani.



4-11. ábra: A szülő titkos kulcsának kiterjesztésével a gyermek titkos kulcsának előállítása

Az index megváltoztatása lehetővé teszi a szülő kiterjesztését, és további gyermek kulcsok előállítását, pl. Gyermek 0, Gyermek 1, Gyermek 2 stb. Mindegyik szülő kulcshoz 2 milliárd gyermek kulcs tartozik.

Ha a folyamatot a fában egy szinttel lejjebb megismételjük, akkor minden egyes gyermekből szülő lesz, és saját gyermeket hoz létre, végtelen sok generációban.

A leszármaztatott kulcsok használata

A származtatott titkos kulcsokat nem lehet megkülönböztetni a nem determinisztikus (véletlen) kulcsuktól. Mivel a származtató függvény egyirányú, a származtatott kulcsból a szülő kulcs nem állapítható meg. A származtatott kulccsal a további leszármazottak sem kereshetők meg. Ha van egy n-ik kulcsunk, akkor ennek ismeretében sem az n-1-ik, sem az n+1-ik testvérét sem lehet megtalálni, sőt, a sorozat egyik elemét sem. Csak a szülő kulccsal és a lánckóddal lehet a gyermeket leszármaztatni. A gyermek lánckódja nélkül a gyermek kulcsokból nem lehetséges az unokák vezetése sem. A gyermek titkos kulcsa és a gyermek lánckódja egyaránt szükséges egy új ág megkezdéséhez és az unoka kulcsok leszármaztatásához.

De akkor mire használhatók önmagukban a gyermek titkos kulcsok? Arra, hogy egy nyilvános kulcsot és egy bitcoin címet állítsunk elő velük. Ezt követően pedig arra, hogy a titkos kulcshoz tartozó bitcoin címre küldött tranzakciókat aláírjuk velük, és ily módon elköltök.



A gyermek kulcs, a hozzá tartozó nyilvános kulcs és a bitcoin cím megkülönböztethetetlen a véletlenszerűen előállított kulcsuktól és címektől. Nem látható rajtuk, hogy egy lánc részei, vagy hogy egy HD pénztárca függvény állította őket elő. A létrejöttük után már pontosan olyanok, mintegy „normális” kulcs.

Kiterjesztett kulcsok

Mint azt már korábban láttuk, a kulcs származtatást végző függvényekkel a fa bármelyik szintjén új leszármaztak (gyermekek) állítható elő. Ehhez három bemenetre van szükség: a kulcsra, a lánckódra, és a kívánt leszármaztott indexére. Ezek közül a két legfontosabb a kulcs és a lánckód, és ezeket egy *kiterjesztett kulcsba* szokták összevonni. A „kiterjesztett kulcs” elnevezés „kibővíthető kulcsot” is jelent, mivel az ilyen kulcsokkal gyermekek állítható elő.

A kiterjesztett kulcsok a 256 bites kulcs és a 256 bites lánckód összefűzése révén, egy 512 bites értékkel ábrázolhatók. Kétféle kiterjesztett kulcs van. A kiterjesztett titkos kulcs a titkos kulcs és a lánckód összefűzéséből áll, és a leszármaztak titkos kulcsainak (ezekből pedig a nyilvános kulcsainak) az előállítására használható. A kiterjesztett nyilvános kulcs a nyilvános kulcsból és a lánckódból áll, és a leszármaztak nyilvános kulcsai állíthatók elő vele, amint azt az „[Egy nyilvános kulcs előállítása](#)” részben leírtuk.

A kiterjesztett titkos kulcsot úgy képzelhetjük el, mint a HD pénztárca fa szerkezetének a gyökerét. A gyökér ismeretében a többi ág levezethető. A kiterjesztett titkos kulccsal egy teljes ág előállítható, míg a kiterjesztett nyilvános kulccsal egy teljes ág nyilvános kulcsai.



A kiterjesztett kulcs egy titkos vagy nyilvános kulcsból és egy lánckódból áll. A kiterjesztett kulccsal leszármaztak állíthatók elő, vagyis a fa szerkezetben a saját ága. Egy kiterjesztett kulcs megosztása az egész ághoz hozzáférést biztosít.

A kiterjesztett kulcsokat Base58Check kódolással kódolják, hogy könnyű legyen a különféle BIP0032-kompatibilis pénztárcák közötti exportjuk és importjuk. A Base58Check kódolás a kiterjesztett kulcsok esetén egy különleges verziószámot használ, amelyből az "xprv" vagy "xpub" Base58 karakterek jönnek létre a kódolás során. Ezáltal a kiterjesztett kulcsok könnyen felismerhetők. Mivel egy kiterjesztett kules 512 vagy 513 bites, emiatt sokkal hosszabb, mint a korábban látott Base58Check kódolású stringek.

Íme, egy példa egy Base58Check kódolású kiterjesztett titkos kulcsra:

```
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoOp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9  
CAWrUE9i6GoNMKuga5biW6Hx4tws2six3b9c
```

És itt a neki megfelelő kiterjesztett nyilvános kulcs, szintén Base58Check kódolásban:

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVK  
ZBPLrtJunSDMstweyLXhRgPxdp14sk9tJPW9
```

Gyermekek nyilvános kulcsainak leszármaztatása

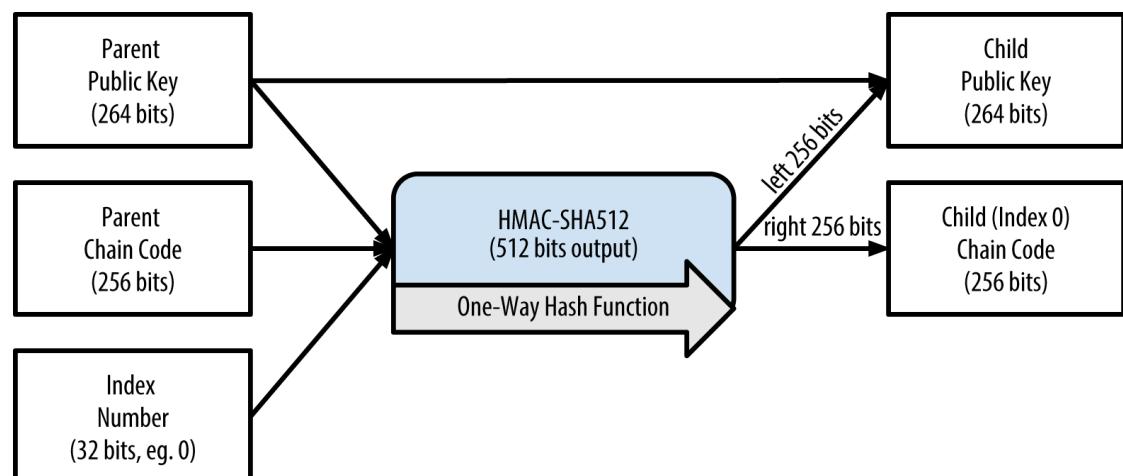
Mint azt előzőleg említettük, a hierarchikus determinisztikus pénztárcák nagyon hasznos jellemzője, hogy a gyermekek nyilvános kulcsai *anélkül* is előállíthatók a szülők nyilvános kulcsaiból, hogy ehhez titkos kulcsokra lenne szükség. Ezért aztán a gyermekek nyilvános kulcsai kétféleképpen is előállíthatók: egyrészt a gyermek titkos kulcsából, másrészt közvetlenül a szülő nyilvános kulcsából.

A kiterjesztett nyilvános kulcsból tehát a HD pénztárca egy ágának az összes *nyilvános kulcsa* (és csak a nyilvános kulcsok) leszármaztathatók.

Ezzel a trükkkel nagyon biztonságos csak-nyilvános-kulcsokat tartalmazó rendszerek hozhatók létre, ahol a szerver alkalmazásban csak a kiterjesztett nyilvános kulcs másolata van meg, és semmilyen titkos kulcsot sem tartalmaz. Az ilyen rendszerekben végtelen sok nyilvános kulcs és bitcoin cím hozható létre, de ezekről a címekről nem lehetséges pénzt költeni. A kiterjesztett titkos kulccsal egy másik, biztonságosabb szerveren a nyilvános kulcsoknak megfelelő összes titkos kulcs levezethető, a tranzakciók aláírhatók és a pénz elkölthető.

Ennek a megoldásnak az egyik gyakori alkalmazása az, hogy a kiterjesztett nyilvános kulcsot egy web szerveren installálják, amely egy e-kereskedelmi alkalmazást szolgál ki. A web szerver a leszármaztató függvénytel révén minden tranzakció számára (pl. egy ügyfél bevásárló kosara számára) új bitcoin cím előállítására. A web szerveren nem tartalmaz egyetlen egy titkos kulcsot sem, mert azt ellophatják. A HD pénztárcák nélkül csak úgy lehetne bitcoin címek ezreit előállítani, hogy a címeket egy másik, biztonságos szerveren állítják elő, majd betöltsék őket az e-kereskedelmi alkalmazásba. Ez sok bonyodalommal járna, és állandó karbantartást igényelne, mert biztosítani kellene, hogy az e-kereskedelmi alkalmazás soha „ne fusson ki” a kulcsokból.

A megoldás egy másik gyakori alkalmazását a hideg tárolók vagy a hardver pénztárcák jelentik. Ebben az esetben a kiterjesztett titkos kulcsot egy papír pénztárca vagy egy hardver pénztárca tárolja, ilyen pl. a Terzor hardver pénztárca, míg a kiterjesztett nyilvános kulcs online hozzáférhető. A felhasználó tetszés szerint tud „fogadó” címeket létrehozni, míg a titkos kulcsok biztonságos módon, offline vannak tárolva. Az összegek elköltéséhez az szükséges, hogy a felhasználó a kiterjesztett titkos kulccsal egy offline bitcoin kliensben vagy egy hardver eszközzel (pl. a Trezorral) aláírja a tranzakciót. A 4-12. ábra szemlélteti, hogyan lehetséges a szülő nyilvános kulcs kiterjesztésével a gyermek kulcsok levezetése.



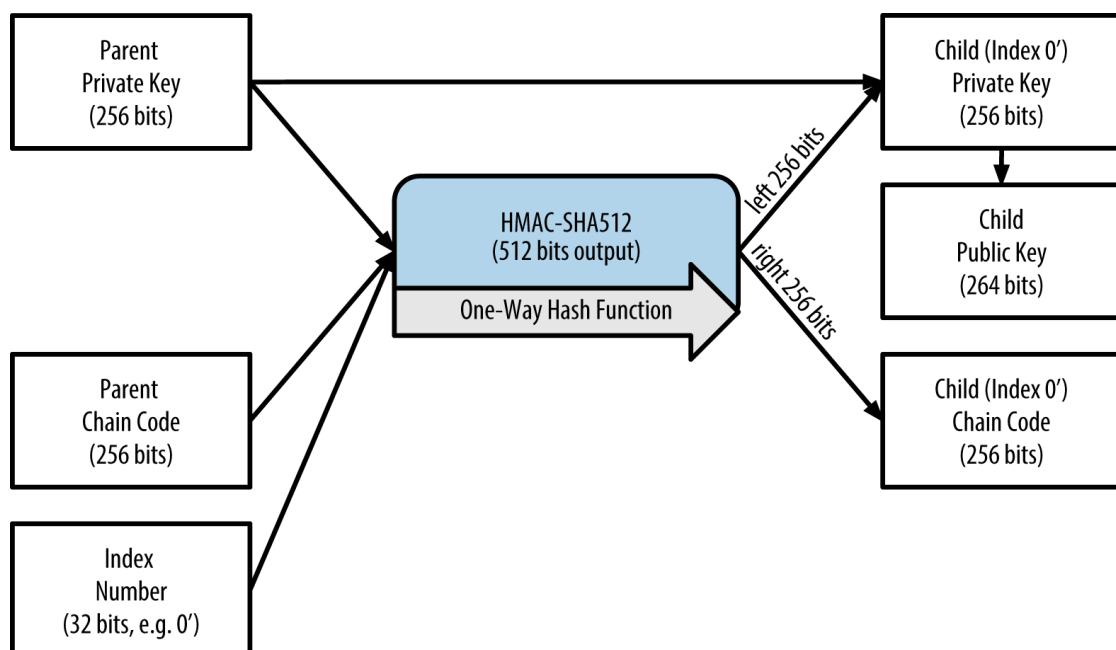
4-12. ábra: A szülő nyilvános kulcs kiterjesztése gyermek kulcsok levezetése céljából

Megerősített gyermek kulcsok előállítása

Az, hogy egy ág összes nyilvános kulcsa a kiterjesztett nyilvános kulcsból vezethető le, nagyon hasznos, de potenciális veszéllyel is jár. A kiterjesztett nyilvános kulcs alapján a gyermekek titkos kulcsa nem állítható elő. Mivel azonban a kiterjesztett nyilvános kulcs tartalmazza a lánckódöt, ezért ha kiszivárog vaz egyik gyermek titkos kulcsa, akkor ebből a

lánckód ismeretében az összes többi gyermek titkos kulcsa is előállítható. Egyetlen egy kiszivárgott titkos kulcs és a szülő lánckódja az összes gyermek titkos kódját felfedi. Ami még ennél is rosszabb, a gyermek titkos kulcsából és a szülő lánckódjából a szülő titkos kulcsa is megállapítható.

Ennek a veszélynek a kivédése érdekében a HD pénztárcák egy alternatív kulcs előállító függvényt használnak, az ún. *megerősített kulcs előállítást*, amely „megszakítja” a szülő nyilvános kulcsa és a gyermek lánckódja közötti összefüggést. A megerősített kulcs előállító függvény a szülő nyilvános kulcsa helyett a szülő titkos kulcsát használja a gyermek lánckódjának a levezetésére. Ez „tűzfalat” hoz létre a szülő/gyermek sorozatban, és a lánckód a szülő vagy a gyermek titkos kódját már nem tudja kompromittálni. A megerősített kulcs levezető függvény majdnem megegyezik a gyermekek szokásos titkos kulcs levezetési függvényével, kivéve, hogy a hash függvény a szülő nyilvános kulcsa helyett a szülő titkos kulcsát használja, amint azt a 4-13. ábra mutatja.



4-13. ábra: Gyermek kulcsok megerősített levezetése, a szülő nyilvános kulcs nem szerepel benne

Mikor megerősített titkos kulcs származtatás történik, a eredményként kapott titkos kulcs és a lánckód teljesen különbözik a szokásos származtató függvény eredményétől. Az így kapott „ágón” a kulcsokból olyan kiterjesztett nyilvános kulcsok állítható elő, melyek nem támadhatóak, mivel az általuk tartalmazott lánckód alapján semmilyen privát kulcsot sem lehet előállítani. Ennek megfelelően a megerősített származtatást használják arra, hogy a fát „elszigeteljék” a kiterjesztett nyilvános kulcsok szintje fölötti résztől.

Egyszerűen arról van szó, hogy ha a kiterjesztett nyilvános kulcsok kínálta kényelmet szeretnénk használni az ágak nyilvános kulcsainak a levezetése során, de nem szeretnénk kitenni magunkat a lánc kód kiszivárgása által okozott veszélynek, akkor a kiterjesztett nyilvános kulcsot egy megerősített szülőből kell létrehoznunk, nem pedig egy szokásos szülőből. A legjobb, ha a mesterkulcs 1. szintű gyermeket mindig megerősített levezetéssel állítjuk elő, mert így meg tudjuk akadályozni a mesterkulcsok kompromittálódását.

A szokásos és a megerősített kulcsképzés indexszámai

A kulcs képző függvényben az indexszám egy 32 bites egész. Annak érdekében, hogy könnyű legyen megkülönböztetni a szokásos kulcsképzést a megerősített kulcsképzéstől, az indexszámot két tartományra osztották. A 0 és $2^{31}-1$ (0x0 és 0x7FFFFFFF) közötti indexszámokat *kizárálag* a szokásos normál kulesképzésre használják. A 2^{31} és $2^{32}-1$ (0x80000000 és 0xFFFFFFFF) közötti indexszámokat pedig *kizárálag* a megerősített kulcsképzésre. Ezért, ha az indexszám 2^{31} -nél kisebb, akkor a gyermek normál módon lett képezve, míg ha az indexszám 2^{31} -nél nagyobb vagy egyenlő, akkor a gyermek megerősített módon lett képezve.

Az indexszám könnyebb megjelenítése érdekében a megerősített gyermek esetén az indexszám jelölése 0-tól kezdődik, de egy vessző áll mögötte. A szokásos gyermek kulcs jelölése a 0-tól kezdődik, míg az első megerősített gyermek (melynek indexe 0x80000000) jelölése a következő: 0'. A sorban a következő megerősített kulcs indexe 0x80000001, melynek jelölése 1', stb. A HD pénztárcáknál az i' index az jelenti, hogy $2^{31}+i$.

A HD pénztárca kulcs azonosítója (útvonal)

A HD pénztárcák kulcsait egy „útvonal” azonosítja, amelyben a szinteket egy per jel (/) választja el egymástól (lásd a 4-8. táblázatot). A titkos mesterkulcsból vezetett titkos kulcsok az "m" betűvel kezdődnek. A nyilvános mesterkulcsból vezetett nyilvános kulcsok az "M" betűvel kezdődnek. Ennek megfelelően a titkos mesterkulcs első gyermeké az m/0. A nyilvános kulcs első gyermeké az M/0. Az első gyermek második unokája az m/0/1, és így tovább.

Egy kulcs „ősei” jobbról balra olvashatók ki, amíg el nem jutunk a mesterkulcs hoz, amelyből a kulcs származik. Például az m/x/y/z azonosító azt a kulcsot jelenti, amely az m/x/y kulcs z-ik gyermeké, ahol az y az m/x y-ik gyermeké, ahol az x az m x-ik gyermeké.

4-8. táblázat: Példák HD pénztárca útvonalakra

HD útvonal	Kulcs leírása
m/0	A titkos mesterkulcsból (m) származó első (0) leszármazott titkos kulcsa.
m/0/0	Az első gyermek (m/0) első unokája
m/0'/0	Az első megerősített gyermek kulcs (m/0') első normális unokája
m/1/0	A második gyermek (m/1) első unokájának titkos kulcsa
M/23/17/0/0	A 24-ik gyermek 18-ik unokájának első dédunokájához tartozó nyilvános kulcs

Navigálás a HD pénztárca fa struktúrájában

A HD pénztárcák fa szerkezete hihetetlenül rugalmas. Mindegyik szülő-kulcsnak 4 milliárd gyermeké lehet: 2 milliárd normális gyermeké és 2 milliárd megerősített gyermeké. Ezen gyermekek mindegyikének szintén 4 milliárd gyermeké lehet, és így tovább. A fa tetszőleges mélységű lehet, és végtelen sok generációt tartalmazhat. A rugalmassággal azonban együtt jár az is, hogy ebben a végtelen fában egészen nehéz a navigálás. Különösen nehéz a HD pénztárcák különféle implementációk közötti mozgatása, mivel az ágak belső felépítésére végtelen sok lehetőség van.

A Bitcoin Javítására tett Javaslatok (Bitcoin Improvement Proposals (BIP-ek) megoldást nyújtanak erre a problémára: szabványos fa szerkezeteket javasolnak a HD pénztárcák felépítésére. A BIP0043 azt javasolja, hogy az első megerősített gyermek indexét különleges azonosító gyanánt használják, amely a fa szerkezet „célját” adja meg. A BIP0043 alapján a HD pénztárcák csak a fa 1-szintű ágait használhatják, ahol a cél definiálása révén az indexszám azonosítja a fa további részének névterét és szerkezetét. Például egy HD pénztárca, amely csak az m/i'/ ágat használja, egy adott céllal szolgál, és ezt a célt az "i" indexszám adja meg.

A BIP0044 ennek a specifikációnak a kiterjesztésével egy többszörös számla szerkezetet javasol, melynek „célját” a BIP0043 alatt a 44' adja meg. Az összes, BIP0044 szerkezetnek megfelelő pénztárcát az azonosítja, hogy a fának csak egyetlen ágát használja: m/44'.

A BIP0044 definíciója szerint a fa szerkezet öt, előre definiált szintből áll:

```
m / cél' / érme_típus' / számla' / visszajáró / cím_index
```

Az első szinten lévő „cél” értéke mindig 44'. A második szinten lévő „érme típus” a digitális pénzt fajtáját határozza meg, és ily módon több pénznem kezelését is lehetővé teszi egy HD pénztárcában: minden pénznemnek saját al-faja van a második szinten. Jelenleg három pénznem van definiálva: a Bitcoin az m/44'/0', a Bitcoin Testnet az m/44'/1'; a Litecoin pedig az m/44'/2'.

A fa harmadik szintjét a „számla” alkotja, amely lehetővé teszi, hogy a felhasználók a pénztárcáikat logikailag különálló al-számlákra osszák, pl. könyvelési vagy szervezeti szempontok alapján. Például egy HD pénztárca az alábbi két „számlát” tartalmazhatja: m/44'/0'/0' és m/44'/0'/1'. Mindegyik számla a saját rész-fájának a gyökerét alkotja.

A negyedik szinten, a „visszajáró” pénz szintjén a HD pénztárcáknak két al-fája van: az egyik a fogadó címek, a másik a visszajáró pénz számára. Figyeljék meg, hogy míg az előző szintek megerősített kulcs származtatást használtak, ez a szint normál származtatást használ. Ez lehetővé teszi az ezen a szinten lévő kiterjesztett nyilvános kulcsok exportját, és nem fokozott biztonságú környezetben történő használatát. A HD pénztárca a használható címeket a negyedik szint gyermekeiként definiálja, vagyis a fa ötödik szintjéből lesz a „cím index”. Például a fő számla harmadik fogadó címe az lesz, hogy M/44'/0'/0'/0/2. A 4-9. táblázat néhány további példát mutat.

4-9. táblázat: Példák a BIP0044 HD pénztárca szerkezetre

HD útvonal	Kulcs leírása
M/44'/0'/0'/0/2	A fő bitcoin számla harmadik nyilvános fogadó kulcsa
M/44'/0'/3'/1/14	A negyedik bitcoin számla visszajáró pénz kezelésre szolgáló 15-ik nyilvános kulcsa
m/44'/2'/0'/0/1	Egy Litecoin főszámla tranzakciók aláírására szolgáló második titkos kulcsa

Bitcoin Explorer-rel végzett kísérletek HD pénztárcákkal

A [3. fejezetben](#) bevezetett Bitcoin Explorer parancssori eszközzel különféle kísérleteket végezhetünk BIP0032 determinisztikus kulcsok előállítására és kiterjesztésére vonatkozóan, valamit különféle formátumokban tudjuk megjeleníteni őket :

```
$ bx seed | bx hd-new > m # új titkos mesterkulcs előállítása a magból, és
tárolása az "m" állományban
$ cat m # a titkos mesterkulcs kiíratása
xprv9s21ZrQH143K38iQ9Y5p6qoB8C75TE71NfpyQPdfGvzghDt39DHPFpovvtWZaRgY5uPwV7R
pEgHs7cvdgfiSjLjbjuGKGcjRyU7RGGSS8Xa
$ cat m | bx hd-public # az M/0 kiterjesztett nyilvános kulcs előállítása
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgyeQbBs2UAR6KECeeMVK
ZBPLrtJunSDMstweyLXhRgPxpd14sk9tJPW9
$ cat m | bx hd-private # az m/0 kiterjesztett titkos kulcs előállítása
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9
CAWrUE9i6GoNMKUga5biW6Hx4tw2six3b9c
$ cat m | bx hd-private | bx hd-to-wif # az m/0 titkos kulcs kiíratása WIF
formátumban
L1pbvV86crAGoDzqmgY85xURkz3c435Z9nirMt52UbnGjYMzKBUN
$ cat m | bx hd-public | bx hd-to-address # M/0 bitcoin címének kiíratása
1CHCnCjgMNb6digimckNQ6TBVcTWBAmPHK
$ cat m | bx hd-private | bx hd-private --index 12 --hard | bx hd-private -
-index 4 # m/0/12'/4 előállítása
xprv9yL8ndfdPVeDWJenF18oiHguRUj8jHmVrqqD97YQHeTcR3LCeh53q5PXPkLsy2kRaqgwoS6
YZBLatZRyUeAkRPe1kLR1P6Mn7jUrXFquUt
```

Kódolt titkos kulcsok (BIP0038)

A következő részben kulcsok és címek egyéb fajtáira fogunk példákat látni, pl. a kódolt (titkosított) titkos kulcsokra, script és multi-sig címekre, kérkedő címekre, valamint papír pénztárcákra.

Titkosított (kódolt) titkos kulcsok (BIP0038)

A titkos kulcsoknak titokban kell maradniuk. A titkos kulcsok *bizalmas* volta olyan evidencia, amelyet a gyakorlatban egészen nehéz megvalósítani, mivel ütközik egy ugyanilyen fontos biztonsági céllal, a *rendelkezésre állással*. A titkos kulcsok titokban tartása sokkal nehezebb, ha a titkos kulcsokról biztonsági másolatokat kell tárolni, nehogy elveszítsük őket. A pénztárcákban lévő, jelszóval védett titkos kulcsok biztonságban vannak, de a pénztárcáról biztonsági másolatot kell készíteni. Néha a felhasználók az egyik pénztárcából a másikba mozgatják át a kulcsokat – például a pénztárca program újabb változatának installálásakor vagy egy másik programra való lecserélésekor. A titkos kulcsokról készített biztonsági mentések papíron (lásd a „[Papír pénztárcák](#)” részt) vagy külső tároló eszközön, pl. USB kulcson is tárolhatók. De mi történik, ha a másolatot ellopják vagy elveszítjük? Ezek az egymásnak ellentmondó biztonsági követelmények vezettek egy hordozható és kényelmes szabvány, a BIP0038 létrejöttéhez (lásd [\[bip0038\]](#)), mellyel a titkos kulcsok úgy titkosíthatók, hogy sok különféle pénztárca és bitcoin kliens megértse őket.

A BIP0038 szabvány a titkos kulcsok jelmonddal történő titkosításáról, és Base58Check kódolásáról szól. Célja az, hogy a titkos kulcsok biztonságosan tárolhatók legyenek a mentő eszközön, és átvihetők legyenek a pénztárcák között, vagy olyan körülmények között is kezelhetők legyenek, ahol a kulcs nyilvánosságra kerülhet. A BIP0038 titkosítási szabvány az AES-t (Advanced Encryption Standard) használja, melyet az Amerikai Szabványügyi Hivatal

(NIST, National Institute of Standards and Technology) fogadott el, és széles körben használják kereskedelmi és katonai alkalmazásokban.

A BIP0038 titkosítás esetén a titkos kulcsból indulunk ki, amely általában Base58Check string formájában, „5” előtaggal, WIF formátumban (Wallet Import Format, pénztárca import formátum) van kódolva. Ezen kívül a BIP0038 titkosításnak egy jelmondatra – egy hosszú jelszóra – van szüksége, amely általában számos szóból vagy egy bonyolult alfanumerikus karakterláncból áll. A BIP0038 titkosítás eredménye egy olyan Bas58Check kódolású titkos kulcs, amely a 6P előtaggal kezdődik. Ha egy olyan kulccsal találkoznak, amely 6P-vel kezdődik, az azt jelenti, hogy a kulcs kódolt, és egy jelmondatra van szükség ahhoz, hogy WIF-formátumú titkos kulccsá tudjuk visszaalakítani (visszakódolni), amely 5-tel kezdődik, és bármelyik pénztárcában használható. Sok pénztárca alkalmazás felismeri a BIP0038 kódolású titkos kulcsokat. Ezek a kulcs dekódolása és importálása céljából megkérdezik a felhasználótól, hogy mi a jelmondat. Vannak egyéb alkalmazások, pl. a hihetetlenül hasznos web böngésző alapú [Bit Address](#), amellyel (a „pénztárca részletei” fülön) szintén elvégezhető a BIP0038 kulcsok dekódolása.

A BIP0038 titkosított kulcsokat leggyakrabban a papír pénztárcák esetén alkalmazzák. A papír pénztárcákkal a titkos kulcsok egy papírlapon tárolhatók. Ha a felhasználó egy erős jelmondatot választ, a BIP0038 kódolt papír pénztárcák nagyon biztonságosak, és kiválóan alkalmasan arra, hogy egy offline bitcoin tárolót hozzunk létre (ezeket „hűtő tárolónak” (cold storage) is hívják).

A 4-10. táblázatban látható kódolt kulcsok a bitaddress.org-gal lettek előállítva, és az szemléltetik, hogyan lehet egy kulcsot egy jelmondat beadásával kódolni:

4-10. táblázat: Példa egy BIP0038 kódolt titkos kulcsra

Titkos kulcs (WIF)	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Jelmondat	MyTestPassphrase
A titkosított kulcs (BIP0038)	6PRTHL6mWa48xSopbU1cKrVjpKbBZxcLRRCDctLJ3z5yxE87MobKoXdTsJ

Fizetés script hashnek (P2SH, pay to script hash) címek és több aláírást megkövetelő (multi-sig) címek

Mint tudjuk, a hagyományos bitcoin címek „1”-gyel kezdődnek, és a nyilvános kulcsból származnak, a nyilvános kulcs pedig a titkos kulcsból. Az „1”-gyel kezdődő címekre bárki küldhet bitcoint, de csak az tudja elkölni, aki be tudja mutatni a titkos kulccsal létrehozott megfelelő aláírást és a nyilvános kulcs zanzáját.

A „3”-mal kezdődő bitcoin címek fizetés-script-hashnek (P2SH) bitcoin címek, melyeket néha hibásan több aláírást megkövetelő, vagy multi-sig címeknek hívnak. A bitcoin tranzakció kedvezményezettjét a script hashével, nem pedig a nyilvános kulcs tulajdonosával adják meg. Ezt az újítást 2012. januárjában, a BIP0016 keretében vezették be (lásd [\[bip0016\]](#)). Az újítás széles körben elterjedt, mert lehetővé teszi, hogy magához a címhez legyen hozzárendelve valamelyen funkcionálitás. A hagyományos, „1” kezdetű bitcoin

címeket használó tranzakciók neve fizetés-nyilvános-kulcs-hashnek (P2PKH, pay-to-public-key-hash). Ezekkel a hagyományos tranzakciókkal szemben, a „3” kezdetű címekre küldött pénzek esetében nem csak egy nyilvános kulcs hashének bemutatására és a tulajdonjogot bizonyító, titkos kulccsal történő aláírására van szükség. A követelmények meghatározása a cím létrehozásakor történik. A cím a hozzá tartozó összes bemenetet ugyanolyan módon korlátozza.

A fizetés-script-hashnek címet egy tranzakciós scriptból hozzák létre. Ez a tranzakció határozza meg, hogy ki költheti el a tranzakció kimenetét (részletesebben lásd a „[Fizetés script hashnek \(P2SH\)](#)” részt). A fizetés-scrip-hashnek cím kódolásához ugyanúgy a kettős hash függvényt kell használni, mint a hagyományos bitcoin címeknél, de a nyilvános kulcs helyett a scripten kell a műveletet elvégezni:

```
script hash = RIPEMD160(SHA256(script))
```

Az eredményként kapott „script hasht” Base58Check segítségével „5” verzió előtaggal kódolják, ami egy 3-mal kezdődő címet eredményez. Pl. egy P2SH cím: `32M8ednmuyZ2zVbes4puqe44NZumgG92sM`, amelyet a Bitcoin Explorer következő parancsaival lehet előállítani: `script-encode`, `sha256`, `ripemd160`, és `base58check-encode` (lásd „[A Libbitcoin és a Bitcoin Explorer](#)”). A parancsok a következőképpen használhatók:

```
$ echo dup hash160 [ 89abcdefabbaabbaabbaabbaabbaabbaabba ] equalverify
checksig > script
$ bx script-encode < script | bx sha256 | bx ripemd160 | bx base58check-
encode --version 5
3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM
```



A P2SH nem feltétlenül egyezik meg egy több aláírást megkövetelő, szabványos multi-sig tranzakcióval. A P2SH *leggyakrabban* egy multi-sig scriptnek felel meg, de más tranzakciótípusok scriptjeit is ábrázolhatja.

Multi-signature címek és P2SH

Jelenleg a P2SH függvényt a leggyakrabban a multi-sig script esetén alkalmazzák. Mint a multi-sig script neve is mutatja, a tulajdonjog igazolásához és a pénz elköltséhez egynél több aláírást követel meg. A bitcoin multi-sig N kulcs esetén M aláírásra van szükség. Ennek M-of-N multi-sig a neve, ahol M kisebb vagy egyenlő, mint N. például, Bob, az [1. fejezetben](#) megismert kávéház tulajdonos használhat olyan 1-of-2 multi-sig címeket, amelyeknél az egyik kulcs az övé, a másik a feleségéé, vagyis mindenki el tudja költeni az ilyen címeken lévő zárolt tranzakció kimeneteket. Ez hasonlít a hagyományos bankok „közös számlájához”, ahol a számlatulajdonosok bármelyike egyedül is képes számlaműveleteket végezni. Gopeshnek, a web tervezőnek, aki Bob web helyét tervezte, lehet viszont egy 2-of-3 multi-sig címe az üzleti vállalkozásához, ami biztosítja, hogy a címről csak akkor lehet pénzt költeni, ha az üzlettársak közül legalább kettő aláírja a tranzakciót.

Az [5. fejezetben](#) fogjuk megvizsgálni, hogyan lehet P2SH tranzakciókat létrehozni és hogyan lehet P2SH tranzakciókról pénzt költeni.

Kérkedő címek

A kérkedő címek olyan bitcoin címek, melyek olvasható üzeneteket tartalmaznak, például az `1LoveBPZZD72PUXLzCkYAtGFYmK5vYNR33` egy olyan érvényes cím, amely a „Love” szót tartalmazza az „1” utáni négy Base-58 betűn. A kérkedő címekhez titkos kulcsok milliárdjait kell generálni és tesztelni, amíg a származtatott bitcoin címben létre nem jön a kívánt minta. Noha a kérkedő címet előállító algoritmusban vannak optimalizálások, a folyamat alapjában véve annak felel meg, hogy véletlenszerűen választunk egy titkos kulcsot, előállítjuk belőle a nyilvános kulcsot, ebből pedig a bitcoin címet, és leellenőrizzük, hogy megfelel-e a kívánt mintának – mindezt milliárdnyiszer megismételve, amíg sikerrel nem járunk.

Ha találtunk egy kérkedő címet, amely megfelel a kívánt mintának, akkor a titkos kulcs épp úgy használható, mint bármely más címnél. A kérkedő címek épp olyan biztonságosak, mint a többi bitcoin cím. Ugyanaz az elliptikus görbükkel történő titkosítás (ECC, Elliptic Curve Cryptography) és biztonságos hash algoritmus (SHA, Secure Hash Algorithm) van mögöttük, mint bármely más cím mögött. Ha van egy adott mintával rendelkező kérkedő cím, akkor ennél sem lehet könnyebben megtalálni a titkos kulcsot, mint bármely más cím esetén.

Az [1. fejezetben](#) találkoztunk Eugéniával, aki egy gyermek-védelmi alap vezetője a Fülöpszigeteken. Tegyük fel, hogy Eugénia egy bitcoin gyűjtést szervez, és a nagyobb reklám érdekében szeretne a gyűjtéshez egy kérkedő bitcoin címet használni. Eugénia egy olyan kérkedő címet fog létrehozni, amely úgy kezdődik, hogy „1Kids”, ezzel is elősegítve a gyűjtést a gyerekek számára. Vizsgáljuk meg, hogyan hozható létre ez a kérkedő cím, és mit jelent mindez Eugénia gyűjtésének a biztonsága szempontjából.

Kérkedő címek előállítása

Fontos megértenünk, hogy a bitcoin cím egyszerűen csak egy szám, amely az Base-58 ábécé szimbólumaival van ábrázolva. Az „1Kids” minta keresése az `1Kids11111111111111111111111111111111` és az `1Kidszzzzzzzzzzzzzzzzzzzzzzzzzz` között történhet. Kb. 58^{29} (kb. $1.4 * 10^{51}$) ilyen cím van ebben a tartományban, és ezek mindegyike úgy kezdődik, hogy „1Kids”. A 4-11. táblázatban látható az „1Kids” kezdető címek címtartománya.

4-11. táblázat: Az „1Kids” kezdetű kérkedő címek tartománya

Mettől	<code>1Kids11111111111111111111111111111111</code>
	<code>1Kids11111111111111111111111111111112</code>
	<code>1Kids11111111111111111111111111111113</code>
	...
Meddig	<code>1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzz</code>

Tekintsük úgy az „1Kids” mintát, mint egy számot, és nézzük meg, milyen gyakran található meg ez a minta egy bitcoin címben (lásd 4-12. táblázat). Egy átlagos asztali számítógéppel, melyben nincs semmilyen célhardver, másodpercenként kb. 100'000 kulcs vizsgálható meg.

4-12. táblázat: Egy kérkedő cím minta (1KidsCharity) előfordulási gyakorisága és a megtalálásához szükséges idő egy asztali számítógépen

Hossz	Minta	Gyakoriság	Átlagos keresési idő
1	1K	58-ból 1	< 1 millisec
2	1Ki	3364-ből 1	50 millisec
3	1Kid	195'000-ből 1	< 2 mp
4	1Kids	11 millióból 1	1 perc
5	1KidsC	656 millióból 1	1 óra
6	1KidsCh	38 milliárdból 1	2 nap
7	1KidsCha	2.2 billióból 1	3–4 hónap
8	1KidsChar	128 billióból 1	13–18 év
9	1KidsChari	7000 billióból 1	800 év
10	1KidsCharit	0.4 trillióból 1	46'000 év
11	1KidsCharity	23 trillióból 1	2.5 millió év

Mint látható, Eugénia nem fogja tudni az „1KidsCharity” címet belátható idő alatt létrehozni, még akkor sem, ha sok ezer számítógépet használ. minden egyes további karakter 58-szorosára növeli a nehézséget. A hét karakternél hosszabb mintákat általában speciális hardverrel, pl. erre a célre összeépített asztali számítógépekkel keresik, melyekben több grafikus feldolgozó egység (GPU, Graphical Processing Unit) található. Ezek általában olyan újrahasznosított bitcoin bányász „platformok”, melyek bitcoin bányászatra már gazdaságtalanok, de a kérkedő címek keresésére még hatékonyan használhatók. A GPU-val rendelkező rendszereken a kérkedő címek keresése sok nagyságrenddel gyorsabb lehet, mint egy általános célú CPU-n.

Kérkedő címek úgy is előállíthatók, hogy megbízást adunk egy bányász közösségnak, amely ilyen címek keresésére szakosodott, lásd pl. a [Vanity Pool](#) web címet. Ez a bányatársaság a GPU hardverrel rendelkező tagok számára lehetővé teszi, hogy a kérkedő címek keresése révén bitcoinokhoz jussanak. Egy kis fizetség fejében (0.01 bitcoin, vagyis írásunk idején kb. 5\$ ellenében) Eugénia külső megbízást adhat a 7-karakterből álló minta megkeresésére, és ahelyett, hogy egy CPU-n hónapokig keresné a mintát, már néhány órán belül megkapja az eredményt.

Egy kérkedő cím előállítása a nyers erő módszerével történik: kipróbálunk egy véletlen kulcsot, és megnézzük, hogy az így kapott cím illeszkedik-e a kívánt mintával. Ha nem, akkor megismétljük a folyamatot. A 4-7. példa a „kérkedő címek bányászatára” mutat egy példát, vagyis egy olyan C++ programot mutat be, mellyel kérkedő címek állíthatók elő. A példák a libbitcoin könyvtárat használják, melyet az „[Alternatív kliensek, könyvtárak és eszközök](#)” részben ismertettünk.

4-7. példa: Kérkedő cím bányászat

```
$ cat vanity-miner.cpp
#include <bitcoin/bitcoin.hpp>

// The string we are searching for
const std::string search = "1kid";

// Generate a random secret key. A random 32 bytes.
bc::ec_secret random_secret(std::default_random_engine& engine);
// Extract the Bitcoin address from an EC secret.
std::string bitcoin_address(const bc::ec_secret& secret);
// Case insensitive comparison with the search string.
bool match_found(const std::string& address);

int main()
{
    // random_device on Linux uses "/dev/urandom"
    // CAUTION: Depending on implementation this RNG may not be secure
    enough!
    // Do not use vanity keys generated by this example in production
    std::random_device random;
    std::default_random_engine engine(random());

    // Loop continuously...
    while (true)
    {
        // Generate a random secret.
        bc::ec_secret secret = random_secret(engine);
        // Get the address.
        std::string address = bitcoin_address(secret);
        // Does it match our search string? (1kid)
        if (match_found(address))
        {
            // Success!
            std::cout << "Found vanity address! " << address << std::endl;
            std::cout << "Secret: " << bc::encode_hex(secret) << std::endl;
            return 0;
        }
    }
    // Should never reach here!
    return 0;
}

bc::ec_secret random_secret(std::default_random_engine& engine)
{
    // Create new secret...
    bc::ec_secret secret;
    // Iterate through every byte setting a random value...
    for (uint8_t& byte: secret)
        byte = engine() % std::numeric_limits<uint8_t>::max();
    // Return result.
    return secret;
}

std::string bitcoin_address(const bc::ec_secret& secret)
{
    // Convert secret to pubkey...
    bc::ec_point pubkey = bc::secret_to_public_key(secret);
    // Finally create address.
    bc::payment_address payaddr;
```

```

        bc::set_public_key(payaddr, pubkey);
        // Return encoded form.
        return payaddr.encoded();
    }

bool match_found(const std::string& address)
{
    auto addr_it = address.begin();
    // Loop through the search string comparing it to the lower case
    // character of the supplied address.
    for (auto it = search.begin(); it != search.end(); ++it, ++addr_it)
        if (*it != std::tolower(*addr_it))
            return false;
    // Reached end of search string, so address matches.
    return true;
}

```



A fenti példa a `std::random_device`-t használja. A megvalósítástól függően ez akár egy kriptográfiailag biztonságos véletlenszám generátor (cryptographically secure random number generator (CSRNG)) is lehet. A UNIX-szerű operációs rendszerek, pl. a Linux esetén a példa a `/dev/urandom`-ot fogja használni. A véletlenszám generátor itt csupán szemléltetésre szolgál. Éles rendszerben *nem* használható bitcoin kulcsok előállítására, mivel ez a megvalósítás nem rendelkezik elégsges biztonsággal.

Ezt a példát egy C++ fordítóval kell lefordítani, és össze kell szerkeszteni a libbitcoin könyvtárral (előbb a libbitcoin könyvtárat kell installálni a rendszeren). A példa futtatása úgy lehetséges, hogy a `vanity-miner` végrehajtható programot paraméterek nélkül futtatjuk (lásd 4-8. példa). A program egy "1kid" kezdetű kérkedő címet próbál találni.

4-8. példa: A kérkedő cím bányászatára vonatkozó példa fordítása és futtatása

```

$ # A kód lefordítása g++ -szal
$ g++ -o vanity-miner vanity-miner.cpp $(pkg-config --cflags --libs
libbitcoin)
$ # A példa futtatása
$ ./vanity-miner
Kérkedő címet találtam! 1KiDzkG4MxmovZryZRj8tK81oQRhbZ46YT
Titok: 57cc268a05f83a23ac9d930bc8565bac4e277055f4794cbd1a39e5e71c038f3f
$ # Újrafuttatáskor különböző lesz az eredmény
$ ./vanity-miner
Kérkedő címet találtam! 1Kidxr3wsmMzzouwXibKfwTYs5Pau8TUFn
Secret: 7f65bbbbe6d8caae74a0c6a0d2d7b5c6663d71b60337299a1a2cf34c04b2a623
$ # A "time" használatával vizsgálható meg, mennyi ideig tart egy eredmény
megtalálása
$ time ./vanity-miner
Kérkedő címet találtam! 1KidPWhKgGRQWD5PP5TAnGfDyfWp5yceXM
Titok: 2a802e7a53d8aa237cd059377b616d2bfcfa4b0140bc85fa008f2d3d4b225349

real    0m8.868s
user    0m8.828s
sys     0m0.035s

```

A mintapélda néhány másodperc alatt talált egy három karakteres mintát ("kid"), amint azt az időmérésre szolgáló `time` Unix parancsból láthatjuk. A `search` keresési minta megváltoztatásával megvizsgálhatjuk, hogy meddig tart egy négy vagy öt karakteres minta megtalálása!

A kérkedő címek biztonsága

A kérkedő címek valódi kételű kardot jelentenek, mert a biztonság fokozható is, de *csökkenthető* is velük. Ha a kérkedő címeket a biztonság javítására használjuk, a jellegzetes címek megnehezítik, hogy a támadó a saját címét helyettesítse be, és az ügyfelek neki fizessék. Sajnos, a kérkedő címek azt is lehetővé teszik, hogy bárki létrehozzon egy olyan címet, ami *hasonlít* egy másik véletlen címhez, vagy akár egy másik kérkedő címhez, és így be tudja csapni az ügyfeleket.

Eugénia eljárhat úgy, hogy egy véletlenszerűen generált címet tesz közzé (pl. 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy), amelyre bárki elküldheti az adományát. Vagy generálhat egy kérkedő címet is, amelynek 1Kids a kezdete, hogy jellegzetesebbé tegye a címet.

Mindkét esetben az egyetlen fix cím használatának (az egyes adományozóknak külön, dinamikusan generált dinamikus címekkel szemben) az a veszélye, hogy egy tolvaj behatolhat a webre, és a saját címével helyettesítheti a címet, ezzel az adományokat magához irányíthatja át. Ha az adományokat fogadó cím számos különböző helyen lett reklámozva, akkor a felhasználók az utalás előtt vizuálisan ellenőrizni tudják, hogy valóban ugyanaz-e a cím, mint amit a weben, a nekik küldött levélben vagy szórólapon láttak. Egy olyan véletlen cím esetében, mint amilyen pl. a 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy, az átlagos felhasználó az első néhány karaktert ellenőrzi, például a "1J7mdg"-t, és ha ez egyezik, akkor úgy tekinti, hogy a cím helyes. Ha valaki lopási céllal egy hasonlónak látszó címet állít elő egy kérkedő címet generáló programmal, akkor gyorsan generálható egy olyan cím, melynek első néhány karaktere megegyezik az adománygyűjtés címével. Lásd a 4-13. táblázatot.

4-13. táblázat: Egy véletlen címmel egyező kérkedő cím előállítása

Eredeti véletlen cím	1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
Kérkedő cím (4 kar. egyezés)	1J7md1QqU4LpctBetHS2ZoyLV5d6dShhEy
Kérkedő cím (5 kar. egyezés)	1J7mdgYqyNd4ya3UEcq31Q7sqRMXw2XZ6n
Kérkedő cím (6 kar. egyezés)	1J7mdg5WxGENmwyJP9xuGhG5KRzu99BBCX

Növeli-e egy kérkedő cím a biztonságot? Ha Eugénia azt a kérkedő címet állítja elő, hogy 1Kids33q44erFfpeXrmDSz7zEqG2FesZEN, a felhasználók a kérkedő karaktereket, valamint az *ezek mögött álló néhány karaktert* fogják megvizsgálni, pl. a cím "1Kids33" részét. Ez arra kényszeríti a támadót, hogy egy olyan kérkedő címet állítson elő, amely legalább 6 karakter hosszú, de ehhez 3364-szer (58 * 58) több munkára van szükség, mint Eugéniának a 4 karakteres kérkedő címéhez. Lényegében az Eugénia (vagy az általa megfizetett bányászközösséggel) által elvégzett munka arra „kényszeríti” a támadót, hogy hosszabb kérkedő címet állítson elő. Ha Eugénia egy bányászközösséget fogad föl egy 8 karakter hosszú kérkedő cím előállítására, akkor ezáltal a támadó a 10 karakteres tartományba kényszerül, amelynek személyi számítógépen lehetetlen az előállítása, de még egy cél-hardverrel vagy bányászközösséggel is nagyon költséges. Ami Eugéniának még megfizethető, a támadónak megfizethetetlen, különösen akkor, ha a csalás által szerezhető pénz arra sem elég, hogy fedezze a kérkedő cím előállításának a költségét.

Papír pénztárcák

A papír pénztárcák papírra kinyomtatott privát kulcsok. A papír pénztárca kényelemi okokból gyakran a titkos kulcshoz tartozó bitcoin címet is tartalmazza, de ez nem feltétlenül szükséges, mivel a bitcoin cím előállítható a titkos kulcsból. A papír pénztárcák nagyon hatékony módszert jelentenek biztonsági mentések, vagy offline bitcoin tárolók létrehozására. Az offline bitcoin tárolók neve: „hideg tároló”. Mentési mechanizmusként a papír pénztárca megvédi a kulcsot, ha egy számítógép meghibásodik, pl. ha tönkremegy a diszkje, vagy ha ellopják a számítógépet, vagy ha a kulcs véletlenül törlésre kerül. A papír pénztárcák „hideg tárolóként” nagyon biztonságos védelmet jelentenek a hackerekkel, key-loggerekkal és más számítógépes fenyegetésekkel szemben, ha offline állították elő őket, és soha nem voltak online rendszeren tárolva.

A papír pénztárcák sokféle alakban és méretben léteznek, de lényegében csupán egy papírra kinyomtatott kulcsból és címből állnak. A papír pénztárcáknak ez a legegyszerűbb alakja (4-14. táblázat).

4-14. táblázat: A legegyszerűbb papír pénztárca: a bitcoin cím és a titkos kulcs kinyomtatva

Bitcoin cím	Titkos kulcs (WIF)
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn

A papír pénztárcák könnyen előállíthatók olyan eszközökkel, mint pl. a bitaddress.org címen található kliens-oldali Javascript generátor. Ez a web lap a kulcsok és a papír pénztárcák előállításához szükséges összes kódot tartalmazza, és a működéséhez nincs szükség Internet kapcsolatra. Használatához mentsük el a HTML oldalt a lokális meghajtónra vagy egy külső USB meghajtóra. Szakítsuk meg az Internet kapcsolatot, és nyissuk meg az állományt egy Web böngészőben. Még jobb, ha egy friss operációs rendszert töltünk be, például egy CDROM-ról bootolható Linuxot. Míg offline vagyunk, az eszköz által generált kulcsok a helyi nyomtatón egy USB kábellel (nem Wifi-vel) kinyomtathatók, ezáltal olyan papír tárcák állíthatók elő, melyek kulcsai csak a papíron léteznek, és soha nem voltak online rendszerben tárolva. Ha ezeket a papír pénztárcákat egy tűz-biztos széfbe tesszük, és bitcoint „küldünk” a bitcoin címeikre, akkor így egy egyszerű, de nagyon hatékony „hideg tárolót” valósítunk meg. A 4-14. ábrán egy papír pénztárca látható, amely a bitaddress.org segítségével lett előállítva.



4-14. ábra: Példa egy egyszerű papír tárcára a bitaddress.org-ról

Az egyszerű papír pénztárcák hátránya az, hogy a kinyomtatott kulcsok sebezhetők a lopással szemben. Ha egy tolvaj hozzáfér a papír tárcához, akkor ellophatja vagy lefényképezheti a kulcsokat, és a birtokába juthat a kulcsok által őrzött bitcoinoknak. Egy fejlettebb papír pénztárca rendszer BIP0038 kódolt privát kulcsokat használ. A papír tárcára kinyomtatott kulcsokat jelmondat védi, melyet a tulajdonos kívülről tud. A jelmondat nélkül a kódolt kulcsok használhatatlanok. Ugyanakkor ez a megoldás még mindig jobb, mint egy jelszóval védett pénztárca, mert a kulcsok soha sem voltak online, és fizikailag kell őket elővenni egy széfből vagy más, fizikailag biztonságos tárolóból. A 4-15. ábrán egy BIP0038 titkosított privát kulccsal rendelkező papír pénztárca látható, amely a bitaddress.org segítségével lett létrehozva.

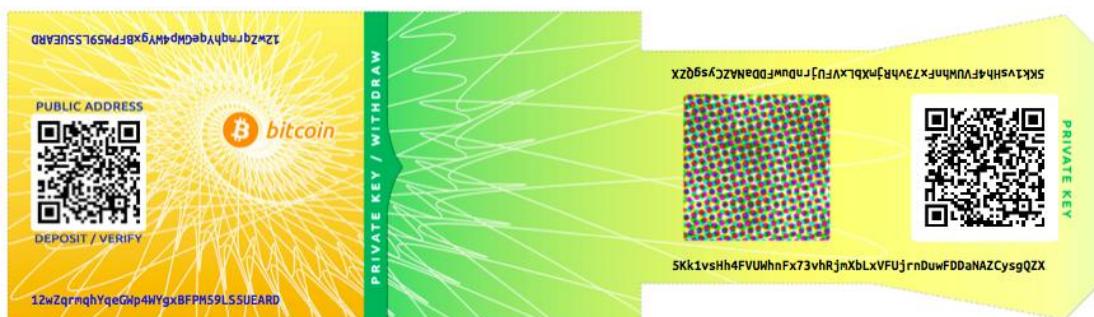


4-15. ábra: Példa egy kódolt papír tárcára a bitaddres.org-ról. A jelmondat: „test”



Egy papír tárcába akár többször is lehet pénzt küldeni, de a pénzt csak egyszer lehet belőle felvenni. Ekkor a benne lévő összes pénzt el kell költeni. Ez azért van így, mert a pénz elköltése során felfedjük a privát kulcsot, másfelől azért, mert némelyik pénztárca a visszajáró pénznek egy további, újabb címet állít elő, ha nem az egész összeget költjük el. Mindent úgy költhetünk el, ha papír tárcában lévő összes pénzt felvesszük, és a maradék pénzt egy új papír tárcába küldjük.

Sokfélé méretű és kivitelű papír tárca van, melyek különféle tulajdonságokkal rendelkeznek. Némelyik ajándékul szolgál, és az alkalomhoz illő témákat tartalmaz, pl. Karácsonyi vagy Újévi jeleneteket. Mások arra a célra szolgálnak, hogy egy banki páncélteremben vagy széfben őrizzük őket. Ezeken a privát kulcs valamelyen módszerrel ki van takarva, pl. nem átlátszó, lekaparható matricával, vagy össze van hajtogatva és egy biztonságos öntapadó fóliával van leragasztva. Az alábbi 4-16., 4-17. és 4-18. ábrákon különféle papír tárcák láthatók.



4-16. ábra: Példa egy papír tárcára a bitcoinpaperwallet.com-ról, ahol a titkos kulcs egy összehajtható fülön van.



4-17. ábra: A bitcoinpaperwallet.com-ról származó papír tárcá, ahol a titkos kulcs rejte van.

Más típusok leválasztható ellenőrző szelvények formájában a kulcsból és a címiből több példányt tartalmaznak, hasonlóan a jegyek ellenőrző szelvényeihez, ezáltal több példányban tárolhatók, ami megvédi őket a tűzesetek, árvizek és más természeti katasztrófák ellen.



4-18. ábra: Példa egy papír tárcára, amely egy tartalék „fülön” a kulcsok további másolatait tartalmazza.

5. Tranzakciók

Bevezetés

A tranzakciók a bitcoin rendszer legfontosabb részei. A bitcoinban minden más úgy lett megtervezve, hogy biztosítsa a tranzakciók létrehozását, hálózaton keresztüli továbbítását, ellenőrzését és végül a tranzakciók hozzáadását a rendszer globális főkönyvéhez, a blokklánchoz. A tranzakciók olyan adatstruktúrák, melyek az érték átruházását kódolják a bitcoin rendszer résztvevői között. Mindegyik tranzakció egy nyilvános bejegyzés a bitcoin kettős könyvelésében, a blokkláncban.

Ebben a fejezetben a tranzakciók különféle fajtáit vizsgáljuk: mit tartalmaznak, hogyan hozhatók létre, hogyan ellenőrizhetők és hogyan válnak az összes tranzakciót megorokító maradandó feljegyzés részévé.

A tranzakciók életciklusa

A tranzakciók életciklusa a tranzakció létrehozásával kezdődik. Ezután a tranzakció aláírásra kerül, vagyis egy vagy több aláírás kerül rá, ami engedélyezi a tranzakció által hivatkozott összegek elköltését. A tranzakció ezután továbbításra kerül a bitcoin hálózatban. A hálózat minden egyes csomópontja (résztvevője) ellenőri a tranzakciót, és továbbítja azt, amíg a tranzakció el nem jut a hálózat (majdnem) valamennyi csomópontjához. Végül a tranzakciót egy bányász csomópont ellenőri, és befoglalja egy tranzakciókat tartalmazó blokkba, amely a blokkláncban tárolódik.

Miután a tranzakció a blokkláncban tárolásra került, és a blokkot elégséges számú további blokk (megerősítés) követi, a tranzakció a bitcoin főkönyv állandó részévé válik, és az összes résztvevő érvényesnek tekinti. A tranzakció által az új tulajdonoshoz rendelt összeget ezután egy újabb tranzakcióban lehet elkölni. Az új tranzakcióval tovább bővül a tulajdonosi lánc, és a tranzakciós életciklus ismét elkezdődik.

Tranzakciók létrehozása

Segítségünkre lehet, ha a tranzakciót úgy képzeli el, mint egy papír csekket. Egy csekkhez hasonlóan a tranzakció is egy olyan eszköz, amellyel pénz továbbítási szándék fejezhető ki, de a pénzügyi rendszer számára csak akkor lesz látható, ha már fel lett adva végrehajtásra. A csekkhez hasonlóan a tranzakció kezdeményezője sem feltétlenül azonos a tranzakció aláírójával.

Tranzakciók online vagy offline módon még akkor is létrehozhatók, ha a tranzakciót létrehozó személy nincs meghatalmazva arra, hogy aláírja a számlát. Például egy pénztáros előkészítheti az igazgató által aláírt csekkeket. Hasonló módon, egy pénztáros létre tud hozni olyan bitcoin tranzakciókat, melyek később az igazgató digitális aláírása érvényesít. Míg a csekk esetében az összeg forrását egy adott számla jelenti, a bitcoin tranzakció nem egy számlára, hanem bizonyos előző tranzakciókra hivatkozik.

A tranzakciót a létrehozása után a forrás összeg tulajdonosa (vagy tulajdonosai) aláírják. Ha a tranzakció alakilag helyes és alá lett írva, akkor érvényessé válik, és az összes olyan információt tartalmazza, amely a pénzküldés végrehajtásához szükséges. Utolsó lépésként az

érvényes tranzakciónak el kell jutnia a bitcoin hálózatba, hogy továbbításra kerülhessen, és egy bányász befoglalhassa a nyilvános főkönyvbe, a blokkláncba.

A bitcoin tranzakció elküldése a bitcoin hálózatnak

Először is a tranzakciót el kell juttatni a bitcoin hálózatba, hogy továbbításra kerülhessen a többi csomópontnak és be lehessen foglalni a blokkláncba. Lényegében egy bitcoin tranzakció csupán 300-400 bajt adat, melynek a több tízezer bitcoin csomópont mindegyikéhez el kell jutnia. A küldőnek nem kell megbízna a tranzakció szétsugárzására használt csomópontokban, ha több csomópontot használatával biztosítja, hogy a tranzakció biztosan szétterjedjen. A csomópontoknak nem kell bízniuk a küldőben, és nem kell megállapítaniuk a küldő „személyazonosságát”. Mivel a tranzakció alá van írva, és nem tartalmaz bizalmas adatokat, titkos kulcsokat vagy tanúsítványokat, bármilyen mögöttes hálózati átviteli mechanizmussal közvetíthető. Ezzel szemben a hitelkártya tranzakciók bizalmas adatokat tartalmaznak, és csak titkosított hálózati kapcsolaton továbbíthatók. A bitcoin tranzakciók viszont bármilyen hálózatot használhatnak. Ha a tranzakció képes eljutni egy bitcoin csomóponthoz, amely továbbítja azt a bitcoin hálózatnak, lényegtelen, hogy a tranzakció hogyan jutott el az első csomóponthoz.

A bitcoin tranzakciók emiatt nem titkosított hálózati kapcsolatokkal is eljuttathatók a bitcoin hálózatba. Használható pl. Wifi, Bluetooth, Chirp, vonalkódok, vagy egy web nyomtatványba történő bemásolás. Rendkívüli esetekben a bitcoin tranzakció csomagkapcsolt rádióval, műholdas reléállomással vagy rövidhullámú adással is továbbítható. Ha fontos a fedett és zavarásmentes kommunikáció, akkor szort spektrumú kommunikáció vagy frekvencia ugrásos rendszerek használhatók. A bitcoin tranzakciók még hangulatjelzések (smileys) segítségével is kódolhatók, posztolhatók nyilvános fórumokon, vagy elküldhetők szöveges üzenet vagy Skype üzenet formájában. A bitcoin a pénzt adatstruktúrává változtatja át, és lényegében mindenkinél lehetővé teszi a bitcoin tranzakciók létrehozását és végrehajtását.

A tranzakciók szétterjedése a bitcoin hálózatban

Miután a bitcoin tranzakciót továbbítottuk a bitcoin hálózat egy tetszőleges csomópontjának, a csomópont ellenőri a tranzakciót. Ha a tranzakció érvényes, akkor a csomópont továbbítja a vele kapcsolatban lévő többi csomópontnak, és a sikerről szinkron módon egy üzenetet ad vissza a kezdeményezőnek. Ha a tranzakció érvénytelen, akkor a csomópont elutasítja a tranzakciót, és az elutasítás tényéről szinkron módon egy üzenetet küld a kezdeményezőnek.

A bitcoin hálózat egy peer-to-peer hálózat, ami azt jelenti, hogy minden bitcoin csomópont kapcsolatban van néhány további bitcoin csomóponttal. Ezeket a csomópontokat a kliens a peer-to-peer protokoll révén, az induláskor találja meg. Az egész hálózat egy lazán kapcsolódó háló, melynek nincs rögzített topológiája vagy valamelyen adott szerkezete, és amelyben az összes csomópont egyenrangú. Az üzeneteket, pl. a tranzakciókat és a blokkokat az egyes csomópontok az összes olyan csomópontnak továbbítják, amellyel kapcsolatban vannak. Ezt a folyamatot „elárasztásnak” („flooding”) hívják. Ha bármelyik csomópontra egy új, érvényes tranzakció érkezik, a csomópont továbbküldi azt a vele kapcsolatban lévő szomszédos csomópontoknak. A tranzakciót mindenki szomszéd továbbküldi a vele kapcsolatban lévő csomópontoknak, és így tovább. Ily módon néhány másodperc alatt az érvényes tranzakció egy exponenciálisan bővülő hullámban tovaterjed a hálózatban, amíg minden egyes kapcsolódó csomóponthoz el nem jut.

A bitcoin hálózat úgy lett megtervezve, hogy a tranzakciókat és a blokkokat hatékony módon továbbítsa az összes csomópontnak, és a külső támadásokkal szemben védett legyen. A bitcoin rendszerrel szembeni támadások (pl. spamming, DDoS) kivédése érdekében minden egyik csomópont a többtől függetlenül minden egyes tranzakciót ellenőriz, mielőtt továbbítaná. Egy helytelen formátumú tranzakció egy csomópontnál nem jut tovább. A tranzakciók ellenőrzésére szolgáló szabályokat részletesebben „[A tranzakciók egymástól független ellenőrzése](#)” rész ismerteti.

A tranzakciók szerkezete

A tranzakció egy olyan *adatstruktúra*, amely pénz küldést tesz lehetővé a pénzforrások, vagyis a *bemenetek* és a rendeltetési helyek, vagyis *kimenetek* között. A tranzakció bemeneteinek és kimeneteinek nincs semmi közük sem a számlákhoz vagy a személyazonosságokhoz. Inkább úgy képzeljék el őket, mint bitcoint mennyiségeket, bitcoin darabkákat, melyek egy olyan titokkal lettek zárolva, melyet csak a tulajdonos vagy a titkot ismerő személy tud megszüntetni. A tranzakció számos mezőt tartalmaz, amint azt az 5-1. táblázat mutatja:

5-1. táblázat: Egy tranzakció szerkezete

Méret	Mező	Leírás
4 bajt	Verzió	Megadja, hogy melyek a tranzakció által követett szabályok
1-9 bajt (VarInt)	Input Counter	A tranzakció bemeneteinek a száma
Változó	Inputs	Egy vagy több tranzakció bemenet
1-9 bajt (VarInt)	Output Counter	A tranzakció kimeneteinek a száma
Változó	Outputs	Egy vagy több tranzakció kimenet
4 bajt	Locktime	Unix időbélyeg vagy blokk szám

Tranzakció zárolási idő

A Locktime (Zárolási idő) definiálja, hogy a tranzakció legkorábban mikor adható a blokklánchoz. A referencia kliensben nLockTime a neve. A legtöbb tranzakcióban 0 az értéke, ami az azonnali végrehajtásnak felel meg. Ha a Locktime nem nulla, és 500 millió alatti szám, akkor blokk magasságként van értelmezve, és azt jelenti, hogy a tranzakció nem érvényes, és a megadott blokk magasság elérése előtt nem kerül továbbításra ill. nem kerül be a blokkláncba. Ha az érték 500 millió feletti, akkor Unix időbélyeget (az 1970. jan. 1. óta eltelt másodpercek számát) jelenti, és a tranzakció a megadott idő előtt nem érvényes. A zárolási idővel rendelkező tranzakciókat, melyekben jövőbe mutató idő vagy jövőbeli blokk szerepel, az őket létrehozó rendszerben kell megőrizni, és csak akkor szabad a bitcoin hálózatba továbbítani, ha már érvényesekké váltak. A zárolási idő megfelel egy papír alapú csekk antedatálásának.

Tranzakció kimenetek és bemenetek

Egy bitcoin tranzakció alapvető építő eleme az *elköltetlen tranzakció kimenet* vagy UTXO (unspent transaction output). Az UTXO a bitcoin oszthatatlan darabja, amely egy adott tulajdonoshoz van kötve, szerepel a blokkláncban, és az egész hálózat által elismert pénzegység. A bitcoin hálózat az összes rendelkezésre álló (elköltetlen) UTXO-t nyomon követi. Ezek száma jelenleg a milliós tartományban van. Ha a felhasználónak bitcoint küld valaki, az összeg nagysága a blokklánon belül UTXO-ként van rögzítve. Így aztán egy felhasználó bitcoinjai UTXO-k formájában tranzakciók százai és blokkok százai között lehetnek szétszórva. Igazából egy bitcoin cím egyenlege vagy egy számla egyenlege önállóan nem is létezik, csak szétszórt UTXO-k vannak, melyek egy adott felhasználóhoz vannak kötve. A felhasználó bitcoin egyenlege egy olyan fogalom, amely a pénztárca szintjén jelenik meg. A felhasználó egyenlegét a pénztárca számítja ki oly módon, hogy végigpásztázza a blokkláncot és összegzi az adott felhasználóhoz tartozó összes UTXO-t.



A bitcoinban nincsenek számlák ill. egyenlegek, csak *elköltetlen tranzakció kimenetek* (UTXO-k) vannak, melyek szét vannak szórva a blokkláncban.

Az UTXO-kat Satoshiban mérjük, és az értékük tetszőleges lehet. A dollár esetében a legkisebb egység a két tizedesjeggyel ábrázolható cent. Hasonló módon a bitcoinnál a legkisebb egység a nyolc tizedesjeggyel ábrázolható Satoshi. Noha egy UTXO értéke tetszőleges lehet, a létrejötte után már épp úgy oszthatatlan, mint egy érme, amely nem vágható ketté. Ha az UTXO nagyobb, mint a tranzakció kívánt értéke, akkor is teljes mértékben el kell költeni, és a tranzakcióban visszajáró pénzt kell generálni. Más szóval, ha van egy 20 bitcoinos UTXO-nk, és 1 bitcoin szeretnénk kifizetni, akkor a tranzakciónknak az egész 20 bitcoinos UTXO-t el kell költenie, és két kimenetet kell létrehozna: az első kimenet 1 bitcoin fizet a kívánt címzettnek, a második a visszajáró 19 bitcoin uralja a saját pénztárcánka. Emiatt a bitcoin tranzakcióknak legtöbbször a visszajáró pénzt is kezelniük kell.

Képzeljünk el egy vásárlót, aki 1.50 \$-ért vesz valamilyen italt, benyúl a pénztárcájába, és megpróbálja érmékkal és bankjegyekkel kifizetni az 1.50 \$-os összeget. Fizethet egy egy dolláros bankjeggyel és két negyeddolláros érmével, vagy aprópénzzel (6 db negyeddollárossal), vagy akár egy nagyobb címletű bankjeggyel is (pl. egy 5 dolláros bankjeggyel). Ha a vásárló egy nagyobb címletű bankjeggyel, pl. egy 5 dollárossal fizet, akkor 3.50 \$ visszajár, ezt elteszi a pénztárcájába, és jövőbeli tranzakciókban tudja felhasználni.

Hasonlóképpen, egy bitcoin tranzakció is a felhasználónál rendelkezésre álló, különféle címletű UTXO-kból jön létre. A tranzakció nem tudja az UTXO-kat félbe vágni, mint ahogyan egy dolláros bankjegy sem vágható félbe. A felhasználó pénztárca alkalmazása a felhasználó számára rendelkezésre álló UTXO-k közül általában úgy válogatja össze a különböző értékeket, hogy azok a kívánt tranzakció összegénél nagyobb vagy egyenlő összeget eredményezzenek.

A valós élethez hasonlóan a bitcoin alkalmazás is különféle módszereket használhat a vásárlás összegének kifizetéséhez: használhat több kisebb egységet, és a segítségükkel pontosan

megadhatja a kívánt összeget, vagy használhat egy, a tranzakció összegénél nagyobb egységet, és ilyenkor pénzt kap vissza. Az UTXO-k kezelésének bonyolult műveletét a pénztárca automatikusan végzi, a felhasználók ezt észre sem veszik. Csak akkor van ennek jelentősége, ha az UTXO-kból egy programmal állítunk elő egy tranzakciót.

A tranzakció által elfogyasztott UTXO-kat a tranzakció bemeneteinek, míg a tranzakció által létrehozott UTXO-kat a tranzakció kimeneteinek nevezzük. Ily módon bitcoin érték-darabkák vándorolnak tulajdonosról tulajdonosra a tranzakciós láncon, ennek során UTXO-k semmisülnek meg és UTXO-k jönnek létre. A tranzakciók úgy fogyasztanak el egy adott UTXO-t, hogy az adott tulajdonos az aláírásával felszabadítja azt a zárolás alól, és úgy hoznak létre egy új UTXO-t, hogy azt az új tulajdonos bitcoin címéhez kötik.

A bemeneti és kimeneti láncból kilóg egy speciális tranzakciótípus, az ún. *coinbase* tranzakció, amely mindegyik blokkban az első tranzakció. Ezt a tranzakciót a „nyertes” bányász helyezi el a blokkban. Ez a tranzakció vadonatúj bitcoinokat hoz létre, melyek a nyertes bányásznak fizetendők ki, jutalmul a bányászatért. A bitcoinban így jön létre az új pénz a bányászat során, amint azt a [8. fejezetben](#) látni fogjuk.



Mi volt előbb? A bemenetek vagy a kimenetek, a tyúk vagy a tojás? Tulajdonképpen a kimenetek voltak előbb, mert a coinbase tranzakcióknak, melyek új bitcoinokat állítanak elő, nincsenek bemenetei és kimeneteket hoznak létre a semmiből.

A tranzakció kimenetei

Minden bitcoin tranzakció kimeneteket hoz létre, ezeket a bitcoin főkönyv örökíti meg. Ezen kimenetek szinte mindenike, egy típus kivételével (lásd „[Adat kimenet \(OP_RETURN\)](#)”) elkölhető bitcoin darabokat hoz létre, melyeket *elkölhetetlen tranzakció kimeneteknek*, vagy UTXO-nak hívunk. Az UTXO-kat az egész hálózat általánosan elfogadja, és egy jövőbeli tranzakcióban elkölthetők. Valakinek bitcoint küldeni egyenértékű azzal, hogy olyan elköltetlen tranzakció kimenetet (UTXO-t) hozunk létre, amely a címzett bitcoin címéhez tartozik, és a címzett tudja elkölni.

Az UTXO-kat mindenike teljes bitcoin kliens a memóriában tartja, egy *UTXO poolnak* nevezett adatbázisban. Az új tranzakciók az UTXO poolból fogyasztanak (költenek) el egy vagy több kimenetet.

A tranzakció kimenetek két részből állnak:

- egy bitcoin összegből, amely a legkisebb bitcoin mértékegységen, *Satoshiban* van megadva
- Egy *zárolást végző scriptből*, másképpen „akadályból”, amely oly módon „zárolja” ezt az összeget, hogy megadja, mely feltételeknek kell teljesülnie a kimenet elköltéséhez

A tranzakció script nyelvét, melyet a feljebb említett zároló script használ, részletesen „[A tranzakciós scriptek és a script nyelv](#)” rész tárgyalja. Az 5-2. táblázat a tranzakció kimenetének felépítését mutatja.

5-2. táblázat: Egy tranzakciós kimenet szerkezete

Méret	Mező	Leírás
8 bajt	Összeg	Bitcoin érték Satoshiban (10^{-8} bitcoinban)
1-9 bajt (VarInt)	Zároló script mérete	A zároló script hossza bajtokban, e nélkül a szám nélkül
Változó	Zároló script	Egy script, amely a kimenet elköltéséhez szükséges feltételeket definiálja

Az 5-1. példában a blockchain.info API-val keressük meg egy adott cím elköltetlen kimeneteit (UTXO).

5-1. példa: A blockchain.info API-t hívó script, mely egy cím UTXO-it keresi meg

```
$ cat get-utxo.py
# get unspent outputs from blockchain API

import json
import requests

# example address
address = '1Dorian4RoXcnBv9hnQ4Y2C1an6NJ4UrjX'

# The API URL is https://blockchain.info/unspent?active=<address>
# It returns a JSON object with a list "unspent_outputs", containing UTXO,
like this:
#{      "unspent_outputs": [
#    {
#      "tx_hash": "ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad9084062167",
#      "tx_index": 51919767,
#      "tx_output_n": 1,
#      "script": "76a9148c7e252f8d64b0b6e313985915110fcfefcf4a2d88ac",
#      "value": 8000000,
#      "value_hex": "7a1200",
#      "confirmations": 28691
#    },
#    ...
#  ]
#}

resp = requests.get('https://blockchain.info/unspent?active=%s' % address)
utxo_set = json.loads(resp.text)["unspent_outputs"]

for utxo in utxo_set:
    print "%s:%d - %ld Satoshis" % (utxo['tx_hash'], utxo['tx_output_n'],
utxo['value'])
```

A script futtatása egy listát állít elő. A lista sorai a tranzakciók azonosítóját, az elköltetlen tranzakciós kimenet (UTXO) és az UTXO Satoshiban megadott értékét tartalmazzák. A zároló script ebben az 5-2. példában nem szerepel.

5-2. példa: A get-utxo.py script futtatása

```
$ python get-utxo.py
ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad9084062167:1 -
8000000 Satoshi
6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ecf:0 -
16050000 Satoshi
74d788804e2aae10891d72753d1520da1206e6f4f20481cc1555b7f2cb44aca0:0 -
5000000 Satoshi
b2affea89ff82557c60d635a2a3137b8f88f12ecec85082f7d0a1f82ee203ac4:0 -
10000000 Satoshi
...
```

Költési feltételek (akadályok)

A tranzakció kimenetek egy (Satoshiban) megadott összeget egy adott *akadállyal*, vagy zároló scripttel hoznak kapcsolatba. Ez a zároló script adja meg, hogy milyen feltételeknek kell teljesülniük az összeg elköltéséhez. A legtöbb esetben a zároló script a kimenetet egy adott bitcoin címhez köti, ezáltal az összeg tulajdonjogát egy új felhasználóhoz rendeli hozzá. Mikor Alíz kifizette a csésze kávéját, Alíz tranzakciója egy 0.015 bitcoinos kimenetet hozott létre, amely a kávéház bitcoin címéhez volt hozzákötve, vagyis ez volt az *akadály*. A 0.015 bitcoinos kimenet a blokkláncon került rögzítésre, és az elköltetlen tranzakció kimenetek (UTXO) halmazának részévé vált, vagyis Bob pénztárcájában a rendelkezésre álló egyenleg része lett. Ha Bob szeretné elkölni ezt az összeget, akkor az általa létrehozott tranzakció eltávolítja az akadályt, vagyis megszünteti a kimenet zárolását. Ezt oly módon teszi, hogy létrehoz egy scriptet, amely tartalmaz egy aláírást Bob titkos kulcsával.

A tranzakció bemenetei

A tranzakció bemenetei csupán mutatók az UTXO-kra. Egy bemenet úgy mutat egy adott UTXO-ra, hogy megadja a tranzakció hasht és egy sorszámot, amely megmutatja, hogy az UTXO hányadik a tranzakció kimenetek között. A tranzakció bemenete tartalmaz továbbá egy zárolást feloldó scriptet, amely teljesíti az UTXO-ban meghatározott feltételeket és amellyel az UTXO elkölhető. A zárolást feloldó script általában egy aláírás, amely annak a bitcoin címnek a tulajdonjogát bizonyítja, amely a zárolási scriptben szerepel.

Ha a felhasználó fizetni szeretne, akkor a pénztárcája a rendelkezésre álló UTXO-kból állít össze egy tranzakciót. Például 0.015 bitcoin kifizetéséhez a pénztárca választhat egy 0.01 bitcoin értékű UTXO-t és egy 0.005 bitcoin értékű UTXO-t, mert e kettő együtt éppen a kívánt összeget eredményezi.

A lenti 5-3. példa egy „mohó” algoritmust használ arra, hogy a rendelkezésre álló UTXO-kból a megkívánt összeget előállítsa. A példában a rendelkezésre álló UTXO-k egy konstans tömbben vannak megadva, de a valóságban a rendelkezésre álló UTXO-ket RPC hívással a Bitcoin Core-ból vagy egy harmadik fél által szállított API segítségével kérdezik le, amint azt 5-1. példa mutatja.

5-3. példa: Egy script, amellyel UTXO-k választhatók ki

```
$ cat select-utxo.py
# Selects outputs from a UTXO list using a greedy algorithm.

from sys import argv

class OutputInfo:

    def __init__(self, tx_hash, tx_index, value):
        self.tx_hash = tx_hash
        self.tx_index = tx_index
        self.value = value

    def __repr__(self):
        return "<%s:%s with %s Satoshi>" % (self.tx_hash, self.tx_index,
                                                self.value)

# Select optimal outputs for a send from unspent outputs list.
# Returns output list and remaining change to be sent to
# a change address.
def select_outputs_greedy(unspent, min_value):
    # Fail if empty.
    if not unspent:
        return None
    # Partition into 2 lists.
    lessers = [utxo for utxo in unspent if utxo.value < min_value]
    greater = [utxo for utxo in unspent if utxo.value >= min_value]
    key_func = lambda utxo: utxo.value
    if greater:
        # Not-empty. Find the smallest greater.
        min_greater = min(greater)
        change = min_greater.value - min_value
        return [min_greater], change
    # Not found in greater. Try several lessers instead.
    # Rearrange them from biggest to smallest. We want to use the least
    # amount of inputs as possible.
    lessers.sort(key=key_func, reverse=True)
    result = []
    accum = 0
    for utxo in lessers:
        result.append(utxo)
        accum += utxo.value
        if accum >= min_value:
            change = accum - min_value
            return result, "Change: %d Satoshi" % change
    # No results found.
    return None, 0

def main():
    unspent = [
        OutputInfo("ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad908406216
7", 1, 8000000),
        OutputInfo("6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ec
f", 0, 16050000),
        OutputInfo("b2affea89ff82557c60d635a2a3137b8f88f12ecec85082f7d0a1f82ee203ac
4", 0, 10000000),
```

```

OutputInfo("7dbc497969c7475e45d952c4a872e213fb15d45e5cd3473c386a71a1b0c136a
1", 0, 25000000),
OutputInfo("55ea01bd7e9af3d3ab9790199e777d62a0709cf0725e80a7350fdb22d7b8ec
6", 17, 5470541),
OutputInfo("12b6a7934c1df821945ee9ee3b3326d07ca7a65fd6416ea44ce8c3db0c078c6
4", 0, 10000000),
OutputInfo("7f42eda67921ee92eae5f79bd37c68c9cb859b899ce70dba68c48338857b781
8", 0, 16100000),
]

if len(argv) > 1:
    target = long(argv[1])
else:
    target = 55000000

print "For transaction amount %d Satoshis (%f bitcoin) use: " %
(target, target/10.0**8)
print select_outputs_greedy(unspent, target)

if __name__ == "__main__":
    main()

```

Ha paraméter nélkül futtatjuk a *select-utxo.py* scriptet, akkor a script egy 55'000'000 Satoshi (0.55 bitcoin) nagyságú fizetséghez próbálja meg előállítani az UTXO halmazt (és a visszajáró pénzt). Ha paraméterként megadjuk a cél összeget, a script annyi UTXO-t választ ki, amennyi fedezíti a cél összeget. Lent a script futtatásával 0.5 bitcoin (azaz 50'000'000 Satoshi) kifizetését kíséreltük meg:

5-4. példa: A *select-utxo.py* script futtatása

```

$ python select-utxo.py 50000000
For transaction amount 50000000 Satoshis (0.500000 bitcoin) use:
([<7dbc497969c7475e45d952c4a872e213fb15d45e5cd3473c386a71a1b0c136a1:0 with
25000000 Satoshi>,
<7f42eda67921ee92eae5f79bd37c68c9cb859b899ce70dba68c48338857b7818:0 with
16100000 Satoshi>,
<6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ecf:0 with
16050000 Satoshi>], 'Change: 7150000 Satoshi')

```

Az UTXO-k kiválasztását követően a pénztárca előállítja az egyes UTXO-khoz az aláírt zárolást feloldó scripteket, ami elkölthetővé teszik őket, hiszen így már kielégülnek a zároló script által meghatározott feltételek. A pénztárca ezeket az UTXO hivatkozásokat és zárolást feloldó scripteket a tranzakció bemeneteihez adja hozzá. Az 5-3. táblázat egy tranzakció bemenet szerkezetét mutatja.

5-3. táblázat: Egy tranzakció bemenet szerkezete

Méret	Mező	Leírás
32 bájt	Tranzakció hash	Mutató arra a tranzakcióra, amely az elköltendő UTXO-t tartalmazza
4 bájt	Output Index	Az elköltendő UTXO indexe, az első 0
1-9 bájt (VarInt)	A zárolást feloldó script mérete	A zárolást feloldó script mérete bájtokban
Változó	A zárolást feloldó script	Az UTXO-t zároló script feltételeit kielégítő script
4 bájt	Sorszám	Tx-helyettesítő lehetőség, Jelenleg letiltva, 0xFFFFFFFF



A sorszámmal a tranzakció a zárolási idő lejárta előtt módosítható, de ez jelenleg le van tiltva a bitcoinban. A legtöbb tranzakció a max. egész értékre (0xFFFFFFFF) állítja ezt az értéket, amit a bitcoin hálózat elhanyagol. Ha a tranzakció zárolási ideje nem nulla, akkor a zárolási idő csak akkor jut érvényre, ha a bemeneti közül legalább az egyiknél a sorszám 0xFFFFFFFF alatt van.

Tranzakciós díjak

A legtöbb tranzakció tranzakciós díjat tartalmaz, amely a bitcoin bányászokat jutalmazza a hálózati biztonság megteremtéséért. A bányászat, a tranzakciós díjak és a bányászok által kapott jutalmak a [8. fejezetben](#) vannak részletesen tárgyalva. Ebben a részben azt vizsgáljuk meg, hogyan kerül tranzakciós díj egy tipikus tranzakcióba. A legtöbb pénztárca automatikusan kiszámítja és befoglalja a tranzakciós díjakat. Ha azonban programból állítjuk elő a tranzakciókat, vagy egy parancssori felületet használunk, akkor kézzel kell kiszámítani és alkalmazni ezeket a díjakat.

A tranzakciós díj – azáltal, hogy minden tranzakcióra egy kis költséget ró ki – azt ösztönzi, hogy a tranzakció befoglalásra kerüljön a következő blokkba, és véd a „spam” tranzakciókkal szemben, melyek visszaélnek a rendszerrel. A tranzakciós díjat az a bányász kapja meg, aki kibányássza a tranzakciót tartalmazó blokkot, és ily módon a tranzakció bekerül a blokkláncba.

A tranzakciós díj nem a tranzakció bitcoinban mért nagyságától, hanem a tranzakció kilobájtokban mért méretétől függ. Összefoglalva, a tranzakciós díjak a bitcoin hálózaton belüli piaci hatások alapján határozhatók meg. A bányászok különféle szempontok alapján állítják sorba a tranzakciókat, pl. a tranzakciós díj alapján, de bizonyos körülmények között akár ingyen is feldolgozzák őket. A tranzakciós díj a feldolgozási prioritást is befolyásolja, vagyis egy megfelelő tranzakciós díjjal rendelkező tranzakció nagyobb valószínűséggel kerül be a következőnek kibányászott blokkba, míg egy kevesebb vagy nulla tranzakciós díjjal rendelkező tranzakció késedelmet szenvedhet, és csak pár blokkal később történik meg a feldolgozása, vagy egyáltalán nem kerül feldolgozásra. A tranzakciós díj nem kötelező, és végső soron a tranzakciós díj nélküli tranzakciók is feldolgozásra kerülhetnek, de a tranzakciós díj megadása elősegíti a gyors feldolgozást.

A tranzakciós díjak kiszámítási módja és a tranzakció prioritására gyakorolt hatásuk nem minden volt olyan, mint most. Először a tranzakciós díj fix összeg volt az egész hálózatban.

Fokozatosan lazítottak a díjstruktúrán, hogy a díjat a hálózati kapacitás és a tranzakciók száma alapján a piaci erők is befolyásolhassák. A jelenlegi legkisebb tranzakciós díj kilobájtonként 0.0001 bitcoin, vagy másnéven egytized millibitcoin, és nemrég csökkentették le egy millibitcoinról. A legtöbb tranzakció egy kilobájtnál kisebb méretű, de azok, amelyeknek sok bemenetük és kimenetük van, nagyobbak is lehetnek. A bitcoin protokoll jövőbeli változatainál a pénztárca alkalmazás várhatóan a korábbi tranzakciók átlagos díja alapján, statisztikai elemzéssel fogja kiszámítani a legmegfelelőbb tranzakciós díjat.

A bányászok által jelenleg használt algoritmust, amely a tranzakciós díj alapján priorizálja a tranzakciók blokkba foglalását, részletesen a [8. fejezetben](#) fogjuk megvizsgálni.

A tranzakciós díj megadása

A tranzakciók adatstruktúrájában nincs díj mező. A díjak hallgatólagosan a bemenetek összegének és a kimenetek összegének különbségével egyenlők. Az összes kimenetnek az összes bemenetből történő levonása után megmaradó összeg a bányászoké lesz.

A tranzakciós díj hallgatólagos, a bemenetek és a kimenetek különbsége utáni maradék

$$\text{Díj} = \text{Összeg (Bemenetek)} - \text{Összeg (Kimenetek)}$$

Ez a tranzakciók kissé zavarba ejtő jellemzője, de fontos megérteni, mert ha mi magunk állítjuk elő a tranzakcióinkat, akkor vigyáznunk kell arra, nehogy nagyon nagy legyen a díj, mert a bemenetkből nem költünk el eleget. Ez azt jelenti, hogy figyelembe kell vennünk az összes bemenetet, és ha szükséges, akkor a visszajáró pénzt is kezelnünk kell, különben a bányászok nagyon nagy borralót kapnak a végén!

Például, ha egy 20 bitcoin értékű UTXO-t használunk egy 1 bitcoinos fizetséghoz, akkor egy 19 bitcoin értékű kimenetet kell létrehoznunk a visszajáró pénznek. Ha nem így teszünk, akkor a „maradékot” tranzakciós díjnak fogják tekinteni, és azé a bányászé lesz, aki a tranzakciót blokkba fogalta. Igaz ugyan, hogy sürgősségi feldolgozásban lesz részünk, és egy bányászt nagyon boldoggá teszünk, de nem biztos, hogy ezt szerettük volna.



Ha egy kézzel előállított tranzakcióban elfelejtünk a visszajáró pénznek egy kimenetet létrehozni, akkor a visszajáró pénz teljes egészében a tranzakciós díjat fogja növelni. „Tartsa meg a visszajáró pénzt!” – nem biztos, hogy ez volt a szándékunk.

Nézzük meg, hogyan működik minden a gyakorlatban, ismét Alíz kávévásárlását vizsgálva. Alíz 0.015 bitcoint szeretne elkölni, hogy kifizesse a kávéját. Szeretné, ha a tranzakciója gyorsan feldolgozásra kerülne, ezért tranzakciós díjat is megad, mondjuk 0.001 bitcoin. Ez azt jelenti, hogy a tranzakció teljes költsége 0.016 bitcoin. A pénztárcájában lévő UTXO halmoz összegének ezért 0.016 bitcoinnak vagy nagyobbnak kell lennie, és ha szükséges, kezelni kell a visszajáró pénzét. Mondjuk, legyen a pénztárcában 0.2 bitcoin UTXO. Ennek az UTXO-nak a felhasználásával létre kell hozni egy 0.015 BTC-s kimenetet Bob kávéháza számára, és egy második kimenetet 0.184 bitcoinnal, amely ezt a visszajáró pénzt Alíz saját pénztárcájába utalja vissza. Ily módon 0.001 bitcoin marad, vagyis ez lesz a tranzakció implicit díja.

Most vizsgáljunk meg egy ettől eltérő helyzetet. Eugénia, a gyermekvédelmi alap igazgatója gyűjtést szervezett, hogy a Fülöp-szigeti gyerekeknek tankönyveket vásárolhasson. Sok ezer kicsiny adományt kapott szerte a nagyvilágból, összesen 50 bitcoint. Most szeretne pár száz tankönyvet venni a helyi kiadótól, és bitcoinnal szeretne fizetni.

Eugénia pénztárca programjának a sok ezer piciny adományból kell egy nagyobb kifizetést létrehoznia, vagyis a piciny összegeket tartalmazó UTXO-kból kell a fedezetet biztosítania. Ez azt jelenti, hogy az eredményként létrejövő tranzakciónak száznál is több kis értékű UTXO-t tartalmazó bemenete lesz, de csak egyetlen egy kimenete, amellyel a könyvkiadónak fizet. Az ilyen sok bemenetet tartalmazó tranzakció nagyobb lesz egy kilobájtnál, akár 2-3 kilobájt is lehet. Emiatt a 0.0001 bitcoin minimális hálózati díjnál nagyobb díjra lesz szükség.

Eugénia pénztárca programja úgy számítja ki a megfelelő díjat, hogy összeszorozza a a tranzakció méretét a kilobájtonkénti díjjal. Sok pénztárca a nagyobb méretű tranzakciók esetén túlfizeti a díjat, hogy biztosítsa a tranzakció gyors feldolgozását. A nagyobb díjat nem azért kell megfizetni, mert Eugénia több pénzt költ, hanem azért, mert a tranzakció bonyolultabb és nagyobb méretű – a díj független attól, hogy a tranzakcióban mekkora érték szerepel.

Tranzakciós láncok, árva tranzakciók

Mint láttuk, a tranzakciók egy láncot alkotnak, ahol egy tranzakció az előző tranzakciók (az ún. szülők) kimeneteit költi el, és kimeneteket hoz létre egy további tranzakció (az ún. gyermek) számára. Néha a függőségekből egy egész tranzakciós lánc alakul ki, pl. ha egy szülő, gyermek és unoka egy bonyolult tranzakciós munkafolyamat során ugyanakkor jön létre, és követelmény, hogy a gyerekek előbb legyenek aláírva, mint a szülő. Például a CoinJoin tranzakciók ezzel a módszerrel egyesítik több ügyfél tranzakciójait, hogy fokozzák a tranzakciók titkosságát.

Ha egy tranzakciós lánc kerül továbbításra a hálózaton, akkor a tranzakciók nem minden az eredeti sorrendben érkeznek meg. Néha a gyerek a szülő előtt érkezik meg. Ebben az esetben azok a csomópontok, melyek a gyermeket látják először, látják, hogy a tranzakció egy olyan szülőre hivatkozik, amely még ismeretlen. De nem vetik el a gyereket, hanem egy átmeneti halmazba teszik, ahol várakozhat a szülő megérkezésére, és továbbítják a többi csomópontról. A szülő nélküli tranzakciók halmazának a neve: az *árva tranzakciók poolja* (*halmaza*). Ha megérkezik a szülő tranzakció, akkor azok a gyerekek, melyek a szülő által létrehozott UTXO-ra hivatkoznak, kikerülnek a listából, rekurzív módon ismét ellenőrzésre kerülnek, és azután az egész tranzakciós lánc bekerül a kibányászható tranzakciók készletbe. A tranzakciós láncok tetszőleges hosszúak lehetnek, és egymással párhuzamosan tetszőleges számú generáció továbbítható a hálózaton. Az a mechanizmus, amely az árvákat az árva tranzakciós halmazában tartja, biztosítja, hogy az egyébként érvényes tranzakciók ne legyenek elvetve csak azért, mert a szülők késve érkeztek meg. Végül az a lánc, amelyhez tartoznak, a helyes sorrendben helyreáll, függetlenül az érkezés sorrendjétől.

A memóriában tárolható árva tranzakciók számára van egy felső határ, hogy ne lehessen ily módon DoS támadást indítani a bitcoin csomópontok ellen. A korlátot a bitcoin referencia kliens forráskódjában a `MAX_ORPHAN_TRANSACTIONS` definiálja. Ha az árva tranzakciók száma meghaladja a `MAX_ORPHAN_TRANSACTIONS`-t, akkor egy vagy több véletlenszerűen kiválasztott árva tranzakció eltávolításra kerül a készletből, mindaddig, amíg a pool mérete a korláton belülre nem kerül.

A tranzakciós scriptek és a script nyelv

A bitcoin kliensek egy script végrehajtásával ellenőrzik a tranzakciók helyességét. A script egy Forth-szerű script nyelven van írva. Mind az UTXO-ra helyezett zároló script (akadály), mind a zárolást feloldó, aláírt script ezen a nyelven van megírva. A tranzakció ellenőrzésekor az egyes bemenetekben szereplő, zárolást feloldó scriptet a hozzá tartozó zároló scripttel együtt futtatják, hogy megállapítsák, vajon kielégíti-e a pénz elköltésének a feltételeit.

Manapság a bitcoin hálózatban feldolgozott legtöbb tranzakció „Alíz fizet Bobnak” alakú, és egy olyan scripten alapul, melyet „fizetség-nyilvános-kulcs-hashnek” script a neve (Pay-to-Public-Key-Hash script). Mivel azonban a kimenetek zárolására és a bemeneteken a zárolás feloldására használt scriptek egy programozási nyelvhez hasonlóak, a tranzakciók számtalan feltételt tartalmazhatnak. A bitcoin tranzakciók nem korlátozódnak az „Alíz fizet Bobnak” típusú és alakú tranzakcióra.

A fenti példa csak a jéghegy csúcsát jelenti a script nyelvvel kifejezhető lehetőségek között. Ebben a részben a bitcoin tranzakciós nyelvének elemeit szemléltetjük, és bemutatjuk, hogyan lehet őket bonyolult feltételek kifejezésére használni, és hogyan lehet ezeket a feltételeket a zárolást feloldó scriptekben kielégíteni.



A bitcoin tranzakciókban az ellenőrzés nem statikus, hanem egy script nyelv végrehajtásával valósul meg. Ez a nyelv szinte végtelen számú feltétel kifejezését teszi lehetővé. A bitcoin ezáltal lesz „programozható pénz”.

Script létrehozása (zárolás + zárolás feloldás)

A bitcoinban a tranzakciók ellenőrzése kétféle script vizsgálatával történik – a zárolást végző és a zárolás feloldó scriptével.

A zároló script a kimenetre helyezett akadály, amely megadja, hogy milyen feltételeket kell teljesíteni a kimenet jövőbeli elköltéséhez. Történetileg a zároló scriptet *scriptPubKey*-nek hívták, mert általában egy nyilvános kulcsot vagy bitcoin címet tartalmazott. Ebben a könyvben „zároló scriptnek” hívjuk, mert jelezni akarjuk a script alkalmazásában rejlö tágabb lehetőségeket. A legtöbb bitcoin alkalmazásban az általunk zároló scriptnek hívott script a forráskódban *scriptPubKey*-ként jelenik meg.

A zárolást feloldó script olyan script, amely „megoldja”, azaz kielégíti azokat a feltételeket, amelyeket a zároló script helyez a kimenetre, és lehetővé teszi a kimenet elköltését. A zárolást feloldó scriptek minden egyes tranzakciós bemenetben szerepelnek, és a legtöbbször egy digitális aláírást tartalmaznak, amelyet a felhasználó pénztárcája állít elő a titkos kulcsból. Történetileg a zárolás feloldó scriptet *scriptSig*-nek hívták, mert általában egy digitális aláírást tartalmazott. Ebben a könyvben „zárolást feloldó scriptnek” hívjuk, ismét csak azért, hogy jelezzük a script írási módszerben rejlö lehetőségeket, hiszen nem minden zárolást feloldó scriptnek kell aláírást tartalmaznia.

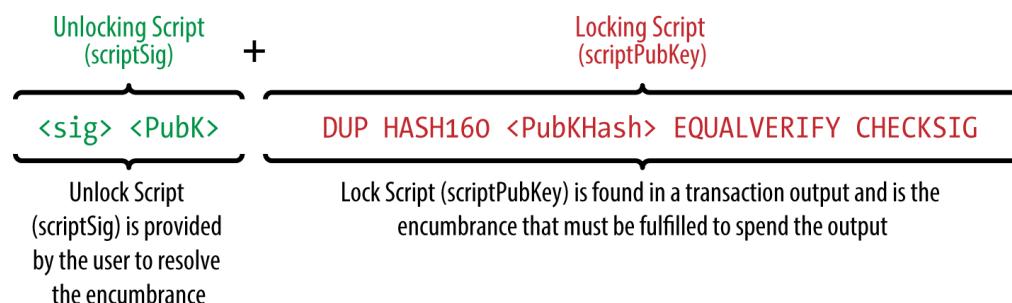
A bitcoin kliensek a tranzakciókat úgy ellenőrzik, hogy a zárolást feloldó és a zároló scripteket együtt hajtják végre. A tranzakció bemeneteire vonatkozóan az ellenőrző program először azokat az UTXO-t keresi meg, melyekre a bemenet hivatkozik. Ez az UTXO egy

zároló scriptet tartalmaz, amely a kimenet elköltéséhez szükséges feltételeket definiálja. Az ellenőrző program ezután veszi a bemenetben szereplő, zárolást feloldó scriptet, amely megkísérli az UTXO elköltését, és végrehajtja a két scriptet.

Az eredeti bitcoin kliensben a zárolást feloldó és a zároló scriptet összefűzte a program, és egymás után hajtotta végre őket. Biztonsági okokból ez 2010-ben megváltozott, mert volt egy támadhatóság, amely egy rosszul formált zárolást feloldó scriptnek megengedte, hogy adatokat tegyen a verembe, és a zároló scriptet tönkretegye. A jelenlegi implementációban a scriptek végrehajtása egymás után történik, és a verem a két végrehajtás között az alábbiaknak megfelelően kerül továbbításra.

Először a zárolást feloldó script kerül végrehajtásra. Ha ez hiba nélkül lefut (pl. nem maradtak „függő” operátorok), akkor a fő veremtár (nem az alternatív) lemásolásra kerül, és zároló script kerül futtatásra. Ha a zárolást végző script eredménye a zárolást feloldó script adataival futtatva „IGAZ”, akkor az zárolást feloldó scriptnek sikerült a zároló script által támasztott feltételeket kielégítenie, vagyis a bemeneten egy érvényes meghatalmazás van az UTXO elköltésére. Ha a kombinált script végrehajtása az „IGAZ”-tól eltérő eredménnyel zárul, akkor a bemenet érvénytelen, mivel nem sikerült kielégítenie az UTXO által támasztott feltételeket. Megjegyzendő, hogy az UTXO a blokkláncban végleges és megváltoztathatatlan formában van tárolva, emiatt egy új tranzakció sikertelen költési kísérletei nem befolyásolják. Csak az UTXO feltételeit helyesen kielégítő, érvényes tranzakció hatására lesz az UTXO „elköltve”, és lesz a rendelkezésre álló (elköltetlen) UTXO-k halmazából eltávolítva.

Az 5-1. ábrán a leggyakrabban előforduló bitcoin tranzakció scriptre (kifizetés egy nyilvános kúles hashnek) látható példa, amely a script ellenőrzése előtti állapotban bemutatja a zárolást feloldó és zároló script összefűzésével előálló teljes scriptet:



5-1. ábra: A scriptSig és scriptPubKey összefűzésével előálló tranzakciós script kiértékelése

Script nyelv

A bitcoin tranzakciós script nyelve, melyet eléggyé zavaró módon szintén *Script*-nek hívnak, egy Forth-szerű, fordított lengyel jelölésnek megfelelő, verem alapú végrehajtási nyelv. Ha ez blablának hangzik, akkor önök valószínűleg nem tanulmányozták az 1960-as évek programozási nyelvezetit. A *Script* egy nagyon egyszerű, pehelysúlyú nyelv, amely korlátozott célokra szolgál, és számos hardver típuson végrehajtható, még olyan egyszerű hardvereken is, mint egy beágyazott eszköz, vagy egy kézi számológép. Minimális feldolgozási igénye van, és sok olyan feladat elvégezhető vele, mint a modern programozási nyelvekkel. A programozható pénz esetében egy tudatos biztonsági megoldásról van szó.

A bitcoin script nyelvét azért hívják verem-alapú nyelvnek, mert egy *veremnek* nevezett adatstruktúrát használ. A verem egy nagyon egyszerű adatszerkezet, melyet úgy lehet elképzelni, mint egy kártyapaklit. A pakli két műveletet tesz lehetővé: ráhelyezést (push) és levételt (pop). Ráhelyezéskor egy újabb téTEL kerül a verem tetejére. A levétel eltávolítja a verem tetején lévő elemet.

A script nyelv úgy hajtja végre a scriptet, hogy balról jobbra minden egyes elemet végrehajt. A számok (adat konstansok) a veremre kerülnek. A műveletek egy vagy több paramétert eltávolítanak a veremről, elvégzik az adott műveletet, majd az eredményt a veremre helyezik vissza. Például az `OP_ADD` két téTEL távolít el a veremről, összeadjá őket, és az eredményként kapott összeget visszahelyezi a veremre.

A feltételes műveletek egy feltétel kiértékelése után IGAZ vagy HAMIS eredményt állítanak elő. Például az `OP_EQUAL` két téTEL távolít el a veremről, és IGAZ értéket tesz a veremre (az IGAZ értéknek az 1 felel meg) ha a két szám egyenlő, és HAMIS értéket (amelyet a nulla ábrázol), ha a két szám nem egyenlő.

Az 5-2. ábrán a `2 3 OP_ADD 5 OP_EQUAL` script az `OP_ADD` összeadási műveletet szemlélteti: összead két számot, az eredményt a veremre helyezi, majd ezt követően egy `OP_EQUAL` feltételes művelettel megvizsgálja, hogy az eredményül kapott összeg egyenlő-e 5-tel. A rövidség kedvéért az `OP_` előtagot a részletes, lépésről-lépésre történő kiértékelésben elhagytuk.

A következő egy kicsit bonyolultabb példa, amely a $2 + 7 - 3 + 1$ kifejezés értékét számítja ki. Figyeljük meg, hogy ha a script számos egymás utáni műveletet tartalmaz, a verem lehetővé teszi, hogy az eredményt a következő művelet felhasználhassa:

```
2 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL
```

Próbálják meg papírral és ceruzával kiértékelni a fenti scriptet. A script kiértékelésének a végén a veremnek az IGAZ értéket kell tartalmaznia.

Noha a legtöbb zároló script egy bitcoin címre vagy egy nyilvános kulcsra hivatkozik, és ennek révén megköveteli a tulajdonjog bizonyítását az összeg elkötése előtt, a scriptnek nem kell ennyire bonyolultnak lennie. A zárolást feloldó és zároló scriptek bármely kombinációja, amely IGAZ eredményt ad, érvényes. Az általunk használt egyszerű számtani példa, melyet a fenti script példában használtunk, szintén érvényes zároló scriptet alkot, amellyel zárolható egy tranzakció kimenete.

Használjuk zároló scriptként a számtani példa következő részét:

```
3 OP_ADD 5 OP_EQUAL
```

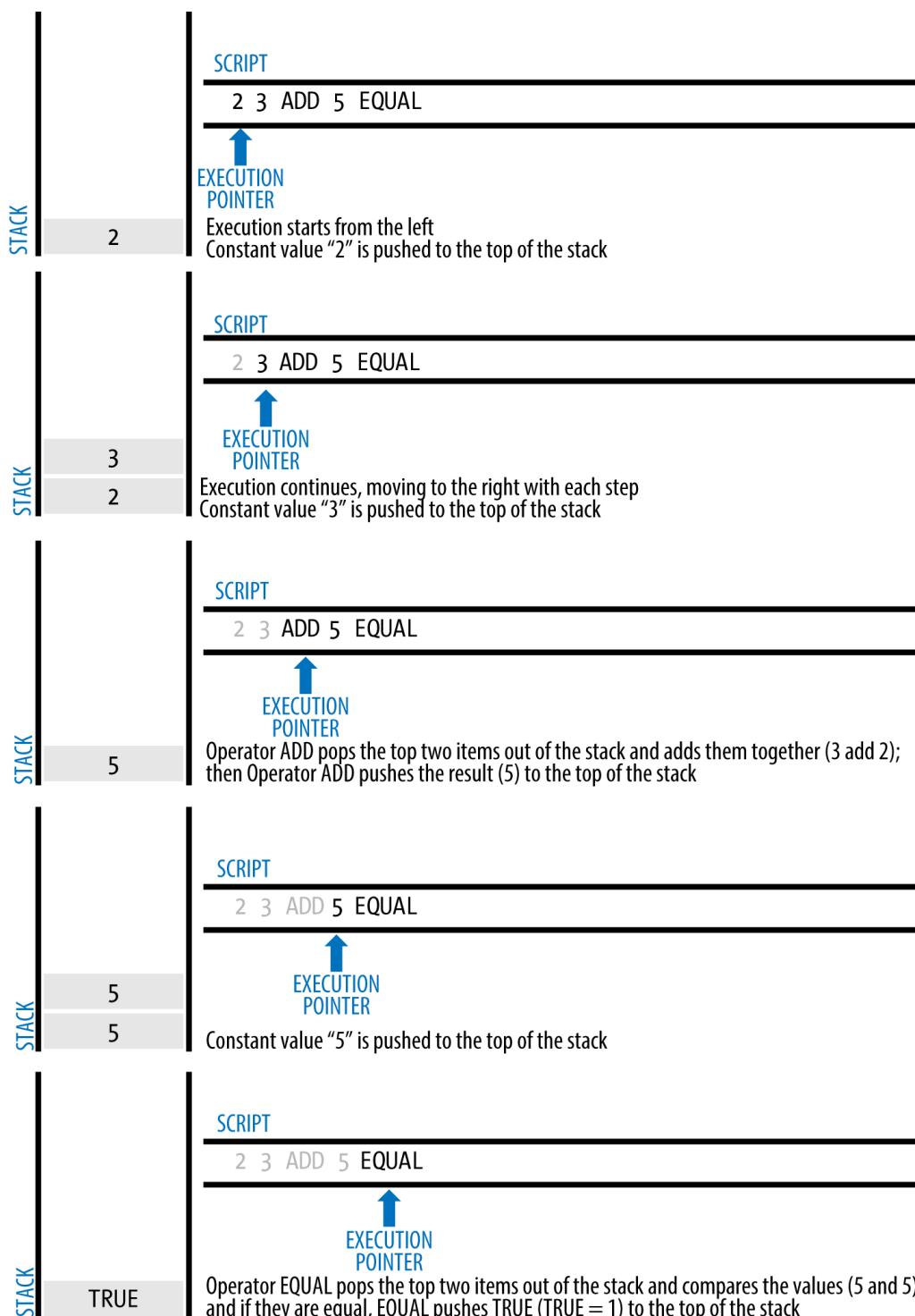
amely egy olyan tranzakcióval elégíthető ki, melynek bemenetén a következő, zárolást feloldó script áll:

2

Az ellenőrző szoftver összekapcsolja a zárolást feloldó és zároló scripteket. Az eredményül kapott script a következő:

2 3 OP_ADD 5 OP_EQUAL

Az 5-2. ábra fenti script lépésről lépésre történő végrehajtását mutatja. A script végrehajtásakor az eredmény IGAZ, vagyis a tranzakció érvényes. Egy érvényes zároló scriptet hoztunk létre, amelynél a létrehozott UTXO elköltésére bárki képes, aki ismeri a számtant, és tudja, hogy a 2 kielégíti a scriptet.



5-2. ábra: A bitcoin script ellenőrző algoritmusára az egyszerű matek példában



Egy tranzakció akkor érvényes, ha a verem tetején lévő eredmény IGAZ (`{0x01}`), vagy bármilyen nem nulla érték, vagy a verem üres a script végrehajtása után. A tranzakció érvénytelen, ha a verem tetején lévő érték HAMIS (egy nulla hosszúságú üres érték, melyet úgy jelölünk, hogy `{}`), vagy a script végrehajtását valamelyik művelet, pl. `OP_VERIFY`, `OP_RETURN` vagy egy feltételes művelet, pl. `OP_ENDIF` leállította. A részletek az [A. függelékben](#) találhatók.

Turing nem teljesség

A bitcoin tranzakciós script nyelve sok műveletet tartalmaz, de egy fontos tekintetben tudatosan korlátozott – nincsenek benne ciklusok vagy a feltételes kifejezéseken kívül más, bonyolultabb vezérlésátadó lehetőségek. Emiatt a nyelv nem *Turing-teljes*, ami azt jelenti, hogy a nyelvnek korlátozott a bonyolultsága és megjósolható a végrehajtási ideje. Ezek a korlátozások biztosítják, hogy a nyelvben a tranzakción belül ne lehessen végtelen ciklust vagy más efféle „logikai bombát” létrehozni, ami a bitcoin rendszer elleni Denial-of-Service (szolgáltatás megtagadási) támadást tenne lehetővé. Emlékeztetünk rá, hogy a bitcoin hálózat mindenkor megakadályozza, hogy ezt az ellenőrzési mechanizmust használja fel valaki támadásra.

Állapotmentes ellenőrzés

A bitcoin tranzakciós script nyelve állapotmentes, mivel a scriptnek a végrehajtás előtt nincs állapota, és a végrehajtása után nem kerül semmilyen állapot sem elmentésre. Emiatt a végrehajtáshoz szükséges összes információ a scriptben van tárolva. A script megjósolható módon, ugyanúgy hajtódkik végre bármelyik rendszerben. Ha a rendszerünk az ellenőrzés során helyesnek találja a scriptet, akkor biztosak lehetünk benne, hogy a bitcoin hálózat bármelyik másik tagja szintén helyesnek fogja találni a scriptet, ami azt jelenti, hogy a tranzakció mindenki számára érvényes, és mindenki tudja ezt. Az eredménynek ez a megjósolhatósága a bitcoin rendszer egyik legfontosabb előnye.

Szabványos tranzakciók

A bitcoin fejlesztés első néhány évében a fejlesztők bizonyos korlátozásokat vezettek be a referencia kliens által feldolgozható script típusok vonatkozásában. Ezek a korlátozások az `isStandard()` függvényben vannak kódolva. A függvény ötféle „szabványos” tranzakciót definiál. Ezek a korlátozások átmenetiek, és lehet, hogy amikor ön e sorokat olvassa, már nem lesznek érvényben. Addig is, a referencia kliens és a legtöbb bányász, amelyik a referencia klienst futtatja, csak az ötféle szabványos tranzakciós scriptet fogadja el. Lehetséges ugyan olyan nem szabványos tranzakció létrehozása, amelyikben a script semelyik szabványos script típusnak sem felel meg, de ha a tranzakciót blokkba szeretnénk foglaltatni, akkor találnunk kell egy bányászt, amelyik nem alkalmazza ezeket a korlátozásokat.

A Bitcoin Core kliens (referencia kliens) forráskódjából állapítható meg, hogy éppen melyek a megengedett tranzakciós scriptek.

Az öt szabványos tranzakciós script típus a következő: (1) Pay-to-Public-Key-Hash (P2PKH), (2) Public Key, (3) Multi-Signature (max. 15 kulcsra korlátozva), (4) Pay-to-Script-Hash (P2SH), és (5) adat kimenet (`OP_RETURN`). Ezeket alább részletesebben ismertetjük.

Fizetés nyilvános kulcs hashnek, (P2PKH, Pay-to-Public-Key-Hash)

A bitcoin hálózatban feldolgozott tranzakciók túlnyomó többsége „Fizetés nyilvános kulcs hashnek” tranzakció, melyet P2PKH tranzakcióként is hívnek. Ezek olyan zároló scriptet tartalmaznak, amely a kimenetet egy nyilvános kulcs hashének értékével zárolja. A nyilvános kulcs hashe nem más, mint a bitcoin cím. Azok a tranzakciók, melyek egy bitcoin címre továbbítják a fizetséget, P2PKH scripteket tartalmaznak. Egy P2PKH scripttel zárolt kimenet zárolása oly módon szüntethető meg, hogy megadjuk a nyilvános kulcsot és a nyilvános kulcschoz tartozó titkos kulccsal egy digitális aláírást.

Például tekintsük ismét Alíz fizetségét. Alíz 0.015 bitcoint fizetett ki a kévéért Bob kávéházának bitcoin címére. A tranzakció kimenetén lévő zároló script a következő formájú:

```
OP_DUP OP_HASH160 <a kávéház nyilvános kulcsának a hash értéke> OP_EQUAL  
OP_CHECKSIG
```

A Kávéház nyilvános kulcsának hashe egyenértékű a kávéház bitcoin címével, de Base58Check kódolás nélkül. A legtöbb alkalmazás a *nyilvános kulcs hashét* hexadecimális kódolással jeleníti meg, nem pedig az ismerős Base58Check formátumban, amely egy „1”-sel kezdődik.

A fenti zároló script a következő formájú, zárolást megszüntető scripttel elégíthető ki:

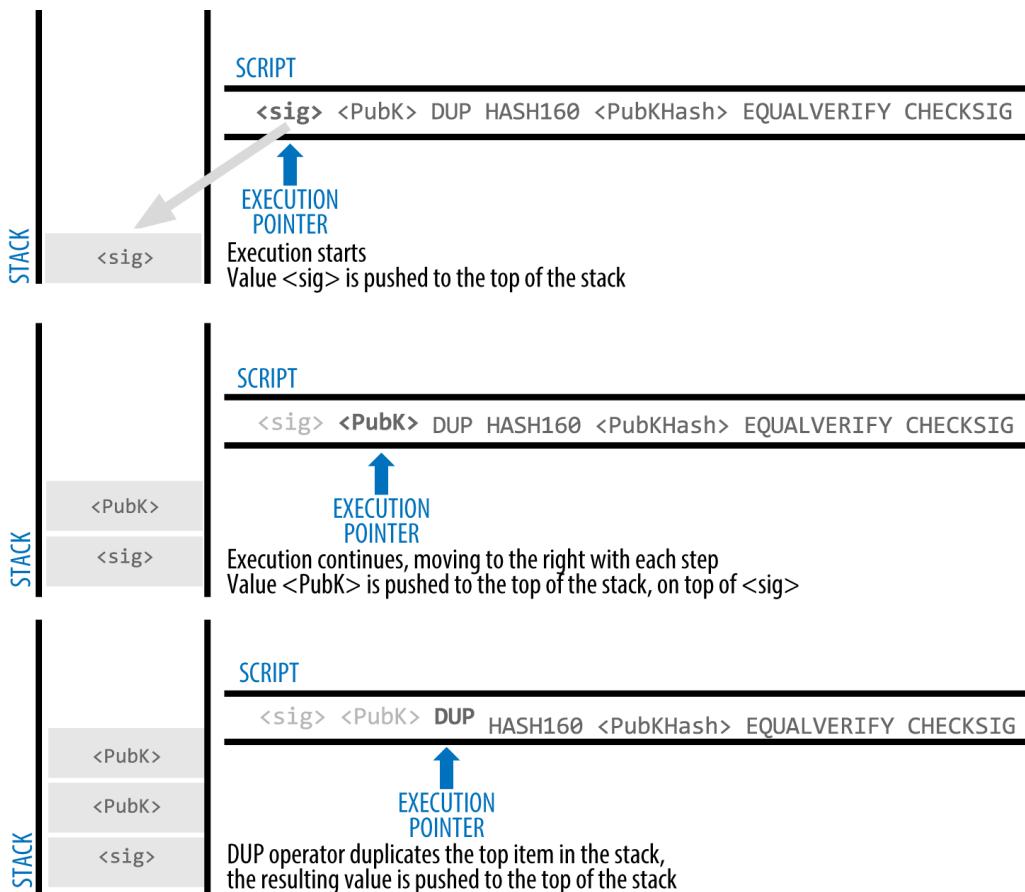
```
<a kávéházhhoz tartozó aláírás> <a kávéház nyilvános kulcsa>
```

A két script együttesen a következő egyesített ellenőrző scriptet alkotja:

```
<a kávéházhhoz tartozó aláírás> <a kávéház nyilvános kulcsa> OP_DUP  
OP_HASH160 <a kávéház nyilvános kulcsának hash értéke> OP_EQUAL OP_CHECKSIG
```

A végrehajtás során az egyesített script akkor és csak akkor lesz IGAZ, ha a zárolást feloldó script megfelel a zároló script által felállított feltételeknek. Más szóval, az eredmény akkor lesz IGAZ, ha a zárolást feloldó scriptben van egy érvényes aláírás a kávéház titkos kulcsával, ami megfelel az akadályként állított nyilvános kulcs hashének.

Az 5-3 és 5-4. ábrák (két részben) az egyesített tranzakció lépésről lépésre történő végrehajtását mutatják, amellyel bizonyítható, hogy érvényes tranzakcióról van szó.



5-3. ábra: Egy P2PKH tranzakció scriptjének kiértékelése (1. rész)

Fizetés nyilvános kulcsnak (Pay-to-Public-Key)

A „fizetés nyilvános kulcsnak” (Pay-to-Public-Key) egy egyszerűbb bitcoin fizetési forma, mint a „fizetés nyilvános kulcs hashnek” (Pay-to-Public-Key-Hash). Ebben a script típusban nem a nyilvános kúlcus hashe, hanem maga a nyilvános kúlcus van a zároló scriptben tárolva. A „fizetés nyilvános kulcs hashnek” (Pay-to-Public-Key-Hash) típust Satoshi találta föl, hogy a bitcoin címek rövidebbek, könnyebben használhatók legyenek. A „fizetés nyilvános kulcsnak” (Pay-to-Public-Key) a leggyakrabban a coinbase tranzakciókban található meg. Ezeket régebbi bányász szoftverek állítják elő, melyeket még nem lettek a P2PKH-ra frissítve.

A „fizetés nyilvános kulcsnak” zároló scriptje így néz ki:

```
<az A nyilvános kulcs> OP_CHECKSIG
```

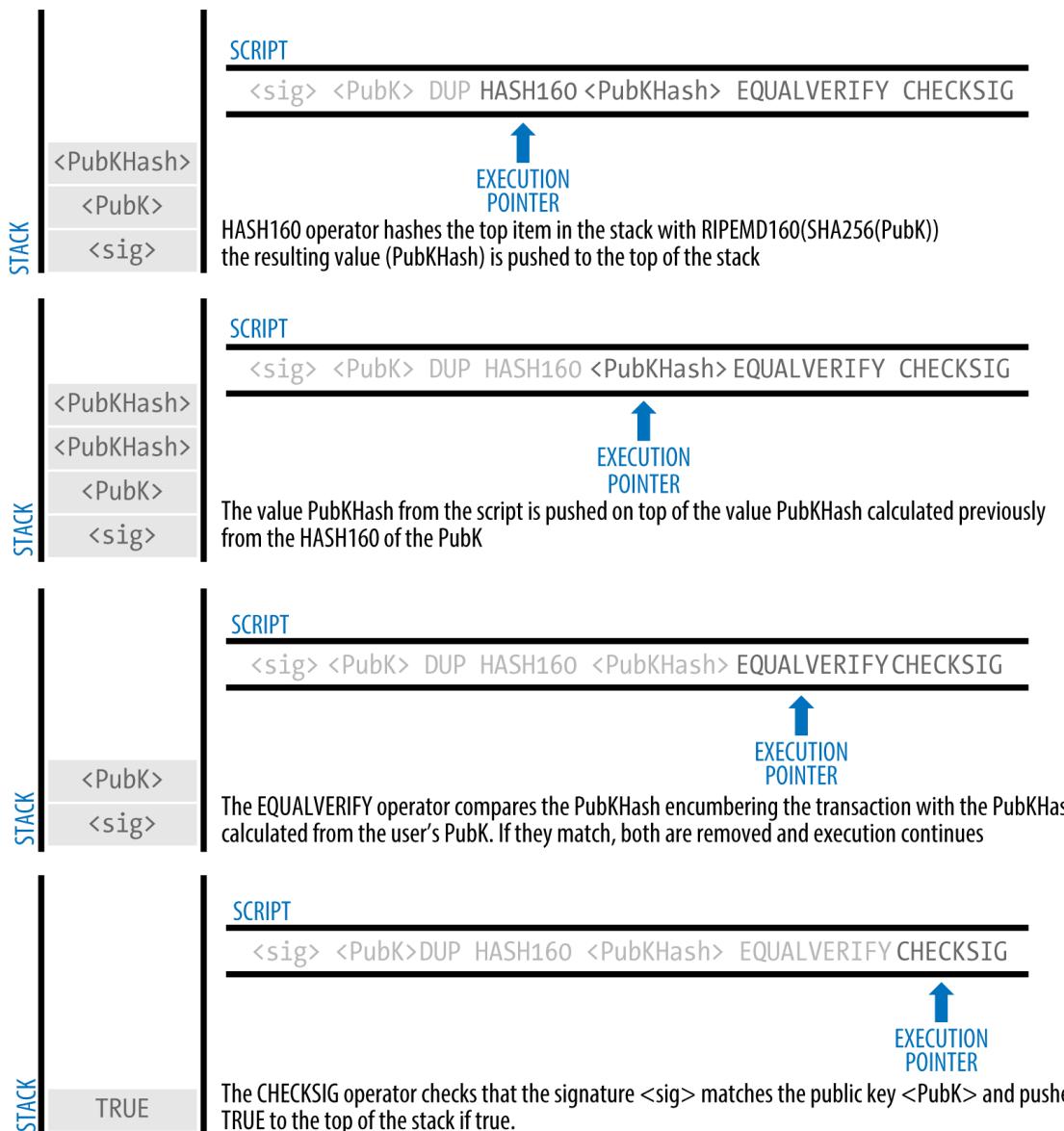
A neki megfelelő zárolást feloldó script, amelyet az ilyen típusú kimenet zárolásának feloldásához kell bemutatni, a következő:

```
<aláírás az A titkos kulccsal>
```

Az egyesített script, melyet a tranzakció ellenőrző program ellenőriz:

```
<aláírás az A titkos kulccsal> <az A nyilvános kulcs> OP_CHECKSIG
```

A fenti script egyszerűen meghívja a CHECKSIG műveletet. Ez a művelet ellenőrzi, hogy az aláírás a megadott kulcsra tartozik-e. Ha igen, akkor IGAZ értéket ad vissza a vermen.



5-4. ábra: Egy P2PKH tranzakció scriptjének kiértékelése (2. rész)

Többszörös aláírás (Multi-Signature)

A több aláírást megkövetelő scriptek az N darab nyilvános kulcsot tartalmazó scriptben úgy állítják be a feltételt, hogy az akadály feloldásához a nyilvános kulcsok közül legalább M darabhoz szerepelnie kell az aláírásnak. Ezt másiképp M-az-N-ből (M-of-N) sémának hívják, ahol N az összes kulcs darabszáma, M pedig az ellenőrzéshez szükséges aláírások küszöbszáma. Például egy 2-a-3-ból többszörös aláírást megkövetelő script esetén 3 nyilvános kulcs szerepel, mint lehetséges aláíró, és közülük legalább 2-nek kell aláírnia a tranzakciót ahhoz, hogy érvényes legyen, és el lehessen költeni. A könyv írásának idején a szabványos többszörös aláírást megkövetelő scriptek legfeljebb 15 nyilvános kulcsot sorolhatnak föl, vagyis az 1-az-1-ből és a 15-a-15-ből közötti bármilyen többszörös aláírást megkövetelő script használható. A 15 kulcsra történő korlátozást lehet, hogy feloldják, mire ez a könyv

megjelenik. Az `isStandard()` ellenőrzésével állapítható meg, hogy a hálózat éppen mit fogad el.

Az M-az-N-ből többszörös aláírást megkövetelő script zárolási feltételének általános alakja:

```
M <1. nyilvános kulcs> <2. nyilv. kulcs> ... <N-ik nyilv. kulcs> N  
OP_CHECKMULTISIG
```

ahol az N a felsorolt nyilvános kulcsok száma, M pedig a kimenet elkötéséhez minimálisan szükséges aláírások száma.

Egy 2-a-3-ból többszörös aláírást megkövetelő script zároló feltétele a következőképpen néz ki:

```
2 <A nyilv. kulcs> <B nyilv. kulcs> <C nyilv. kulcs> 3 OP_CHECKMULTISIG
```

A fenti zároló script egy olyan zárolást feloldó scripttel elégíthető ki, amely legalább két aláírást tartalmaz:

```
OP_0 <B aláírása> <C aláírása>
```

vagy a három felsorolt nyilvános kulcshoz tartozó titkos kulcsok közül bármelyik kettő aláírása.



Az `OP_0` előtagra azért van szükség, mivel a `CHECKMULTISIG` eredeti implementációjában van egy hiba, ami a szükségesnél eggyel több elemet emel le a veremről. Az `OP_0` egyszerűen egy helytöltő, és a `CHECKMULTISIG` elhanyagolja.

A két script együttesen a lenti egyesített ellenőrző scriptet alkotja:

```
OP_0 <B aláírása> <C aláírása> 2 <A nyilv. kulcs> <B nyilv. kulcs> <C nyilv. kulcs> 3 OP_CHECKMULTISIG
```

A végrehajtáskor a fenti egyesített script akkor és csak akkor fog IGAZ eredményt adni, ha a zárolást feloldó script megfelel a zároló script által beállított feltételeknek, vagyis esetünkben a zárolást feloldó script tartalmaz két titkos kulccsal két aláírást, és a két titkos kulcs megfelel az akadályként állított három nyilvános kulcs közül kettőnek.

Adat kimenet (`OP_RETURN`)

A bitcoin elosztott és időbényeggel ellátott főkönyvének, a blokkláncnak a pénz továbbításon kívül számos egyéb alkalmazása lehetséges. A bitcoin rendszer biztonságára és ellenállóképességére alapozva sok fejlesztő próbált a script nyelvvel egyéb alkalmazásokat létrehozni, pl. digitális közjegyzői szolgáltatásokat, részvény tanúsítványokat és intelligens szerződéseket. A korai kísérletekben, melyekben a bitcoin script nyelvét használták erre célra, ez olyan tranzakciók létrehozásával járt, melyek a blokkláncban egyéb adatokat tároltak, például egy állomány digitális ujjlenyomatát. Ily módon a tranzakció segítségével bárki meg tudta állapítani, hogy létezett-e ez az állomány egy adott időpontban.

A bitcoin blokkláncának adattárolásra történő használata független a bitcoin pénztovábbítástól, és ellentmondásos terület. Sok fejlesztő az ilyen felhasználást helytelennék tartja és elutasítja. Mások a blokklánc erősségeinek a jelét látják benne, és bátorítani akarják az ilyen kísérletezést. Azok, akik ellenzik a nem pénzügyi adatok kezelését, úgy érvelnek, hogy ez a „blokklánc meghízásához” vezet, és akadályt jelent azok számára, akik teljes bitcoin csomópontokat futtatnak, mert olyan adatok tárolási költségeit kell elviselniük, amelyeknek eredetileg nem a blokkláncban volt a helye. Még nagyobb gond, hogy az ilyen tranzakciók olyan UTXO-kat hoznak létre, melyek nem költhetők el, mert a címzett bitcoin címét szabad formátumú 20 bájtos mezőként használják. Mivel a címet adatként használják, a cím semmilyen titkos kulcsnak sem felel meg, és az így kapott UTXO soha sem lesz elkölthető, a kifizetés nem valós. Ez a gyakorlat okozta a memóriában tartott UTXO halmaz méretének megnövekedését, mivel ezek a tranzakciók *soha sem* költhetők el, emiatt soha sem lesznek eltávolítva az UTXO-k közül, ami azt UTXO adatbázis méretének állandó növekedését, „meghízását” okozza.

A Bitcoin Core kliens 0.9 verziójában az `OP_RETURN` művelet bevezetésével egy kompromisszumra jutottak. Az `OP_RETURN`-nel a fejlesztők 80 bájt nem pénzügyi adatot tárolhatnak a tranzakció kimenetében. Az „ál” UTXO-kal ellentétben azonban az `OP_RETURN` műveettel egy *bizonyíthatóan nem elkölthető* kimenet jön létre, amit nem kell az UTXO halmazban tárolni. Az `OP_RETURN` kimenetek tárolódnak a blokkláncban, emiatt diszk helyet foglalnak és hozzájárulnak a blokklánc méretének növekedéséhez, de nem tárolódnak az UTXO halmazban, és emiatt nem hízlalják feleslegesen az UTXO memória területet, és a teljes csomópontok RAM költségét sem növelik.

Az `OP_RETURN` script így néz ki:

```
OP_RETURN <adat>
```

Az adat rész 80 bájtra van korlátozva, és leggyakrabban egy hash értéknek felel meg, pl. az SHA256 algoritmus kimenetének (32 bájt). Sok alkalmazás egy előtagot helyez az adatok elé, hogy könnyebb legyen az alkalmazás azonosítása. Például a [Proof of Existence](#) digitális közjegyzői szolgáltatás a „DOCPROOF” 8 bájtos előtagot használja, amely hexadecimális alakban `444f4350524f4f46`.

Emlékeztetünk rá, hogy az `OP_RTEURN`-höz nem tartozik „zárolás feloldó script”, amellyel „elkölthető” lenne az `OP_RETURN`. Az `OP_RETURN`-nek épp az az értelme, hogy nem költhető el az adott kimenetben zárolt pénz, és ezért nem kell a kimenetet az UTXO halmazban tartani, mint potenciálisan elkölthető kimenetet – az `OP_RETURN` *bizonyíthatóan nem elkölthető*. Az `OP_RETURN` általában egy nulla összegű bitcoin kimenet, mivel az ilyen kimenethez rendelt bitcoinok örökre elvesznének. Ha a script ellenőrző program egy `OP_RETURN`-nel találkozik, akkor azonnal félbeszakítja az ellenőrző script végrehajtását, és a tranzakciót érvénytelennek tekinti. Emiatt ha véletlenül egy `OP_RETURN` kimenetre hivatkozunk egy tranzakció bemenetében, akkor a tranzakció érvénytelen lesz.

Egy érvényes tranzakciónak (amely megefelel az `isStandard()` ellenőrzéseknek) csak egy `OP_RETURN` kimenete lehet. Az `OP_RETURN` kimenet azonban tetszőleges egyéb kimeneti típusokkal kombinálható.

A Bitcoin Core 0.10.0 verziójában két új parancsori opció használható. A `datacarrier` az `OP_RETURN` tranzakciók továbbítását és bányászatát szabályozza: alapértelben „1” az

értéke, és engedélyezi őket. A `datacarriersize` opciónak egy numerikus paramétere van, mellyel az `OP_RETURN` adatok maximális hossza adható meg bájtokban, és alapértelmezett értéke 40.



Az `OP_RETURN`-nél eredetileg 80 bájtos korlátozást javasoltak, de a korlát 40 bájtra lett csökkentve, mikor ez az új jellemző megjelent. 2015 februárjában a Bitcoin Core 0.10.0-ban a korlátot ismét 80-ra emelték. A csomópontok választhatnak, hogy nem továbbítják vagy nem bányásszák ki az `OP_RETURN`-t tartalmazó tranzakciókat, vagy csak azokat az `OP_RETURN` tranzakciókat továbbítják és bányásszák ki, mely 80 bájtnál kevesebb adatot tartalmaznak.

Fizetés script hashnek (Pay to Script Hash, P2SH)

A „fizetés script hashnek” (Pay-to-script-hash, P2SH) 2012-ben lett bevezetve. A P2SH egy hatékony új tranzakciótípus, amely nagyban leegyszerűsíti a bonyolult tranzakciós scriptek használatát. Lássunk egy gyakorlati példát arra, hogy miért van szükség P2SH-ra.

Az [1. fejezetben](#) bemutattuk Mohammedet, aki elektronikai termékeket importál Dubaiba. Mohammed cége sokat használja a multi-sig scripteket a cég számláinál. A multi-sig scriptek a leggyakrabban használt korszerű bitcoin scriptek, melyek nagyon hatékonyak. Mohammed cége az összes ügyfél befizetésénél multi-sig scripteket használ, ezt a könyvelők „követelés”-nek hívják. A multi-sig scriptek használata esetén az ügyfelek befizetései úgy vannak zárolva, hogy a felszabadításukhoz legalább két aláírásra van szükség: Mohammedtől és az egyik üzlettársától, vagy az ügyvédjétől, akinek van egy tartalék kulcsa. Az ilyen multi-sig tranzakciók lehetővé teszik a cégvezetés számára a felügyeletet, és védenek a lopás, hűtlen kezelés (sikkasztás) és a veszteségek ellen.

Az így kapott script egészen hosszú, és így néz ki:

```
2 <Mohammed nyilvános kulcsa> <Az 1. partner nyilvános kulcsa> <A 2. partner nyilvános kulcsa> <A 3. partner nyilvános kulcsa> <Az ügyvéd nyilvános kulcsa> 5 OP_CHECKMULTISIG
```

Noha a multi-sig scriptek nagyon hatékonyak, de nehezen használhatók. A fenti script esetén Mohammednek a fizetés előtt mindegyik ügyfélhez el kell juttatnia a fenti scriptet. Mindegyik ügyfélnek különleges bitcoin pénztárcát kell használnia, melynek szoftvere képes egyedi tranzakciós scriptek előállítására, és minden ügyfélnek tudnia kell, hogyan lehet az egyedi scripttel egy tranzakciót létrehozni. Ezen kívül a kapott tranzakció kb. ötször nagyobb lesz, mint egy egyszerű fizetési tranzakció, mivel a script nagyon hosszú nyilvános kulcsokat tartalmaz. Az extra-nagy tranzakció terhét tranzakciós díjak formájában az ügyfél állja. Végül, az ilyen nagy tranzakciós scripteket mindenki teljes csomópont az UTXO halmazban, a RAM-ban tartja mindaddig, amíg el nem költik. Ezen okok miatt a bonyolult scriptek használata a gyakorlatban nehéz.

A „fizetés-script-hashnek” (P2SH, Pay-to-Script-Hash) scripteket azért fejlesztették ki, hogy megoldják ezeket a gyakorlati nehézségeket, és a bonyolult scriptek használatát is olyan egyszerűvé tegyék, mint egy bitcoin címre történő pénzküldést. A P2SH fizetési mód esetében a zároló scriptet a digitális ujjlenyomatá, egy kriptográfiai hash helyettesíti. Amikor utóbb egy olyan tranzakció keletkezik, amely megkísérli elkölni az UTXO-t, akkor ennek a zárolást feloldó scripten kívül tartalmaznia kell azt a scriptet is, amelynek hashe egyezik az eredetileg

megadott hashsel. A P2SH egyszerűen azt jelenti, hogy „fizess annak a scriptnek, amelynek a hashe egyezik ezzel a hashsel, a script később, a kimenet elköltésekor lesz bemutatva”.

A P2SH tranzakciókban a zároló scriptet egy hash helyettesíti, melynek *redeem script* (beváltási script) a neve, mert a beváltáskor kerül majd bemutatásra a rendszernek. Az 5-4. táblázat P2SH nélkül mutatja a példa scriptet, míg az 5-5. táblázat ugyanezt a scriptet P2SH kódolással mutatja.

5-4. táblázat: Egy bonyolult script, P2SH nélkül

Zároló script	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 OP_CHECKMULTISIG
Zárolást feloldó script	Sig1 Sig2

5-5. táblázat: Egy bonyolult script P2SH használatával

Beváltási script	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 OP_CHECKMULTISIG
Zároló script	OP_HASH160 <a beváltási script 20 bájtos hashe> OP_EQUAL
Zárolást feloldó script	Sig1 Sig2 beváltási script

Amint az a fenti táblázatokból látható, a P2SH használatakor a nem jelenik meg az a bonyolult script a kimenetben, amely a kimenet elköltésének feltételeit részletezi. Csak a script egy hashe van jelen a zárolást végző scriptben, a beváltási script pedig később, a kimenet elköltésekor, a zárolást feloldó script részeként kerül bemutatásra. Ennek révén a komplexitás és a tranzakciós díjak terhe a tranzakció küldőjéről a címzetre tevődik át.

Tekintsük Mohanmmed cégét, a bonyolult multi-sig scriptet és az eredményként kapott P2SH scriptet.

Nézzük először azt a multi-sig scriptet, melyet Mohammed cége az összes bejövő fizetésnél használ:

```
2 <Mohammed nyilvános kulcsa> <Az 1. partner nyilvános kulcsa> <A 2. partner nyilvános kulcsa> <A 3. partner nyilvános kulcsa> <Az ügyvéd nyilvános kulcsa> 5 OP_CHECKMULTISIG
```

Ha a fenti üres helyekre behelyettesítjük a tényleges nyilvános kulcsokat, (melyek 04-gyel kezdődő, 520 bites számok), akkor a script nagyon hosszú lesz:

```
2
04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC
5C395D7EBC6984D83F1F50C900A24DD47F569FD4193AF5DE762C58704A2192968D8655D6A93
5BEAF2CA23E3FB87A3495E7AF308EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD2423657738C0
C2B1D5CE65C97D78D0E34224858008E8B49047E63248B75DB7379BE9CDA8CE5751D16485F43
1E46117B9D0C1837C9D5737812F393DA7D4420D7E1A9162F0279CFC10F1E8E8F3020DECDBC3
C0DD389D99779650421D65CBD7149B255382ED7F78E946580657EE6FDA162A187543A9D85BA
AA93A4AB3A8F044DADA618D087227440645ABE8A35DA8C5B73997AD343BE5C2AFD94A504375
```

```
2580AFA1ECED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF  
518C022DD618DA774D207D137AAB59E0B000EB7ED238F4D800 5 OP_CHECKMULTISIG
```

De az egész script egy 20 bájtos hashsel ábrázolható, ha először az SHA256 hash algoritmust, majd a RIPEMD160 algoritmust alkalmazzuk a scriptre. A fenti script 20 bájtos hashe:

```
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

A P2SH tranzakció a kimenetét a hosszabb script helyett a következő scripttel zárolja:

```
OP_HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e OP_EQUAL
```

amely láthatóan sokkal rövidebb. Ahelyett, hogy azt mondánánk, „fizess erre az 5 kulcsból álló multi-sig címre”, az ezzel egyenértékű P2SH tranzakció a következő: „fizess annak a scriptnek, melynek ez és ez a hashe”. Mohammed ügyfeleinek csupán ezt a jóval rövidebb zároló scriptet kell megadniuk. Ha Mohammed el akarja költeni ezt az UTXO-t, akkor be kell mutatnia az eredeti beváltási scriptet (azt, amelynek hashével az UTXO zárolva lett), valamint a zárolást feloldó aláírásokat, pl. így:

```
<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG>
```

A két script összekapcsolása két szakaszban történik. Először a beváltási script hashe kerül ellenőrzésre, hogy megegyezik-e a zároló scriptben lévő hashsel:

```
<2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG> OP_HASH160 <beváltási script  
hashe> OP_EQUAL
```

Ha a beváltási script hashe megegyezik a zároló scriptben lévő hashsel, akkor zárolást feloldó script egymagában kerül végrehajtásra, hogy megszüntesse a zárolást:

```
<Sig1> <Sig2> 2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG
```

„Fizetés script hashnek” címek (Pay-to-Script-Hash Addresses)

A P2SH egy másik fontos jellemzője, hogy a script hash címként is kódolható, amint azt a BIP0013 definiálja. A P2SH címek a script 20 bájtos hashének Base58Check kódolásával állnak elő, pont úgy, ahogy a bitcoin címek a nyilvános kulcs 20 bájtos hashének Base58Check kódolásával. A P2SH címek az „5” verziót előtagot használják, ez pedig „3”-mal kezdődő Base58Check kódolású címeket eredményez. Például Mohammed bonyolult scriptjéből a hashelés és Base58Check kódolás után a

39RF6JqABiHdYHkfChV6USGM6Nsrg6Gzw cím lesz. Mohammed ezt a „címet” oda tudja adni az ügyfeleinek, ők pedig szinte bármilyen pénztárcát használhatnak, és úgy fizethetnek, mintha egy egyszerű bitcoin címről lenne szó. A 3-as előtag jelzi nekik, hogy különleges címről van szó, ami nem nyilvános kulcshoz, hanem script hashhez tartozik, egyébként azonban pontosan úgy működik, mint egy bitcoin címre történő pályázás.

A P2SH címek elrejtik a bonyolultságot, a fizetést végrehajtó személy nem látja a scriptet.

A „fizetés script hashnek” (Pay-to-Script-Hash) előnyei

A „fizetés script hashnek” a következő előnyökkel rendelkezik a kimenetek zárolására használt bonyolult scriptek közvetlen használatához képest:

- a bonyolult scripteket a tranzakció kimenetben a rövidebb ujjenyomatok helyettesítik, ezáltal a tranzakció kisebb lesz
- a scriptek címként kódolhatók, ezért a küldőnek és a küldő pénztárcájának nincs szüksége a P2SH bonyolult megvalósítására
- a P2SH a script előállításának a terhét a küldőről a címzettre hárítja át
- a P2SH a hosszú script adattárolásának a terhét a kimenetről (ami az UTXO halmazban van) a bemenetre (amit csak a blokklánc tárol) hárítja át
- a P2SH a hosszú script adattárolási terhét a jelenből (fizetés) a jövőbe (amikor elköltik) viszi át
- a P2SH a hosszú script miatti tranzakciós díjat a küldőről a címzettre hárítja, mert a címzettnek kell a hosszú beváltási scriptet bemutatnia, ha el akarja költeni az összeget.

A beváltási script és az isStandard ellenőrzés

A Bitcoin Core kliens 0.9.2-es változata előtt a „fizetés script hasnek” (Pay-to-Script-Hash) az `isStandard()` függvény által engedélyezett szabványos bitcoin tranzakciós script típusokra korlátozódott. Ez azt jelenti, hogy az összeg elköltésekor bemutatott beváltási script a következő szabványos típusok valamelyike lehetett: P2PK, P2PKH vagy Multi-Sig, de nem lehetett `OP_RETURN` és P2SH.

A Bitcoin Core kliens 0.9.2-es verziója óta a P2SH scriptek bármilyen érvényes scriptet tartalmazhatnak, ami a P2SH szabványt sokkal rugalmasabbá teszi, és sok újfajta, összetett tranzakciótípus kipróbálását/használatát teszi lehetővé.

Megjegyezzük, hogy egy P2SH beváltási scripten belül nem lehet újabb P2SH script, mivel a P2SH specifikáció nem engedi meg a rekurziót. A beváltási scriptben `OP_RETURN` sem használható, mivel az `OP_RETURN` definíció szerint nem költhető el.

Megjegyezzük, hogy mivel a beváltási script addig nem jelenik meg a hálózatban, amíg meg nem próbáljuk meg elkölni a P2SH kimenetet, ezért ha egy érvénytelen beváltási script hashével zárolunk egy kimenetet, akkor ez feldolgozásra fog kerülni. De a kimenetet nem tudjuk majd elkölni, mivel az összeg elköltésekor meg kell adni a beváltási scriptet, ezt viszont a rendszer nem fogja elfogadni, mert nem érvényes. Ez kockázatot jelent, mivel a P2SH-val úgy zárolhatók bitcoinok, hogy később sem lehet elkölni őket. A hálózat még akkor is elfogadja a P2SH „akadályt”, ha az egy érvénytelen beváltási scriptnek felel meg, mert a script hashe semmilyen utalást sem ad arra vonatkozóan, hogy a hash milyen scriptnek felel meg.



A P2SH zároló scriptek a beváltási script hashét tartalmazzák, ami semmilyen utalást sem ad magára a beváltási scriptre vonatkozóan. A P2SH tranzakció még akkor is érvényes lesz, ha a beváltási script érvénytelen. A P2SH-val véletlenül úgy is zárolhatók a bitcoinok, hogy később sem lesznek elkölhethetők.

6. A bitcoin hálózat

Peer-to-peer hálózati felépítés

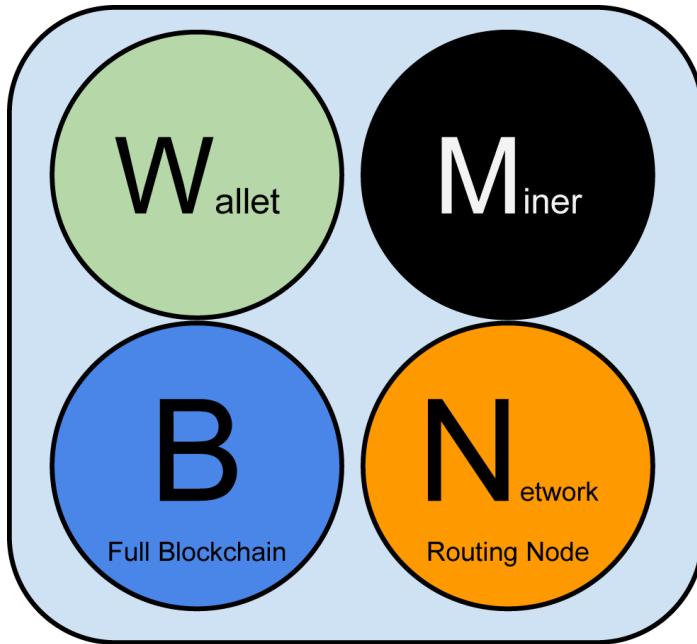
A bitcoin az Internetre épülő peer-to-peer hálózati felépítéssel rendelkezik. A peer-to-peer kifejezés azt jelenti, hogy a hálózatban részt vevő csomópontok egyenrangúak, vagyis nincsenek „különleges” csomópontok, és mindenki csomópont kiveszi a részét abból a teherből, amit a hálózati szolgáltatások nyújtása jelent. A hálózat csomópontjai „egyforma” topológiájú hálózatban kapcsolódnak egymáshoz, nincsenek „szerverek”, központosított szolgáltatások, és a hálózaton belül nincs alá- vagy fölérendeltség. A peer-to-peer hálózatok csomópontjai egyszerre szolgáltatók és fogyasztók, ahol a kölcsönösségek a részvétel egyik ösztönzője. A peer-to-peer hálózatok nagyon ellenállóak, decentralizáltak és nyitottak. A P2P hálózati architektúrára maga a korai Internet jelentett kiváló példát, mert az IP hálózat csomópontjai egyenértékűek voltak. Manapság az Internet felépítése hierarchikusabb, de az Internet Protokoll még mindig őrzi az egynemű topológia lényegét. A bitcoinon kívül a P2P technológia legnagyobb és legsikeresebb alkalmazása a file megosztás: itt a Napster volt az architektúra úttörője, és a bittorrent a legutóbbi fejleménye.

A Bitcoin P2P hálózati felépítése sokkal több, mint topológia kérdése. A bitcoin egy peer-to-peer digitális pénzügyi rendszerként lett megtervezve, és a hálózat felépítése ennek az alapjellemzőnek a tükrözödése és a megtestesülése. Az irányítás decentralizálása tervezési alapelv. Ez az alapelvek csak egy hierarchia nélküli, decentralizált P2P konszenzus révén valósítható meg és tartható fonn.

A „bitcoin hálózat” kifejezés a bitcoin P2P protokollt futtató csomópontok halmazát jelenti. A bitcoinban a P2P protokollon túlmenően egyéb protokollok is vannak, pl. a Stratum, melyet a bányászatnál vagy a pehelysúlyú mobil pénztárcánál alkalmaznak. Ezeket a további protokollokat a bitcoin hálózathoz kapcsolódó gateway router szolgáltatások biztosítják, melyek a P2P protokollon keresztül kapcsolódnak a bitcoin hálózathoz, és a hálózatot az egyéb protokollokat futtató csomópontok irányában terjesztik ki. Például a Stratum szerverek a bányászatot végző Stratum csomópontokat kapcsolják össze a Stratum protokoll segítségével a fő bitcoin hálózattal, vagyis a Stratum protokollt a bitcoin P2P protokolljával kötik össze. A bitcoin P2P protokolljára, a bányatársaságok protokolljaira, a Stratum protokollra és a bitcoin rendszer részeit összekötő egyéb protokollokra együtt a „kiterjesztett bitcoin hálózat” kifejezéssel hivatkozunk.

Csomópont típusok és szerepek

Noha a bitcoin P2P hálózatában lévő csomópontok egyenértékűek, de attól függően, hogy milyen működési módokat támogatnak, különböző „szerepeket” játszhatnak. Egy bitcoin csomópont a következő funkciókat valósíthatja meg: routing, blokklánc adatbázis, bányászat, pénztárca szolgáltatások. Lent a 6-1. ábrán egy teljes csomópont látható, mely minden funkciót támogatja:



6-1. ábra: Egy bitcoin hálózati csomópont, amely mind a négy funkcióval rendelkezik: hálózati router, blokklánc adatbázis, bányászat és pénztárca

(Balról jobbra, felülről lefelé: pénztárca, bányász, teljes blokklánc, hálózati router csomópont)

A hálózat összes csomópontja tartalmazza a router funkciót, hogy a csomópont részt vehessen a hálózatban. A csomópontok egyéb funkciókat is tartalmazhatnak. Valamennyi csomópont ellenőrzi és továbbítja a tranzakciókat és blokkokat, valamint összeköttetéseket hoz létre és tart fenn a többi hálózati csomóponttal. A fenti teljes csomópont esetében a router funkciót egy „Hálózati router csomópont” feliratú sárga kör jelzi.

Vannak olyan csomópontok – ezeket teljes csomópontoknak hívjuk – melyek teljes és naprakész másolatot tartanak fönn a blokkláncról. A teljes csomópontok külső hivatkozás nélkül, önállóan és hitelesen képesek bármely tranzakció ellenőrzésére. Némelyik csomópont csak a blokklánc egy részhalmazát kezeli, és a tranzakciókat az ún. Egyszerűsített Fizetési Ellenőrzés módszerével ellenőrzi (SPV, Simplified Payment Verification). Ezek a csomópontok a SPV vagy másképpen pehelysúlyú csomópontok. A fenti teljes csomópont esetében a blokklánc adatbázis funkciót egy „Teljes blokklánc” feliratú kék kör jelzi.

A bányász csomópontok új blokkokat hoznak létre oly módon, hogy célhardvert használnak a munkabizonyíték (proof-of-work) algoritmus megoldására. Némelyik bányász csomópont egyúttal teljes csomópont is, míg a többiek pehelysúlyú csomópontok, melyek társult bányászatban (pooled mining) vesznek részt, és egy szerverre bízzák a teljes csomópont kezelését. A bányász funkciót a fenti teljes csomópont esetén a „Bányász” feliratú fekete kör jelzi.

A teljes csomópontok pénztárcát is tartalmazhatnak. Az asztali bitcoin kliensek esetében általában ez a helyzet. Egyre több pénztárca, különösen azok, melyek erőforrásokban korlátozott eszközökön, pl. okostelefonokon futnak, SPV csomópontok. A pénztárca funkciót fent egy „Pénztárca” feliratú zöld kör mutatja.

A bitcoin P2P protokollhoz tartozó leggyakoribb csomópont típusokon kívül vannak olyan szerverek és csomópontok, melyek egyéb protokollokat futtatnak, pl. specializált bányász-protokollokat és pehelysúlyú kliens elérési protokollokat.

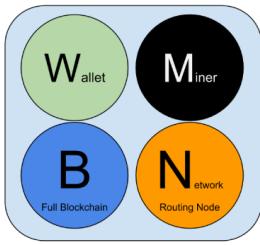
A 6-2. ábra a kiterjesztett bitcoin hálózat különféle csomópont-típusait mutatja.

A kiterjesztett bitcoin hálózat

A P2P protokollt futtató fő bitcoin hálózat kb. 7000 és 10'000 közötti csomópontból áll. A csomópontok a bitcoin referencia kliens (Bitcoin Core) különféle változatait futtatják. Néhány száz további csomópont a bitcoin P2P protokoll különféle egyéb megvalósításait futtatja, ilyen pl. a BitcoinJ, a LibbitcoiN és a btcd. A bitcoin P2P hálózat csomópontjainak kis százaléka egymással versenyezve részt vesz a bányászatban, ellenőrzi a tranzakciókat és új blokkokat hoz létre. Számos nagy cég úgy teremt kapcsolatot a bitcoin hálózattal, hogy a Bitcoin Core kliensen alapuló teljes csomópontot futtat, amelyben szerepel a blokklánc és a hálózat kezelés, de hiányzik belőle a bányászati és pénztárca funkció. Ezek a csomópontok a hálózat határán lévő routerekként használhatók, és különféle egyéb szolgáltatások ráépítését teszik lehetővé (pénzváltók, pénztárcák, blokk explorer-ek, fizetés feldolgozás).

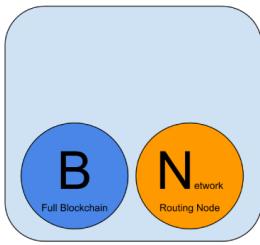
A kiterjesztett bitcoin hálózat tartalmazza a fent leírt, bitcoin P2P protokollt futtató hálózati csomópontokat, valamint a specializált protokollokat futtató csomópontokat egyaránt. A bitcoin P2P hálózathoz számos bányatársaság szervere kapcsolódik, valamint olyan protokoll gateway-ek, melyek a csomópontokat az egyéb protokollokat futtató csomópontokkal kötik össze, többnyire bányász csomópontokkal (lásd a [8. fejezetet](#)), vagy pehelysúlyú kliensekkel, melyek nem tartalmazzák a blokklánc teljes másolatát.

A 6-3. ábra a különféle csomópont típusokat, gateway-eket, routereket, pénztárcákat, valamint a közöttük kapcsolatot teremtő protokollokat mutatja.



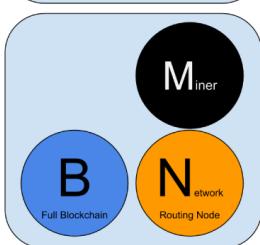
Reference Client (Bitcoin Core)

Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.



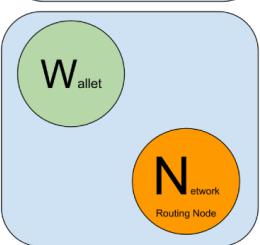
Full Block Chain Node

Contains a full Blockchain database, and Network routing node on the bitcoin P2P network.



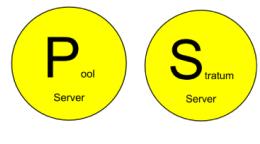
Solo Miner

Contains a mining function with a full copy of the blockchain and a bitcoin P2P network routing node.



Lightweight (SPV) wallet

Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.



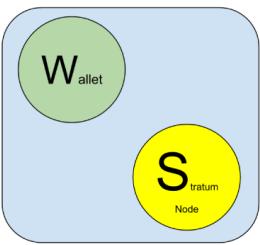
Pool Protocol Servers

Gateway routers connecting the bitcoin P2P network to nodes running other protocols such as pool mining nodes or Stratum nodes.



Mining Nodes

Contain a mining function, without a blockchain, with the Stratum protocol node (S) or other pool (P) mining protocol node.

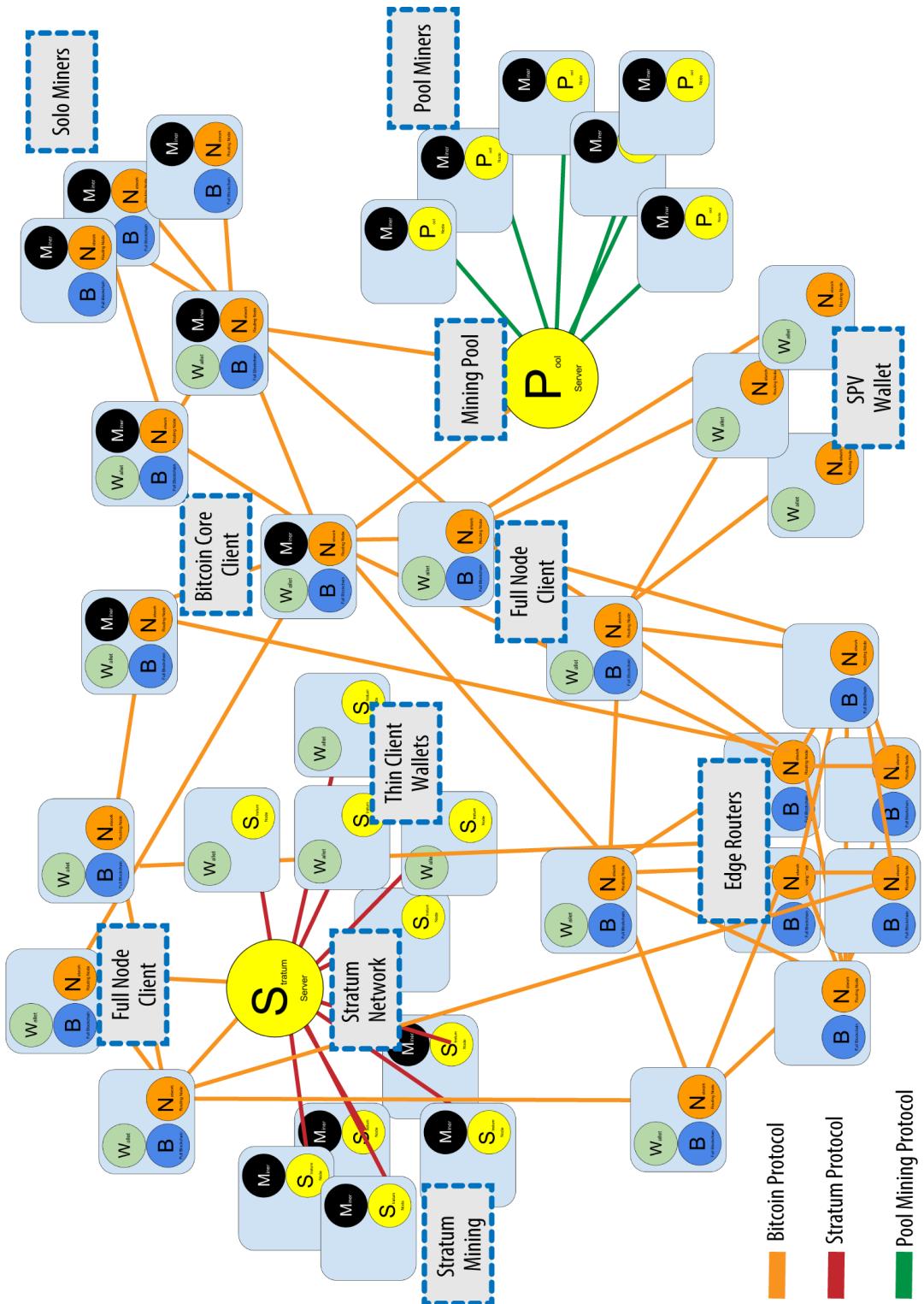


Lightweight (SPV) Stratum wallet

Contains a Wallet and a Network node on the Stratum protocol, without a blockchain.

6-2. ábra: Különböző csomópont típusok a kiterjesztett bitcoin hálózaton

(Balról jobbra, felülről lefelé: teljes csomópont, szingli bányászok, Bitcoin Core kliens, Stratum hálózat, Stratum bányászat, pehelysúlyú pénztárcák, bányatársaság bányászai, bányatársaság, teljes csomópont, Edge routerek, SPV pénztárca)



6-3. ábra: A különféle csomópont típusokat, gateway-eket és protokollokat tartalmazó kiterjesztett bitcoin hálózat

(Balról jobbra, felülről lefelé: teljes csomópont, szingli bányászok, Bitcoin Core kliens, Stratum hálózat, Stratum bányászat, pehelysúlyú pénztárcák, bányatársaság bányászai, bányatársaság, teljes csomópont, Edge routerek, SPV pénztárca)

Hálózat felderítés

Ahhoz, hogy egy új csomópont részt tudjon venni a hálózatban, a csomópontnak az indulásakor fel kell derítenie, hogy milyen más csomópontok vannak a hálózatban. A folyamat beindításához az új csomópontnak fel kell derítenie a hálózat legalább egy létező csomópontját, és kapcsolónia kell hozzá. A csomópontok földrajzi elhelyezkedése lényegtelen, a bitcoin hálózat topológiáját nem a földrajzi hely határozza meg. Ezért a csomópont véletlenszerűen, bármelyik létező bitcoin csomópontot választhatja.

A csomópontok egy ismert peerhez TCP összeköttetéssel kapcsolódnak, általában a 8333-as porton (amely a bitcoin „jól ismert” portja), vagy egy alternatív porton, ha megadtak ilyet. Az összeköttetés létrejötte után a csomópont a `version` (verzió) üzenet elküldésével egy „kézfogást” indít, lásd a 6-4. ábrát. A `version` üzenet azonosító adatokat tartalmaz. Ezek a következők:

`PROTOCOL_VERSION`

a kliens által „beszélt” protokoll verzióját definiáló konstans (pl. 70002)

`nLocalServices`

a csomópont által támogatott helyi szolgáltatások listája, jelenleg csupán a `NODE_NETWORK` (hálózati csomópont) az egyetlen eleme

`nTime`

az aktuális idő

`addrYou`

a távoli csomópont IP címe, ahogyan azt a csomópont látja

`addrMe`

a helyi csomópont IP címe, ahogyan azt a helyi csomópont látja

`subver`

az al-verzió, amely a csomóponton futó szoftver típusát mutatja (pl. "/Satoshi:0.9.2.1/“)

`BestHeight`

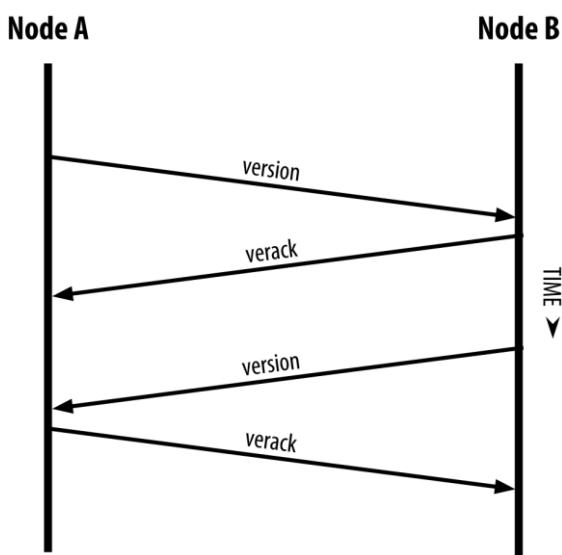
a csomópont blokkláncának a magassága

(A `version` hálózati üzenetre a [GitHub](#) web helyen látható példa.)

A peer csomópont egy `verack` üzenettel válaszol, és opcionálisan elküldi a saját `version` üzenetét, ha szeretné viszonzni a kapcsolatot, és szeretne ő is peerként kapcsolódni.

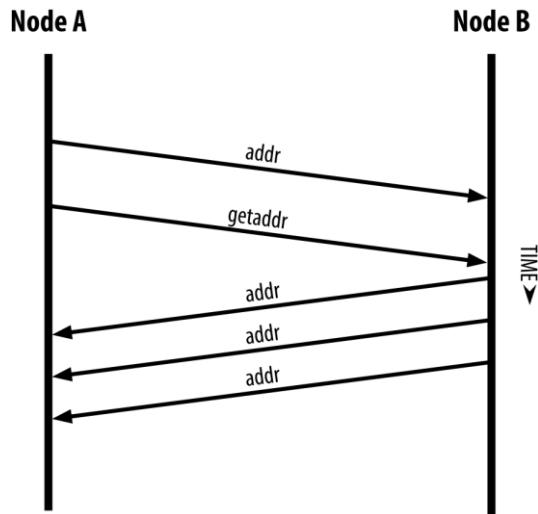
Hogyan találja meg egy új csomópont a peereket? Az első módszert a DNS-ek lekérdezése jelenti bizonyos „DNS magok” használatával, melyek olyan DNS szerverek, melyek bitcoin csomópontok IP címeinek a listáját adják vissza. A DNS magok némelyike stabil bitcoin csomópontok statikus listáját adja vissza. Vannak olyan DNS magok is, melyek a BIND (Berkeley Internet Name Daemon) egyedi megvalósításai, és bitcoin csomópontok véletlen részhalmazának a címeit adják vissza. Ezek a címek egy crawler-ból vagy egy hosszú ideje futó bitcoin csomópontról származnak. A Bitcoin Core kliens öt különböző DNS mag nevét tartalmazza. A DNS magok tulajdonosainak és megvalósításainak a változatos volta biztosítja a kezdeti bootstrap folyamat magas szintű megbízhatóságát. A Bitcoin Core kliensben a `-dnsseed` kapcsoló szabályozza, hogy a kliens használja-e a DNS magokat, és alapértelben 1 az értéke.

Ha viszont az induló csomópont semmit sem tud a hálózatról, akkor legalább egy bitcoin csomópont IP címét meg kell adni neki, és ezután már további bemutatkozások révén a többi csomóponttal is kapcsolatba tud lépni. A `-seednode` parancssori argumentum azt jelzi, hogy a megadott csomóponthoz csak a bemutatkozás kedvéért szeretnénk hozzákapcsolódni, és szeretnénk magként használni. Miután a kezdeti mag csomópont segítségével megtörténtek a bemutatkozások, a kliens lekapcsolódik róla, és az újonnan felfedezett peereket fogja használni.



6-4. ábra: A peerek közötti kezdeti kézfogás

Miután már létrejött egy vagy több kapcsolat, az új csomópont egy `addr` üzenetet fog küldeni a szomszédainak, amely tartalmazza a saját IP címét. A szomszédok az `addr` üzenetet tovább küldik a saját szomszédaiknak, biztosítván ezáltal, hogy a kapcsolódó csomópontok jól ismertek és jobban kapcsolódók legyenek. Az újonnan kapcsolódó csomópont ezen kívül egy `getaddr` üzenetet is küldhet a szomszédainak, amivel azt kéri tőlük, hogy küldjék el neki a többi peer IP címeit. Ily módon a csomópont meg tudja keresni, mely csomópontokhoz kapcsolódhat, valamint hírt tud adni a saját létéiről a hálózaton a célból, hogy a többi csomópont is képes legyen őt megtalálni. A 6-5. ábrán a cím felderítési protokoll látható.



6-5. ábra: Címterjedés és hálózat felderítés

Egy csomópontnak kapcsolódnia kell néhány különböző csomóponthoz, hogy különféle útvonalakon kapcsolódhasson a bitcoin hálózathoz. Az útvonalak nem megbízhatóak, csomópontok jönnek és mennek, emiatt a csomópontnak folytatnia kell az új csomópontok felderítését, mivel a régi kapcsolatai bármikor megszűnhetnek, és emellett segítenie kell a többi csomópont elindulását. Az induláshoz csak egyetlen összeköttetés szükséges, mivel az első csomópont képes bemutatkozásokat felajánlani a peer csomópontoknak, ezek a peerek pedig képesek további bemutatkozásokat felajánlani. A hálózati erőforrások szempontjából szükségtelen és egyúttal pazarló, ha a csomópont pár darab csomópontnál több csomóponthoz kapcsolódik. Az indulás után a csomópont emlékszik a legutolsó sikeres peer kapcsolataira, emiatt újraindítás után az előző peer hálózattal ismét gyorsan létre tudja hozni a kapcsolatait. Ha az újrainduláskor az előző peerek egyike sem válaszol a kapcsolódási kérésére, akkor a csomópont a mag csomópontokat használja.

A Bitcoin Core klienst futtató csomóponton a peer kapcsolatok a `getpeerinfo` parancssal listázhatók ki:

```
$ bitcoin-cli getpeerinfo
[
  {
    "addr" : "85.213.199.39:8333",
    "services" : "00000001",
    "lastsend" : 1405634126,
    "lastrecv" : 1405634127,
    "bytessent" : 23487651,
    "bytesrecv" : 138679099,
    "conntime" : 1405021768,
    "pingtime" : 0.00000000,
    "version" : 70002,
    "subver" : "/Satoshi:0.9.2.1/",
    "inbound" : false,
    "startingheight" : 310131,
    "banscore" : 0,
    "syncnode" : true
  },
  {
    "addr" : "58.23.244.20:8333",
    "services" : "00000001",
```

```

        "lastsend" : 1405634127,
        "lastrecv" : 1405634124,
        "bytessent" : 4460918,
        "bytesrecv" : 8903575,
        "conntime" : 1405559628,
        "pingtime" : 0.00000000,
        "version" : 70001,
        "subver" : "/Satoshi:0.8.6/",
        "inbound" : false,
        "startingheight" : 311074,
        "banscore" : 0,
        "syncnode" : false
    }
]

```

A felhasználók egy IP címekből álló lista megadásával, a `-connect=<IP cím>` opcióval tudják felülbírálni a peerek automatikus kezelését. Ennek az opciónak a használatakor a csomópont csak a megadott IP címekhez fog kapcsolódni, és nem fogja automatikusan felderíteni és karbantartani a peer kapcsolatokat.

Ha egy kapcsolaton nincs forgalom, akkor a kapcsolat fenntartása érdekében a csomópont periodikus üzenetküldést végez rajta. Ha a csomópont egy kapcsolata már több mint 90 perce nem volt használva, akkor a csomópont a kapcsolatot szétkapcsolt állapotúnak tekinti, és egy új peer keresésébe kezd. Ily módon a hálózat dinamikusan alkalmazkodni képes a tranzis csomópontokhoz, a hálózati problémákhoz, és központi irányítás nélkül, organikusan képes nőni vagy csökkenni.

Teljes csomópontok

A teljes csomópontok olyan csomópontok, melyek az összes tranzakciót tartalmazó, teljes blokkláncot kezelik. Ezeket a csomópontokat pontosabban „a teljes blokkláncot kezelő csomópontok”-nak kellene hívni. A bitcoin korai éveiben az összes csomópont teljes csomópont volt, jelenleg a Bitcoin Core kliens kezeli a teljes blokkláncot. Az utóbbi két évben azonban a bitcoin kliensek új fajtái jöttek létre, melyek nem kezelik a teljes blokkláncot, hanem pehelysúlyú kliensként futnak. Ezeket a következő részben fogjuk részletesebben megvizsgálni.

A teljes blokkláncot kezelő csomópontok a bitcoin blokklánc egy teljes és naprakész másolatát kezelik, melyben az összes tranzakció megtalálható. A blokkláncot egymástól függetlenül építik föl és ellenőrzik, az első blokktól (a genezis blokktól) kezdve, egészen a hálózatban ismert legutolsó blokkig bezárólag. Egy teljes blokkláncot kezelő csomópont önmaga képes hiteles módon bármelyik tranzakció ellenőrzésére, anélkül, hogy ehhez valamilyen másik csomópontot vagy információs forrást kellene igénybe vennie. A teljes csomópont a hálózatra támaszkodva kapja az új tranzakciós blokkokról az értesítéseket. Ezeket ellenőrzi, majd beépíti a saját, lokális blokkláncába.

Teljes csomópont futtatásával érezhetjük igazán, milyen a bitcoin: az összes tranzakció függetlenül ellenőrizhető, és ehhez semmilyen más rendszerre sem kell támaszkodnunk, és semmilyen más rendszerben sem kell megbíznunk. Könnyen el lehet dönteneni, hogy teljes csomópontot futtatunk-e, mert több, mint 20 Gb által háttérben van szükség a teljes blokklánc tárolásához. Ha a kliens sok diszk területet fogyaszt és 2-3 napra van

szüksége, hogy „szinkronizálja” magát a hálózattal, akkor teljes csomópontot kezelő kliensről van szó. A központi szervezetektől való teljes függetlenségnek és szabadságnak ez az ára.

A teljes blokkláncot kezelő klienseknek van néhány alternatív megvalósítása, melyek a Bitcoin Core klienstől eltérő programozási nyelvet vagy szoftver architektúrát használnak. De a Bitcoin Core kliens, másnéven a Satoshi kliens referencia implementáció fordul elő a leggyakrabban. A bitcoin hálózati csomópontjainak több, mint 90 %-a a Bitcoin Core különféle változatait futtatja. A `version` üzenetben a „Satoshi” al-verzió string azonosítja őket, a `getpeerinfo` parancsban pedig pl. a `/Satoshi:0.8.6/`, amint azt korábban láttuk.

„Leltár” egyeztetés

Miután a csomópont hozzákapcsolódott a peerjeihez, elsőként egy teljes blokkláncot próbál létrehozni. Ha egy vadonatúj csomópontról van szó, amelynek egyáltalán nincs még blokklánca, akkor csak egyetlen blokkot ismer (a genezis blokkot), amely statikusan be van ágyazva a kliens szoftverbe. A 0-ik blokktól, a genezis blokktól kezdve az új csomópontnak blokkot százezreit kell letöltenie ahoz, hogy szinkronizálhassa magát a hálózattal és újraépíthesse a teljes blokkláncot.

A „szinkronizálás” folyamata a `version` üzenettel kezdődik, amely tartalmazza a `BestHeight`-et, a csomópont aktuális blokkláncának a magasságát (a blokkok számát). A csomópont a peerjeitől kapott `version` üzenetből látja, hogy a peernek hány blokkjuk van, és össze tudja hasonlítani azzal, hogy neki hány blokkja van a saját blokkláncán. A peer csomópontok egy `getblocks` üzenetet váltanak egymással, amely tartalmazza a lokális blokkláncuk legfelső blokkjának hashét (ujjlenyomatát). A kapott hash az egyik peerben egy olyan blokkhoz fog tartozni, amely nem a legfelső blokk, hanem egy régebbi blokk, ebből a peer arra következtet, hogy a saját lokális blokklánca hosszabb, mint a többi peeré.

Az a peer, amelynek hosszabb a blokklánca, több blokkot tartalmaz, mint a többi csomópont, és meg tudja állapítani, hogy a többi csomópontnak mely blokkokra van szüksége ahoz, hogy „felzárkózzanak”. Megállapítja, hogy melyik az első 500 megesztendő blokk, és egy `inv` (inventory, leltár) üzenettel elküldi a blokkok hash értékeit. Az a csomópont, amelyben hiányoznak ezek a blokkok, úgy tudja beszerezni őket, hogy `getdata` üzenetek sorozatát adj ki. Egy `getdata` üzenet elkéri a teljes adatblokkot, és a kért blokkot az `inv` üzenetből származó hashsel azonosítja.

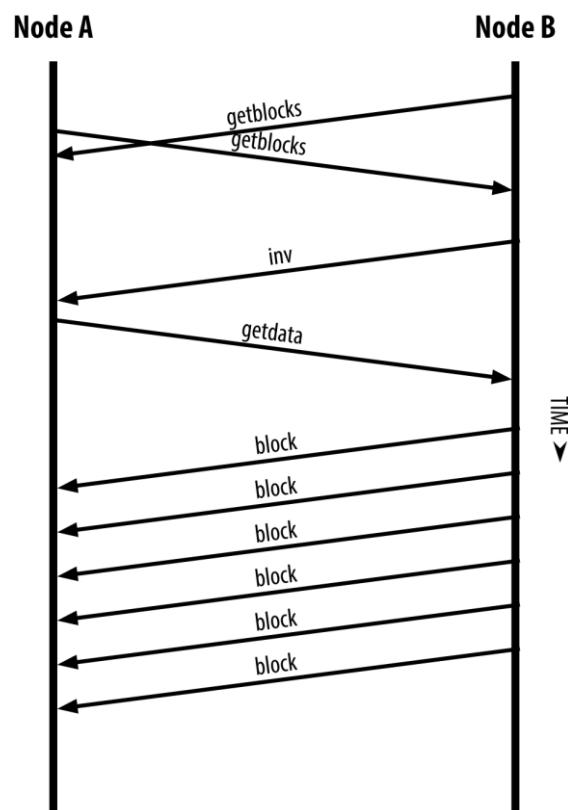
Tegyük fel például, hogy a csomópont csak a genezis blokkot tartalmazza. A peerjeitől egy `inv` üzenetet fog kapni, amely lánc következő 500 blokkjának a hashait tartalmazza. Megkezdi a vele kapcsolatban lévő peerektől a blokkok lekérését oly módon, hogy elosztja a terhelést, nehogy bármelyik peert túlterhelje a kéréseivel. Számon tartja, hogy minden egyes peer kapcsolatnál hány darab blokk van „úton”, vagyis hány darab blokk van, melyet lekért, de még nem kapott meg, és ellenőrzi, hogy a számuk nehogy egy határnál (`MAX_BLOCKS_IN_TRANSIT_PER_PEER`) nagyobb legyen. Ily módon ha a csomópontnak sok blokkra van szüksége, csak akkor kér újabbakat, ha az előző kérései már teljesültek, ami lehetővé teszi, hogy a peerek szabályozhassák a küldés ütemét és a hálózat ne terhelődjön túl. A blokkok megérkezésekor a csomópont hozzáadja a blokkokat a blokklánchoz, amint azt a [7. fejezetben](#) látni fogjuk. Amint a lokális blokklánc fokozatosan felépül, a csomópont további blokkokat kér és kap. A folyamat addig folytatódik, amíg a csomópont be nem éri a hálózat többi részét.

A lokális blokklánc és a peerek blokkláncainak összehasonlítása, valamint a hiányzó blokkok lekérése akkor megy végbe, ha egy csomópont egy időre offline állapotba került. Függetlenül attól, hogy a csomópont csak néhány percig volt offline, és csak pár blokkja hiányzik, vagy hónapokig, és néhány ezer blokkja hiányzik, a folyamat a `getblocks` küldésével kezdődik, válaszként egy `inv` érkezik, majd megtörténik a hiányzó blokkok letöltése. A 6-6. ábra a leltár és blokk terjedési protokollt mutatja.

Egyszerűsített fizetés ellenőrzést használó csomópontok (SPV csomópontok)

Nem minden csomópont tudja a teljes blokkláncot tárolni. Sok bitcoin kliens olyan eszközökön fut, pl. okostelefonokon, tablet-eken vagy beágyazott rendszereken, amelyeknek a hely- és teljesítmény korláta vannak. Az ilyen eszközök egyszerűsített fizetés ellenőrzési módszert (SPV) használnak, amelynél a működéshez nincs szükség a teljes blokklánc tárolására. Ezeket a klienseket SPV klienseknek vagy pehelysúlyú klienseknek nevezzük. Ahogy a bitcoin egyre elterjedtebbé vált, az SPV csomópontok lettek a leggyakrabban előforduló bitcoin csomópontok, különösen a bitcoin pénztárcák esetén.

Az SPV csomópontok csak a blokkok blokkfejeit töltik le, az egyes blokkokba befoglalt tranzakciókat nem. Az így kapott, tranzakciók nélküli blokklánc 1000-szer kisebb a teljes blokkláncnál. Az SPV csomópontok nem tudnak teljes képet alkotni az összes elkölthető UTXO-ról, mivel nem tudnak a hálózatban lévő tranzakciókról. Az SPV csomópontok a tranzakciókat egy kicsit eltérő módon ellenőrizik, és ehhez olyan peereket használnak, melyek kívánság esetén a blokklánc releváns részeiről részleges képet szolgáltatnak.



6-6. ábra: Blokklánc szinkronizálás a peer blokkjainak a letöltésével

Hasonlatképpen: a teljes csomópont olyan, mint egy idegen városban lévő turista, akinek részletes térképe van mindegyik utcáról és címről. Ezzel szemben az SPV csomópont olyan, mint egy idegen városban lévő turista, aki véletlenszerűen idegeneket kérdez meg, hogy merre kell mennie, és csak a főutcát ismeri. Mindkét turista ellenőrizni tudja egy utca meglétét, ha odamegy, de a térkép nélküli turista nem tudja, hogy mi van a mellékutcákban és nem tudja, hogy milyen egyéb utcák léteznek. Ha a térkép nélküli turista a Kossuth út 23-as szám előtt áll, nem tudhatja, hogy vannak-e a városban egyéb „Kossuth út 23” címek, és hogy ez a cím a helyes cím-e. A térkép nélküli turista akkor jár a legjobban, ha megkérdez sok embert, és reménykedik abban, hogy a többségük nem vágja át.

Az egyszerűsített fizetés ellenőrzés a tranzakciókat a blokkláncokon belüli *mélységük* alapján ellenőrzi, nem pedig a *magasságuk* alapján. Míg egy teljes blokkláncot tartalmazó csomópont képes a blokkok és tranzakciók ezreiből álló, időben egészen a genezis blokkig visszanyúló, teljesen ellenőrzött láncok létrehozására, egy SPV csomópont csupán a blokkfejek láncát fogja ellenőrizni, de a tranzakciókét nem, és a blokkfejeket fogja kapcsolatba hozni a kérdéses tranzakcióval.

Például, ha a 300'000-ik blokkban lévő egyik tranzakcióról van szó, egy teljes csomópont a 300'000-ik blokktól egészen a genezis blokkig visszamenően elvégzi az elemzést, és az UTXO-król egy teljes adatbázist épít, vagyis az UTXO elköltetlenségének ellenőrzése révén állapítja meg, hogy a tranzakció érvényes-e vagy sem. Egy SPV csomópont ezzel szemben a tranzakció és az öt tartalmazó blokk közötti kapcsolatot egy *Merkle-út* használatával teremti meg (lásd a „[Merkle-fák](#)” részt). Ezután az SPV csomópont vár mindenkor, amíg a tranzakciót tartalmazó 300'000-ik blokk tetejére további hat blokk nem kerül, és a tranzakciót úgy ellenőrzi, hogy a 300'006 és 300'001 blokkok között megállapítja a tranzakció mélységét. Abból, hogy a hálózat többi csomópontja elfogadta a 300'000-ik blokkot, és azután a megfelelő munkavégzéssel további 6 blokkot hozott létre a 300'000-ik blokk tetején, implicit módon következik, hogy a tranzakció nem kettős költésből származik.

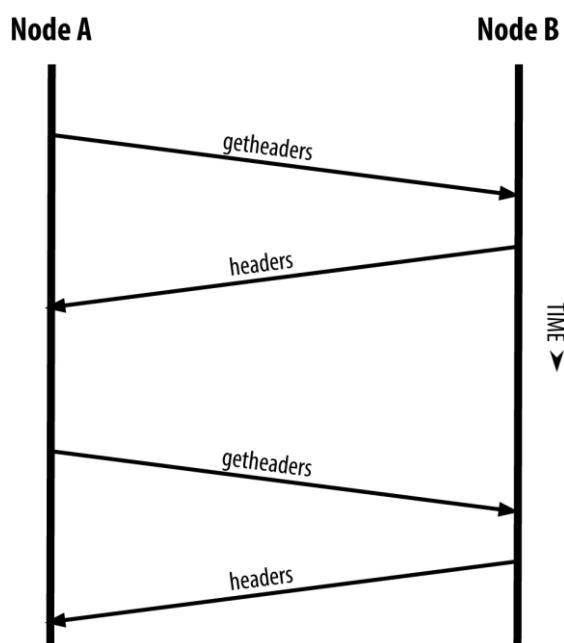
Egy SPV csomóponttal nem lehet elhitetni, hogy egy blokkban létezik egy tranzakció, ha az valójában nem létezik. Az SPV csomópont úgy ellenőrzi egy tranzakció meglétét, hogy lekéri a tranzakció Merkle-újtát, és ellenőrzi a blokkláncban lévő munkabizonyítékokat. De egy tranzakció „rejtve” is maradhat egy SPV csomópont számára. Egy SPV csomópont pontosan meg tudja állapítani, hogy létezik-e egy tranzakció, de azt nem tudja ellenőrizni, hogy nem létezik olyan tranzakció, amely ugyanezt az UTXO-t próbálja duplán elkölni, mert nem rendelkezik az összes tranzakcióval. Az SPV csomópontok ellen minden DoS (denial of service, szolgáltatás megtagadási) támadás vagy kettős költési támadás indítható. Ahhoz, hogy ezt ki lehessen védeni, az SPV csomópontnak számos csomóponttal kell véletlenszerűen kapcsolatba lépnie, így növelni tudja annak a valószínűségét, hogy legalább egy becsületes csomópont van közöttük. Az SPV csomópontok emiatt sérülékenyek a hálózat szétszakadási támadásokkal vagy Sybil támadásokkal szemben, amelyeknél hamis csomópontokra vagy hamis hálózatokra kapcsolódnak, és nem tudják elérni a becsületes csomópontokat vagy a valódi bitcoin hálózatot.

Gyakorlati szempontból a hálózattal szoros kapcsolatban lévő SPV csomópontok elég biztonságosak, és jó kompromisszumot jelentenek az erőforrás felhasználás, a kényelem és a biztonság között. Azoknak, akiknek valóban fontos a biztonság, semmi sem pótolhatja egy teljes blokkláncból álló csomópont üzemeltetését.



A teljes blokkláncból álló csomópont úgy ellenőriz egy tranzakciót, hogy a tranzakció alatti blokkok ezreiből álló lánc vizsgálata révén megbizonyosodik róla, hogy az UTXO valóban elköltetlen, míg az SPV csomópont a blokk fölött lévő néhány blokk segítségével azt ellenőrzi, hogy milyen mélyen van eltemetve a blokk.

A blokkfejeket az SPV csomópontok a nem a `getblocks`, hanem a `getheaders` üzenetekkel kérdezik le. Az a peer, amelyik válaszol, max. 2000 blokkfejet küld el egyetlen headers üzenetben. A folyamat egyébként ugyanolyan, mint amit a teljes csomópontok használnak a teljes blokkok lekérésére. Az SPV csomópontok egy szűrőt is beállítanak a peerekkel létesített kapcsolataiknál, melyek kiszűrik a jövőbeli blokkokat és a peerek által küldött tranzakciókat. Az SPV csomópontok a számukra érdekes tranzakciókat a `getdata` kéréssel kérdezik le. A peer válaszként egy `tx` üzenetet hoz létre, amely a tranzakciót tartalmazza. A 6-7. ábrán a blokkfejlécek szinkronizálása látható.



6-7. ábra: A blokkfejlécek szinkronizálása SPV csomópontok esetén

Mivel az SPV csomópontoknak külön le kell kérdezniük az egyes tranzakciókat ahhoz, hogy ellenőrizni tudják őket, ez veszélyeztetheti a titkosságot. A teljes blokkláncot tartalmazó csomópontokkal szemben (melyek a blokkokban lévő összes tranzakciót tartalmazzák), az SPV csomópontok egyedi adatlekérdezései akaratlanul is felfedhetik, hogy milyen bitcoin címek vannak a pénztárcáikban. Például egy harmadik fél által üzemeltetett megfigyelő hálózat nyilván tudja tartani az SPV pénztárca által kiadott összes kérést, és így kapcsolatba tudja hozni a kérésekben szereplő bitcoin címeket a felhasználó pénztárcájával, ami a privát szféra sérülésével jár.

Az SPV/pehelysúlyú csomópontok bevezetése után nem sokkal a bitcoin fejlesztők az ún. *Bloom szűrőkkel* kívánták megoldani az SPV csomópontok által jelentett adatvédelmi kockázatot. A Bloom szűrők egy valószínűségi szűrőmechanizmus révén lehetővé teszik, hogy az SPV csomópontok csupán a tranzakciók egy részhalmazát fogadják, anélkül, hogy pontosan felfednék, mely címekre kíváncsiak.

Bloom szűrők

A Bloom szűrő egy olyan, valószínűségi kereső szűrő, amellyel egy kívánt minta anélkül írható le, hogy pontosan megadnánk. A Bloom szűrőkkel hatékony módon lehet kifejezni a keresési mintákat, ugyanakkor meg lehet védeni a privát szférát. A Bloom szűrőket az SPV csomópontok arra használják, hogy a peerjeiktől egy adott mintának megfelelő tranzakciókat kérdezzzenek le, de anélkül, hogy pontosan meg kellene adniuk, mely címek érdeklik őket.

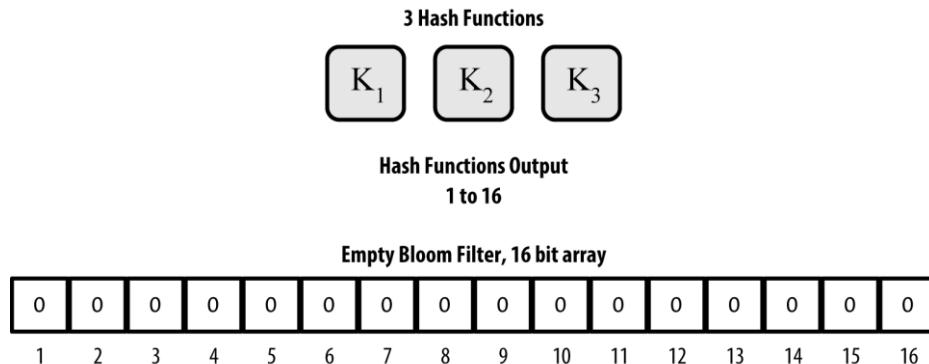
Az előző hasonlatunkban a térkép nélküli turista egy adott cím, pl a „Kossuth út 23” felől érdeklödik. Ha a járókelőktől azt kérdezi, hogy lehet eljutni erre a címre, akaratlanul is elárulja, hogy hová szeretne eljutni. A Bloom szűrő olyan, mint ha azt kérdezné, hogy „Vannak a közelben olyan utcák, melyek neve h-ra végződik?” Egy ilyen kérdés kevesebbet árul el arról, hogy hová szeretne menni, mint a „Kossuth út 23” utáni tudakozódás. Ezzel a módszerrel a turista részletesebben is meg tudja adni a címet, pl. „u-t-h-ra végződik”, vagy kevésbé részletesen, pl. „h-re végződik”. A keresés pontosságának a szabályozása révén a turista több vagy kevesebb információt fed fel, de ennek az az ára, hogy több vagy kevesebb eredményhez jut. Ha egy kevésbé részletes minta után tudakozódik, akkor több lehetséges címet fog kapni és javul az adatvédelem, de az eredmények legtöbbje lényegtelen lesz a számára. Ha egy jobban rögzített minta után tudakozódik, akkor kevesebb eredményt fog kapni, de sérül az adatvédelem.

A Bloom szűrők úgy töltik be ezt a funkciójukat, hogy lehetővé teszik az SPV csomópontok számára, hogy az egyes tranzakcióknál megadott keresési minták a pontosság vagy az adatvédelem irányába mozduljanak el. Egy jobban specifikált Bloom szűrő pontos eredményeket ad, de azon az áron, hogy felfedi a felhasználó pénztárcájában lévő címeket. Egy kevésbé pontos Bloom szűrő több tranzakciót fog eredményként visszaadni, melyek közül sok lényegtelen a csomópont számára, de a csomópont jobb adatvédelmet tud megvalósítani.

Az SPV csomópont a Bloom szűrőt egy „üres” mintával inicializálja. Ebben az állapotában a Bloom szűrő egyetlen egy mintát sem ismer föl. Az SPV csomópont ezután egy listát készít a pénztárcájában lévő címekről, és egy olyan keresési mintát készít, amely megfelel a tranzakciós kimenetekben lévő címeknek. A keresési minta általában egy P2PKH (Pay-to-Public-Key-Hash) zároló script, amely minden olyan tranzakcióban jelen lesz, amely a publikus-kulcs-hashnek (címnek) fizet. Ha az SPV csomópont nyomon követi egy P2SH cím egyenlegét, akkor a keresési minta egy P2SH (Pay-to-Script-Hash) cím lesz. Az SPV csomópont ezután mindegyik keresési mintát megadja a Bloom szűrőnek azzal a céllal, hogy a Bloom szűrő felismerhesse az adott keresési mintázatot, ha az jelen van a tranzakcióban. Végül, a Bloom szűrő elküldi a peernek, és a peer a szűrő segítségével megállapítja, hogy mely tranzakciókat kell elküldenie az SPV csomópontnak.

A Bloom szűrők megvalósítása egy N bites tömbbel, és M db hash függvényel történik. A hash függvények olyanok, hogy a kimenetük mindig 1 és N között van, vagyis a kimenetek a bitek tömbjének megfelelőek. A hash függvényeket determinisztikus módon hozzák létre, ezért egy Bloom szűrőt megvalósító csomópont minden ugyanazokat a hash függvényeket használja, és egy adott bemenet esetén minden ugyanazt az eredményt adja. Különböző hosszságú (N) Bloom szűrő és különböző számú (M) hash függvény választásával a Bloom szűrő különféle pontosságra állítható be, vagyis szabályozható az adatvédelem.

A lenti, 6-8. ábrán egy nagyon kicsi 16 bites tömböt és 3 hash függvényt használunk a Bloom szűrők működésének bemutatására.



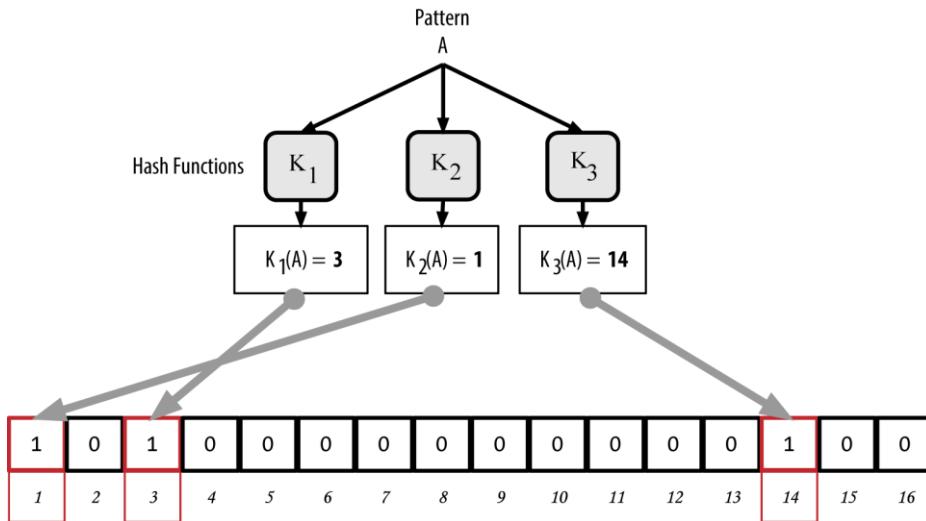
6-8. ábra: Egy egyszerű Bloom szűrő, egy 16 bites tömbbel és 3 hash függvénnyel

(3 hash függvény, hash függvény kimenetek 1-től 16-ig, üres Bloom szűrő, 16 bites tömb)

A Bloom szűrő úgy van inicializálva, hogy a tömb összes bitje nulla. Ha szeretnénk hozzáadni egy mintát a Bloom szűrőhöz, a mintát minden egyes hash függvénnyel összehasheljük. Az első hash függvény a bemenetből egy 1 és N közötti számot állít elő. Az eredménynek megfelelő bitet a tömbben (melynek indexei 1 és N közöttiek) 1-be állítjuk, így rögzítve a hash függvény kimenetét. Ezután a következő hash függvénnyel beállítunk egy másik bitet, és így tovább. Az összes M db hash függvény alkalmazása után egy keresési minta áll elő a Bloom szűrőben, mivel M bitet 0-ról 1-be állítottunk.

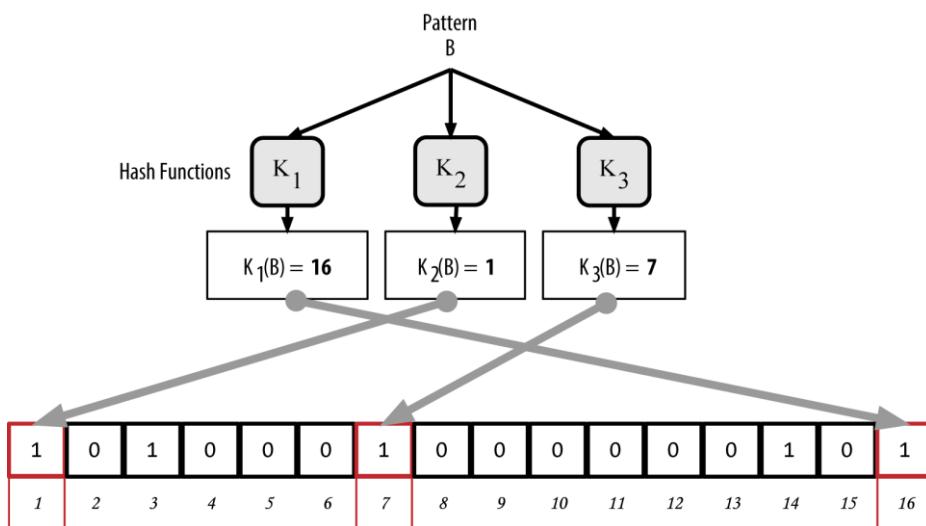
Például, az 6-9. ábrán látható példában a fenti egyszerű Bloom szűrőhöz az „A” keresési mintát adjuk hozzá.

Egy második minta hozzáadása egyszerűen a folyamat megismétlével lehetséges. A mintát minden egyes hash függvénnyel egymás után összehasheljük, és az eredményeket a bitek 1-be állításával rögzítjük. Ahogy a Bloom szűrőt egyre több mintával töltjük föl, valamelyik hash függvény eredménye egybeeshet egy már 1-be állított bittel, ebben az esetben a bitet nem változtatjuk meg. Lényegében, ahogy egyre több mintát rögzítünk ugyanazokban a bitekben, a Bloom szűrő telítetté válik, mert egyre több bitje lesz 1-be állítva, és a szűrő pontossága csökken. A szűrő emiatt tekinthető valószínűségi adatszerkezetnek – egyre több minta hozzáadásakor egyre kevésbé lesz pontos. A pontosság függ a hozzáadott minták számától, a bit tömb méretétől (N), illetve a hash függvények számától (M). Egy nagyobb bit tömbbel és több hash függvénnyel nagyobb pontossággal több minta rögzíthető. Egy kisebb bit tömbbel vagy kevesebb hash függvénnyel kevesebb minta rögzíthető, és kisebb pontosságot kapunk.



6-9. ábra: Az „A” keresési minta hozzáadása az egyszerű Bloom szűrőkhöz

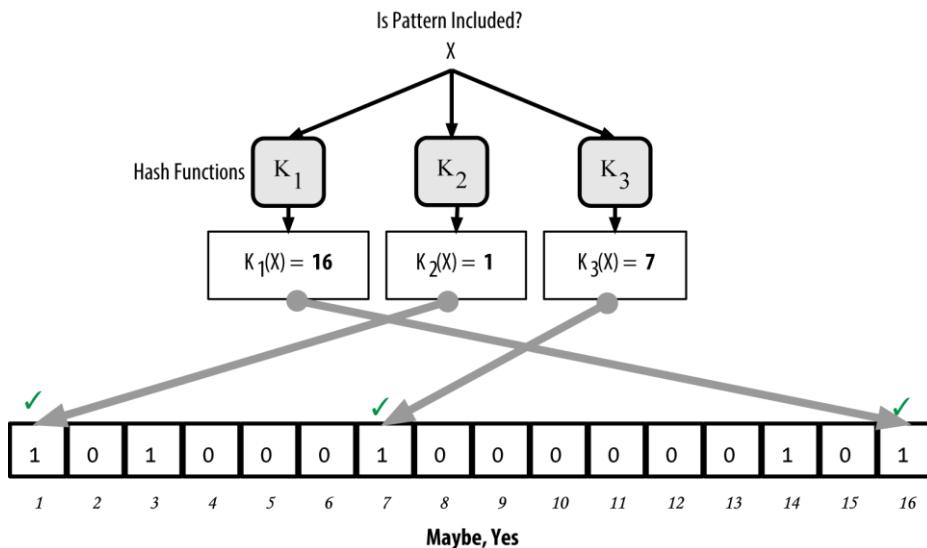
A 6-10. ábrán látható példában az egyszerű Bloom szűrőkhöz egy második keresési mintát adunk, a „B”-t.



6-10. ábra: Egy második keresési minta, a „B” hozzáadása az egyszerű Bloom szűrőkhöz

Ha szeretnénk leellenőrizni, hogy egy minta benne van-e a Bloom szűrőben, akkor hasheljük össze minden egyes hash függvénytel a mintát, és hasonlítsuk össze az így kapott bit mintát a bit tömbbel. Ha a hash függvények által indexelt összes bit 1-ben van, akkor a mintát valószínűleg tartalmazza a Bloom szűrő. Mivel a bitek a különféle minták átfedése miatt is beállításra kerülhetnek, a válasz nem biztos, inkább valószínű. A Bloom szűrónél a pozitív egyezés egyszerűen azt jelenti, hogy „talán igen”.

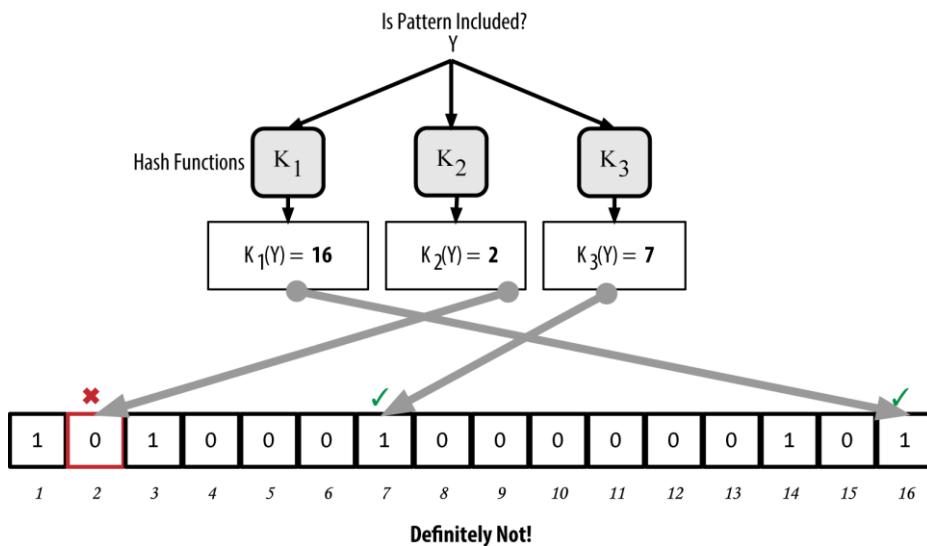
A 6-11. ábrán látható példában azt ellenőrzük, hogy az egyszerű Bloom szűrő tartalmazza-e az „X” mintát. A megfelelő bitek 1-ben vannak, emiatt a minta valószínűleg egyezik:



6-11. ábra: Az „X” minta meglétének ellenőrzése a Bloom szűrőben. Az eredmény pozitív egyezés, ami azt jelenti, hogy „talán”

Ezzel szemben, ha ellenőrizünk egy mintát a Bloom szűrőben, és bármelyik ellenőrzött bit 0, akkor ez azt mutatja, hogy a minta nem volt rögzítve a Bloom szűrőben. A negatív eredmény nem valószínűség, hanem bizonyosság. A Bloom szűrőnél a negatív egyezés egyszerűen azt jelenti, hogy „biztosan nem”.

A 6-12. ábrán látható példában azt ellenőrizzük, hogy az „Y” minta létezik-e az egyszerű Bloom szűrőben. Az egyik szóban forgó bit 0, emiatt a minta biztosan nem illeszkedik.



6-12. ábra: Az „Y” minta létezésének ellenőrzése a Bloom szűrőben. Az eredmény határozott negatív egyezés, ami azt jelenti, hogy „biztosan nem”

A bitcoinban megvalósított Bloom szűrőket a 37. Bitcoin Módosítási Javaslat (Bitcoin Improvement Proposal 37, BIP0037) írja le. Lásd a [B. függeléket](#) vagy a [GitHub](#) webhelyet.

A Bloom szűrők és a leltár frissítések

A peerktől kapott tranzakciók (és az őket tartalmazó blokkok) szűrésére az SPV csomópontok Bloom szűrőket használnak. Az SPV csomópontok egy olyan szűrőt hoznak létre, amely az SPV csomópont pénztárcájában lévő címeknek felel meg. Az SPV csomópont ezután egy `filterload` üzenettel elküldi a kapcsolattartás során használandó Bloom szűrőt a peernek. A szűrő létrejötte után a peer minden egyes tranzakció kimenetét teszteli a Bloom szűrővel. Csak azokat a tranzakciókat küldi el a csomópontnak, amelyeknél a szűrő szerint valamelyik kimenet megfelel a szűrének.

A node-tól kapott `getdata` üzenetre a peerek egy `merkleblock` üzenettel válaszolnak, melyek minden egyes tranzakcióra vonatkozóan csak a filterhez illeszkedő blokkok blokkfejeit tartalmazzák (lásd a „[Merkle-fák](#)” részt). A peerek ezt követően `tx` üzeneteket is küldenek, melyek a filterhez illeszkedő tranzakciókat tartalmazzák.

A Bloom szűrőt beállító csomópont menet közben további mintákkal bővítheti a szűrőt, ehhez a `filteradd` üzenetet kell elküldenie. Mivel a Bloom szűrőből nem lehet mintát eltávolítani, ezért ha valamelyik mintára már nincs szükség, akkor a csomópontnak először egy `filterclear` üzenettel törölnie kell a Bloom szűrőt, majd egy újabb Bloom szűrőt kell küldenie.

Tranzakció poolok

A megerősítetlen tranzakciókból a bitcoin hálózat majdnem mindegyik csomópontja egy listát kepez, az ún. *memory poolt* vagy *tranzakció poolt*. A csomópontok ennek az alapján követik nyomon azokat a tranzakciókat, melyeket a hálózat már ismer, de még nincsenek a blokkláncba foglalva. Például egy olyan csomópont, amelyik pénztárcát is tartalmaz, a tranzakció poolt arra használja, hogy nyomon kövesse a hálózaton át a pénztárcába érkező, de még megerősítetlen befizetéseket.

Az tranzakciókat a csomópont a beérkezésük és ellenőrzésük után a tranzakció poolba helyezi, majd a hálózati szétterjedés érdekében a szomszédos csomópontoknak továbbítja.

Némelyik implementációjában egy külön lista szolgál az elárvult tranzakciók nyilvántartására. Ha a tranzakció bemenetei olyan tranzakcióra hivatkoznak, amely még nem ismert, pl. hiányzik a szülő, akkor az elárvult tranzakció átmenetileg az elárvult tranzakciók pooljában tárolódik, amíg meg nem érkezik a szülő tranzakció.

Ha a tranzakció poolba bekerül egy tranzakció, akkor a csomópont ellenőrzi, hogy az elárvult tranzakciók közül nem hivatkozik-e valamelyik a most bekerült tranzakció valamelyik kimenetére (nem gyereke-e ennek a tranzakciónak), majd ellenőrzi az illeszkedő árva tranzakciókat. Ha a tranzakció érvényes, akkor eltávolítja az elárvult tranzakciók közül, és hozzáadja a tranzakciók pooljához, vagyis kiegészíti a szülő tranzakcióval elkezdett láncot. Az újonnan hozzáadott, már nem árva tranzakcióra vonatkozóan, a folyamatot rekurzív módon megismétli, és további leszármazottakat keres, amíg vannak ilyenek. Ennek a folyamatnak a révén egy szülő tranzakció beérkezése a tőle függő tranzakciók egész láncának rekonstruálását váltja ki, és az árva tranzakciókat ismét egyesíti a szüleikkel.

Sem a tranzakciók, sem az árva tranzakciók poolját (ha van ilyen) nem tárolják diszken. Ezek csak a helyi memóriában léteznek, és dinamikusan, a bejövő hálózati üzenetek alapján kerülnek feltöltésre. Egy csomópont elindulásakor minden két pool üres, és fokozatosan, az új tranzakciók beérkezésekor kerül feltöltésre.

A bitcoin kliens némelyik implementációja egy UTXO adatbázist vagy UTXO poolt is tartalmaz, amely a blokkláncban lévő elkötött kimenetek halmazának felel meg. Noha az „UTXO pool” hasonlónak tűnik a tranzakció poolhoz, de más adathalmazt jelent. A tranzakciók és az elárult tranzakciók pooljával szemben az UTXO pool nem üresen indul, hanem elkötött tranzakció kimenetek millióit tartalmazza, melyek 2009-ig nyúlnak vissza. Az UTXO pool vagy a helyi tárban van, vagy a háttértár egy indexelt adatbázis táblája alkotja.

Míg a tranzakciók és árva tranzakciók poolja a helyi csomóponttól függ, és csomópontról csomópontra jelentősen változhat, attól függően, hogy a csomópont mikor indult vagy mikor indult újra, az UTXO pool a hálózatban kialakult konszenzusnak felel meg, és emiatt csak nagyon kicsiny eltérések lehetségesek az egyes csomópontok között. Ezen túlmenően a tranzakciók és árva tranzakciók pooljában csak megerősítetlen tranzakciók lehetnek, míg az UTXO pool csak megerősített kimeneteket tartalmazhat.

Figyelmeztető üzenetek

A figyelmeztető üzenetek ritkán használatosak, de a funkció a legtöbb csomópontban mégis meg van valósítva. A figyelmeztető üzenetek jelentik a bitcoin „vészjelző rendszerét”, mellyel a bitcoin fejlesztők vészhelyzetben szöveges üzenetet tudnak az összes bitcoin csomópontnak küldeni. Ezt a jellemzőt azért lett megvalósítva, hogy a bitcoin core kliens fejlesztő csapat az összes bitcoin felhasználót értesíteni tudja a bitcoin hálózatban felmerült súlyos problémákról, például egy kritikus hibáról, amely felhasználói beavatkozást igényel. A jelzőrendszeret csak néhányszor használták, ezek közül a legnevezetesebb eset 2013-ban volt, mikor egy kritikus adatbázis hiba miatt elágazás történt a bitcoin blokkláncban.

A figyelmeztető üzeneteket az `alert` üzenettel lehet továbbítani. A figyelmeztető üzenetnek számos mezője van. Ezek a következők:

ID

A figyelmeztető üzenet azonosítója, amivel elkerülhető a figyelmeztetés megkettőzödése

Expiration

a figyelmeztetés lejáratú ideje

RelayUntil

A figyelmeztetés relézési ideje, ami után már nem szabad továbbadni

MinVer, MaxVer

Azoknak a bitcoin protokoll változatoknak a tartománya, amelyekre ez a figyelmeztetés vonatkozik

subVer

Az a kliens szoftver alverzió, amelyre ez a figyelmeztetés vonatkozik

Priority

A figyelmeztetés prioritási szintje, jelenleg nem használt

A figyelmeztetések egy publikus kulccsal vannak aláírva. A publikus kulcs hozzá tartozó privát kulcsot a fejlesztő csapat néhány kiválasztott tagja birtokolja. A digitális aláírás biztosítja, hogy a hálózat ne továbbíthasson hamis figyelmeztetéseket.

Ha egy csomópontra figyelmeztető üzenet érkezik, akkor a csomópont ellenőrzi az üzenetet, többek között a lejáratú időt, és továbbítja az összes peerjének, így biztosítva az egész hálózatban az üzenet gyors szétterjedését. A csomópontok a figyelmeztetés továbbításán túlmenően rendelkezhetnek egy felhasználói interfész funkcióval, amely az üzenetet megjeleníti a felhasználó számára.

A Bitcoin Core kliensben a figyelmeztetéshez az `-alertnotify` parancssori opció tartozik. Ezzel lehet megadni, hogy milyen parancs fusson le, ha figyelmeztető üzenetet kapunk. A figyelmeztető üzenet paraméterként van megadva az `alertnotify` parancsban. Az `alertnotify` parancsot a leggyakrabban úgy állítják be, hogy a figyelmeztető üzenetet tartalmát egy e-mail üzenetben küldje el a csomópont adminisztrátorának. A figyelmeztetés a grafikus felhasználói felületen (bitcoin-QT) egy felugró ablak formájában is megjelenik, ha fut a kliens.

A bitcoin protokoll egyéb implementációiban a figyelmeztetés kezelése eltérő módon történhet. Sok hardverbe integrált bányász rendszer a figyelmeztető üzenet funkciót nem valósítja meg, mivel ezeknek a rendszereknek nincs felhasználói felületük. Erősen javallott, hogy az ilyen bányász rendszereket futtató bányászok a bányatársaság üzemeltetőjénél „fizessék elő” a figyelmeztetésekre, vagy csak a figyelmeztetések miatt futtassanak egy pehelysúlyú csomópontot.

7. A blokklánc

Bevezetés

A blokklánc, mint adatstruktúra a tranzakciós blokkok egy rendezett láncolt listája. A blokkláncot tárolhatjuk sima fájlként, vagy egy egyszerű adatbázisban. A Bitcoin Core kliens a blokklánc metaadatait a Google LevelDB adatbázisának segítségével tárolja. minden blokk visszafelé láncolt, azaz mindegyik visszahivatkozik a blokklánc előző blokkjára. A blokkláncot gyakran egy függőleges halomként ábrázolják, ahol az első blokk minden a halom alján szerepel. A blokkok ilyen megjelenítése miatt használunk olyan kifejezéseket, mint például a első blokktól számított távolság megjelölésére a „magasság”, vagy a legutóbbi hozzáadott blokk esetén a „csúcs”.

A blokklánc minden blokkját egy hash azonosítja, amely a blokk fejlécére alkalmazott SHA256 kriptográfiai hash algoritmussal áll elő. A blokkfejlécben lévő „előző blokk hashe” mező segítségével minden blokk hivatkozik a megelőző blokkra is, amelyet *szülő blokknak* szokás nevezni. Más szóval: a saját fejlécén belül minden blokk tartalmazza a szülő blokkjának a hashét is. Ekképp a hashek sorozatából, melyek minden blokkot összekapcsolnak a szülőjével, egy lánc jön létre, amely minden esetben visszavezet a legeslegelsőként generált blokkhoz, az úgynevezett *geneesis blokkhoz*.

Bár egy blokknak csak egy szülője van, átmenetileg akár több gyereke is lehet. minden gyerek ugyanarra a szülő blokkra hivatkozik, illetve minden gyerek azonos szülő hasht tartalmaz az „előző blokk hashe” mezőjében. Egy blokklánc elágazásnál egyszerre több gyerek is létrejöhét. Az elágazás egy átmeneti állapot, amely akkor jelentkezik, amikor két bányász szinte egy időben fedez fel két különböző blokkot (lásd „[Blokklánc elágazások](#)”). Végül csak az egyik gyerek-blokk fog a blokklánc részévé válni, az „elágazás” pedig megoldódik. Egy blokknak tehát akár több gyereke is lehet, azonban minden blokknak csak egy szülője van. Ez azért van így, mert egy blokknak csak egyetlen olyan „előző blokk hashe” mezője van, amely az egyedüli szülőjére hivatkozik.

Az „előző blokk hashe” mező a blokkfejlécben van, és emiatt befolyásolja az *aktuális* blokk hashét is. Ha a szülő identitása megváltozik, akkor a gyermek identitása is megváltozik. Ha a szülő bármilyen módon megváltozik, akkor a hashe is megváltozik. A megváltozott szülő hashének változása pedig elkerülhetetlenné teszi a gyerekben lévő „előző blokk hashének” a változását is. Emiatt a gyerek hashe is változni fog, ez pedig változást eredményez az unokában lévő hivatkozásban is, tehát megváltozik az unoka is, és így tovább. Ez a kaszkád hatás biztosítja, hogy ha egy blokkot több generáció követ, akkor a blokkot csak úgy lehet megváltoztatni, ha az összes rákövetkező blokkot is újraszámításra kötelezzük. Mivel egy ilyen újraszámítás hatalmas számítási kapacitást igényelne, a blokkok hosszú láncolatának a megléte megváltoztathatatlaná teszi a blokklánc mélyének történelemét, ez pedig a bitcoin biztonságának a kulcsfontosságú eleme.

A blokkláncot valahogy úgy lehet elképzelni, mint a geológiai képződmények rétegeit, vagy mint egy gleccser-mag mintát. A felületi rétegek az évszakoknak megfelelően változhatnak, vagy akár el is tűnhetnek még azelőtt, hogy idejük lett volna sorba rendeződni. Ha viszont pár centivel mélyebbre ásunk, akkor azt tapasztaljuk, hogy a geológiai rétegek egyre stabilabbak. Amikor néhány száz méter mélyen vizsgálódunk, akkor tulajdonképpen a múlt pillanatfelvételét szemléljük, mely évezredeken, vagy évmilliókon át zavartalan maradt. A

blokklánc néhány legutóbbi blokkja módosulhat, ha egy elágazás miatt újra kell számolni a láncot. A legfelső hat blokk olyan, mint a termőtalaj néhány centimétere. De ha egyre mélyebbre ásunk a blokkláncban, pl. 6 blokkon túl, akkor egyre valósánélkülebb lesz, hogy egy blokk megváltozik. 100 blokk után már olyan szintű a stabilitás, hogy az úgynevezett „coinbase” tranzakció (az a tranzakció, amely az újonnan kibányászott érméket tartalmazza) el is költhető. Néhány ezer egymásra rétegződött blokk (kb. egy hónap), és a blokklánc már a történelem része, azaz gyakorlatilag soha nem változik meg. Noha a protokoll lehetővé teszi, hogy a láncot bármikor egy hosszabb lánc váltsa fel, és bármelyik blokk esetén van rá esély, hogy a blokk visszafordításra kerül, az ilyen események valószínűsége az idő múlásával egyre kisebbé válik, amíg infinitézimálissá nem lesz.

Egy blokk szerkezete

A blokk egy olyan adattároló adatszerkezet, amely összegyűjt a tranzakciókat, hogy azokat a nyilvános főkönyvbé, azaz a blokkláncba foglalja. A blokk egy meta-adatokat tartalmazó fejlécből, és az azt követő, a tranzakciók hosszú sorát tartalmazó listából áll, ez utóbbi határozza meg a blokk méretét. A blokkfejléc mérete 80 bájt, míg egy átlagos tranzakcióé minimum 250 bájt. Egy átlagos blokk több mint 500 tranzakciót tartalmaz, így egy teljes blokk, az összes tranzakcióval együtt 1000-szer nagyobb, mint a blokkfejléc. A 7-1. táblázat egy blokk szerkezetét mutatja.

7-1. táblázat: Egy blokk szerkezete

Méret	Mező	Leírás
4 bájt	Blokkméret	A blokk mérete bájtokban, ezt a mezőt követően
80 bájt	Blokkfejléc	A blokkfejléc mezői (lásd alább)
1-9 bájt (VarInt)	Tranzakció számláló	Hány tranzakció jön ezután
változó	Tranzakciók	A blokkban rögzített tranzakciók

A blokkfejléc

A blokkfejléc háromféle blokk meta-adatból áll. Először is, van egy hivatkozás az előző blokk hashére, amely a blokklánc aktuális blokkját köti össze a megelőző blokkal. A második meta-adat halmaz az úgynevezett „difficulty” (nehézség), illetve az időbélyegző és számláló, ezek a bányászattal függnek össze, amint azt a [8. fejezet](#) részletezi. A meta-adatok harmadik része a Merkle-fa gyökér, ami egy olyan adatstruktúra, amely hatékonyan összefoglalja a blokk tranzakcióját. A 7-2. táblázat a blokkfejléc felépítését mutatja.

7-2. táblázat: A blokkfej szerkezete

Méret	Mező	Leírás
4 bájt	Version	A szoftver/protokoll változásokat nyomon követő verziószám
32 bájt	Previous Block Hash	Hivatkozás a blokklánc előző (szülő) blokkjának a hashére
32 bájt	Merkle Root	A blokk tranzakcióihoz tartozó Merkle-fa gyökerének a hashe
4 bájt	Timestamp	Időbényeg: hozzávetőleg mikor jött létre a blokk (a Unix kezdőidő óta eltelt másodpercek)
4 bájt	Difficulty Target	A munkabizonyíték algoritmus által megkövetelt cél nehézségi szint
4 bájt	Nonce	A munkabizonyíték algoritmus által használt számláló

A Nonce, a nehézségi szint és az időbényeg a bányászat során használatos, ezeket a [8. fejezet](#) tárgyalja részletesebben.

Blokk azonosítók: Blokkfejléc, Hash, és Blokk magasság

Egy blokk elsődleges azonosítója a blokk kriptográfiai hashe, azaz a az digitális ujjlenyomata, amely oly módon jön létre, hogy a blokkfejléc az SHA256 algoritmussal kétszer hashelésre kerül. Az így kapott 32 bájtos hash az úgynevezett *blokk hash*, vagy pontosabban a *blokkfejléc hash*, mivel a kiszámításához csak a blokkfejlécre van szükség. Például a legelső legenerált bitcoin blokk blokk hashe a

000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f érték. A blokk hash egyértelmű módon azonosítja a blokkot. Bárminely csomópont képes arra, hogy a többiből függetlenül előállítsa. Ehhez csupán a blokkfejlécet kell hashelnie.

Meg kell jegyezni, hogy a blokk belső adatstruktúrája ténylegesen nem tartalmazza a blokk hasht, sem akkor, amikor a blokk átvitelre kerül a hálózaton, sem akkor, amikor a blokklánc részeként, egy csomópontron tárolásra kerül. Valójában az történik, hogy mikor a blokk a hálózatból megérkezik, minden egyes csomópont újraszámítja a blokk hasht. A blokk hash a blokk meta-adatainak a részeként egy külön adatbázis táblázatban is tárolható. Ez megkönnyíti a blokkok indexelését és meggyorsítja a blokkok merevlemezről történő visszakeresését.

Egy blokk azonosítására egy másik módszer az, hogy megadjuk a blokkláncon belüli pozícióját, az úgynevezett *blokk magasságát*. A legelőször legenerált blokknak a blokk magassága 0 (nulla), ez ugyanaz a blokk, amire az imént a

000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f blokk hashsel hivatkoztunk. A blokk tehát kétféleképpen azonosítható: vagy a blokk hashsel, vagy a blokk magassággal. minden következő blokk az előző „tetejéhez” adódik hozzá, vagyis ahhoz, amelyik éppen a blokklánc „legmagasabb” blokkja volt, valahogy úgy, ahogy dobozokat rakunk egymás tetejére. A blokk magasság 2014. január 1-én körülbelül 278'000 volt, ami azt jelenti, hogy a megjelölt időpontig 278'000 blokk halmozódott fel a 2009 januárjában létrehozott első blokkon.

A blokk hash-sel ellentétben, a blokk magasság nem egy egyedi azonosító. Míg egy blokknak minden van egy adott és állandó blokk magassága, addig ennek a fordítottja nem igaz - a blokk magassághoz nem csak egy blokkot tartozhat. Két vagy akár több blokknak is azonos lehet a blokk magassága. Ezek a blokkok a blokklánc ugyanazon pozíciójáért versenyeznek. Ennek a menetéről a „[Blokklánc elágazások](#)” részben írunk részletesebben. A blokk magasság szintén nem része a blokk adatstruktúrájának; tehát nem kerül a blokkon belül tárolásra. minden csomópont dinamikusan azonosítja az adott blokk blokkláncon belüli helyzetét (magasságát), amikor azt a bitcoin hálózattól megkapja. A gyorsabb visszakeresés érdekében a blokk magasság is tárolható meta-adatként egy indexelt adatbázis táblázatban.



Egy blokk *blokk hashe* minden esetben egyedi módon, egyetlen egy blokkot azonosít. A blokknak minden van egy adott blokk magassága is. De egy adott blokk magasság nem minden esetben azonosít csak egyetlen blokkot. Gyakran előfordul, hogy két vagy akár több blokk is ugyanazért az blokkláncbeli pozícióért versenyez.

A „Genezis” Blokk

A blokklánc első blokkját, amely 2009-ben jött létre, „Genezis blokk”-nak nevezzük. Ez minden blokklánc blokkjának a „közös őse”, ami azt jelenti, hogy ha a blokklánc mentén, bármelyik blokktól, elkezdünk időben visszafelé haladni, akkor végül a „Genezis blokk”-hoz fogunk elérkezni.

Minden csomópont minden egy minimum egy blokkból álló blokklánckal indul, mivel a Genezis blokk megváltoztathatatlan módon, statikusan be van kódolva a bitcoin kliens szoftverbe. minden csomópont minden „tudja”, hogy mi a Genezis blokk hashe és milyen a szerkezete, mikor jött létre, és milyen tranzakciókat tartalmaz. Így minden csomópont rendelkezik a blokklánc kiindulópontjával, azaz van olyan biztonságos „gyökere”, amelyből kiépíthető egy megbízható blokklánc.

A a Bitcoin Core kliensben a statikusan kódolt Genezis blokk a chainparams.cpp állományban van: [chainparams.cpp](#).

A Genezis blokk azonosító hashe:

000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

Ezt a blokk hasht tetszőleges blokklánc vizsgáló web lapon megkereshetjük, például a blockchain.info-n is, az eredményként kapott oldal ennek a blokknak a tartalmát írja le, és mellesleg az URL-ben a hasht is tartalmazza:

<https://blockchain.info/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f>

<https://blockexplorer.com/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f>

A Bitcoin Core referencia kliensét használva a parancssorban:

```
$ bitcoind getblock  
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f  
{  
    "hash" :  
"000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",  
    "confirmations" : 308321,  
    "size" : 285,  
    "height" : 0,  
    "version" : 1,  
    "merkleroot" :  
"4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",  
    "tx" : [  
        "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"  
    ],  
    "time" : 1231006505,  
    "nonce" : 2083236893,  
    "bits" : "1d00ffff",  
    "difficulty" : 1.00000000,  
    "nextblockhash" :  
"0000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"  
}
```

A Genezis blokk tartalmaz egy rejtett üzenetet is. A coinbase tranzakció bemenete az alábbi szöveget is tartalmazza: „The Times 03/Jan/2009 Chancellor on brink of second bailout for banks” („The Times, 2009. jan. 3., A pénzügyminiszter hajlik a bankok második kimentésére”). Ez az üzenet bizonyítja, hogy mikor jött létre a legelső blokk, mivel a *The Times* brit újság akkorai főcímére utal. A konkrét szöveget ironikus tréfaként is felfoghatjuk, hiszen felhívja figyelmünket egy önálló/független monetáris rendszer fontosságára, illetve arra is, hogy a Bitcoin a példátlan világméretű pénzügyi válsággal egy időben indult útjára. Az üzenetet a Bitcoin megalkotója, Satoshi Nakamoto rejtette el az első blokkban.

A blokkok blokklánccá történő összekapcsolása

A Bitcoin csomópontnak a blokkláncból egy helyi példányuk van, amely a Genezis blokkal indul. A blokklánc helyi másolata folyamatosan frissül, mivel új blokkok képződnek, melyek bővítik a láncot. Ha a csomópontra a hálózatból egy blokk érkezik, akkor a csomópont először ellenőrzi a blokkot, majd hozzákapcsolja a blokkot a meglévő blokklánchoz. Az összekapcsoláshoz a csomópont megvizsgálja a beérkező blokk blokkfejlécét, és megkeresi benne az „előző blokk hashét”.

Tegyük fel például, hogy egy csomópontnak 277'314 blokkja van a helyi blokklánc másolatában. A csomópont által ismert utolsó blokk a 277'314, melyben a blokkfejléc hashe: 000000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249.

A bitcoin csomópont ezután kap egy új blokkot a hálózattól, amelyet az alábbiak szerint értelmez:

```
{  
    "size" : 43560,  
    "version" : 2,  
    "previousblockhash" :  
        "000000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",  
    "merkleroot" :
```

```

    "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
    "time" : 1388185038,
    "difficulty" : 1180923195.25802612,
    "nonce" : 4215469401,
    "tx" : [
        "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",
        "#[... sok egyéb, itt fel nem tüntetett tranzakció ...]
        "05cf38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"
    ]
}

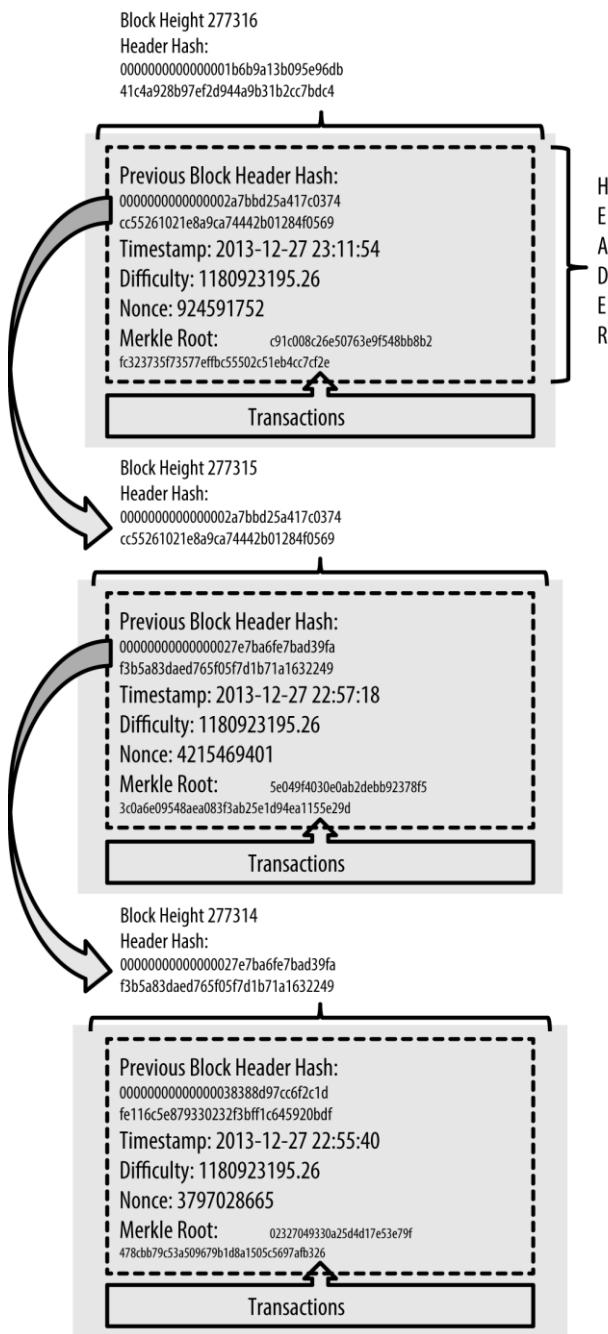
```

Az új blokk vizsgálatakor a csomópont megtalálja az „előző blokk hashe” mezőt, amely a szülő blokk hashét tartalmazza. Ezt a hasht a csomópont ismeri, hiszen ez a lánc utolsó, 277'314-ik blokkjáé. Következésképpen, az új blokk a lánc utolsó blokkjának a gyereke, és kiterjeszhető vele a már meglévő blokklánc. A csomópont az új blokkot a lánc végéhez adja hozzá, 277'315-re növelte a blokklánc magasságát. A 7-1. ábrán egy három blokkból álló lánc látható, a blokkokat a `previousblockhash` mezőben lévő hivatkozások kapcsolják össze.

Merkle-fák

A bitcoin blokklánc minden egyes blokkjában van egy mező, amely egy *Merkle-fa* segítségével a blokkhoz tartozó összes tranzakciót összefoglalja.

A *Merkle-fa*, vagy más néven *bináris hash-fa* egy olyan adatstruktúra, amely nagy adathalmazok hatékony összefoglalására, illetve sértetlenségének az ellenőrzésére használható. A Merkle-fák kriptográfiai hasheket tartalmazó bináris fák. A „fa” kifejezés a számítástechnikában egy elágazó adatszerkezet leírására használatos, ám ezek a fák általában fejjel lefelé vannak ábrázolva, azaz „gyökerük” van az ábra tetején, míg „ levelei” a diagram alján, amint azt a következő példákban is láthatjuk.



7-1. ábra: Az előző blokkfejléc hashére hivatkozó, blokkláncba kapcsolt blokkok

A bitcoinnál a Merkle-fák arra a célra szolgálnak, hogy összefoglalják egy blokk összes tranzakcióját, vagyis a blokkban szereplő tranzakciókból egy átfogó digitális újlenyomatot hozzanak létre, s íly módon egy nagyon hatékony eljárást biztosítanak annak az ellenőrzésére, hogy egy adott tranzakció valóban szerepel-e a blokkban. A Merkle-fa csomópont párok rekurzív hashelésével épül fel, egészen addig, amíg már csak egy hash, az úgynevezett *gyökér* vagy *Merkle-gyökér* marad. A bitcoin esetében a Merkle-fáknál használt kriptográfiai hash algoritmus a kétszer egymás után alkalmazott SHA256, vagyis a dupla SHA256 néven is ismert algoritmus.

Ha N db adatelem egy Merkle-fában van összefoglalva, akkor legfeljebb $2 * \log_2(N)$ számítással ellenőrizhetjük, hogy egy adott adatelem valóban megtalálható-e a fában, emiatt ez az adatstruktúra rendkívül hatékony.

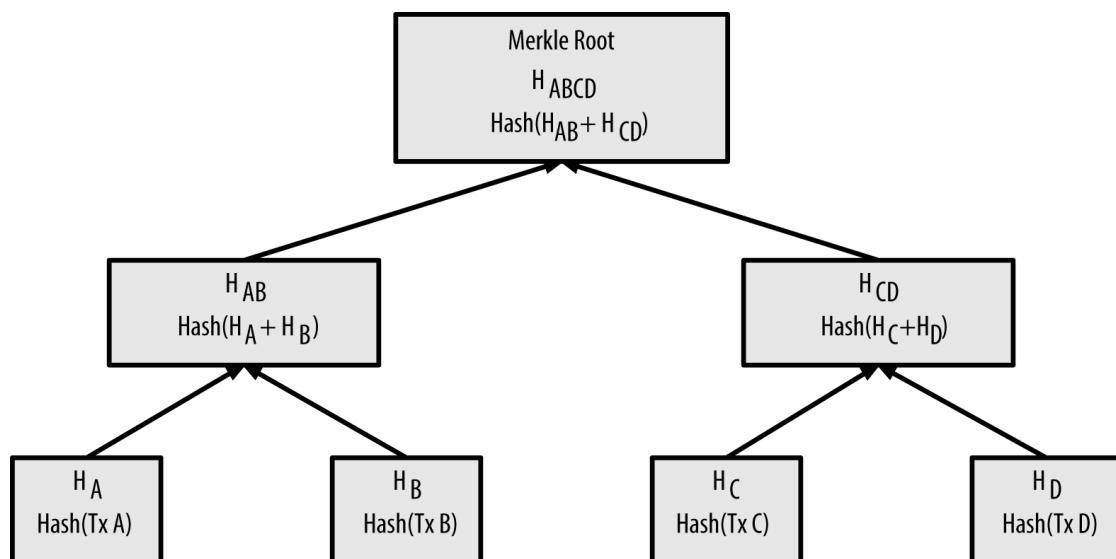
A Merkle-fa alulról felfelé épül. Az alábbi példában négy tranzakcióval kezdjük a munkát, A, B, C és D-vel, amelyek a Merkle-fa leveleit alkotják, ahogy ezt a 7-2. ábra is mutatja. A Merkle-fa nem tárolja a tranzakciókat, inkább hasheli azok adatait, és az így kapott hasht tárolja minden egyes levél-csomópontban, H_A , H_B , H_C és H_D -ként:

$$H_A = \text{SHA256}(\text{SHA256}(\text{Transaction A}))$$

Az egymás utáni levél-csomópontokat ezután a szülő-csomópont foglalja össze úgy, hogy összekapcsolja a két hasht, majd hasheli őket. Például a H_{AB} szülő-csomópont úgy áll elő, hogy a két 32 bájtos hash gyerek összefűzésével egy 64 bájtos string áll elő. Ennek a stringnek a duplán hashelésével áll elő a szülő-csomópont hashe:

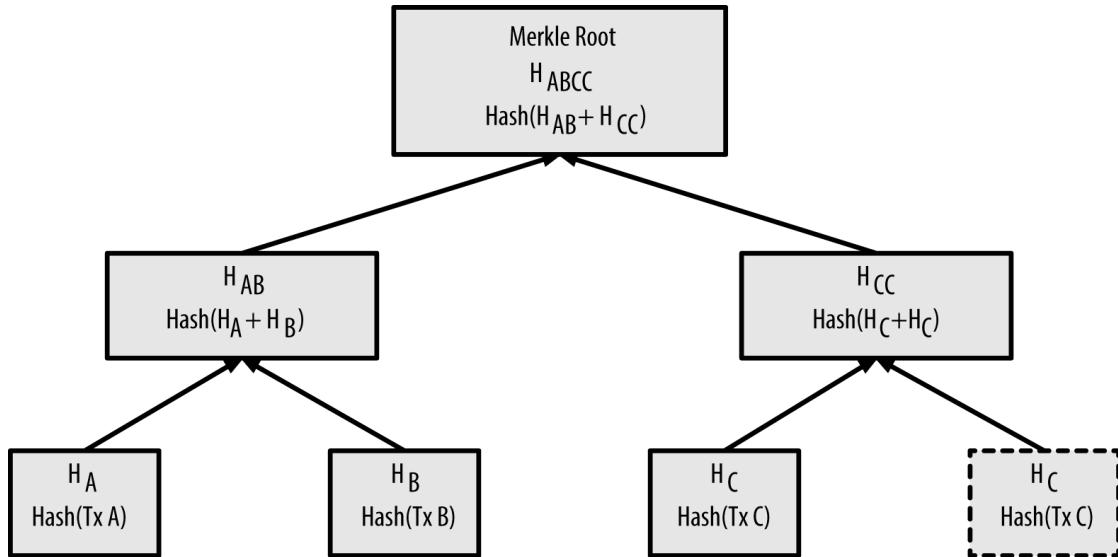
$$H_{AB} = \text{SHA256}(\text{SHA256}(H_A + H_B))$$

A folyamat mindaddig folytatódik, amíg már csak egyetlen csomópont lesz legfelül, az ún. Merkle-gyökér. A blokkfejlécben ez a 32 bájtos hash kerül tárolásra, amely minden tranzakció adatait összefoglalja.



7-2. ábra: Egy Merkle-fa csomópontjainak a kiszámítása

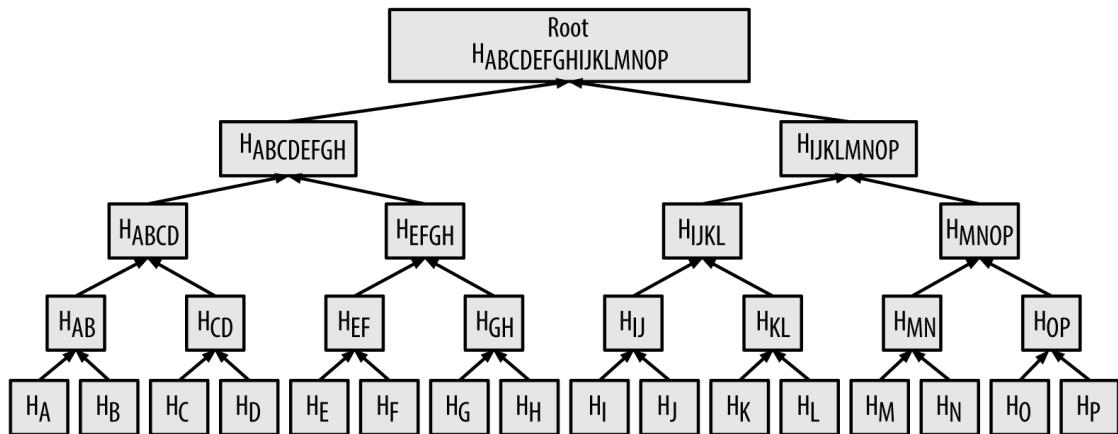
Mivel a Merkle-fa egy bináris fa, így páros számú levél csomópontra van szüksége. Ha páratlan számú tranzakció összefoglalására van szükség, akkor az utolsó tranzakció hashe duplikálódik, így páros számú levél csomópont lesz. Ez az ún. *kiegyensúlyozott fa*. Ezt szemlélteti az alábbi 7-3. ábra, ahol a C tranzakció lett megkettőzve:



7-3. ábra: Egy adatelem megkettőzésével páros számú adat elem áll elő

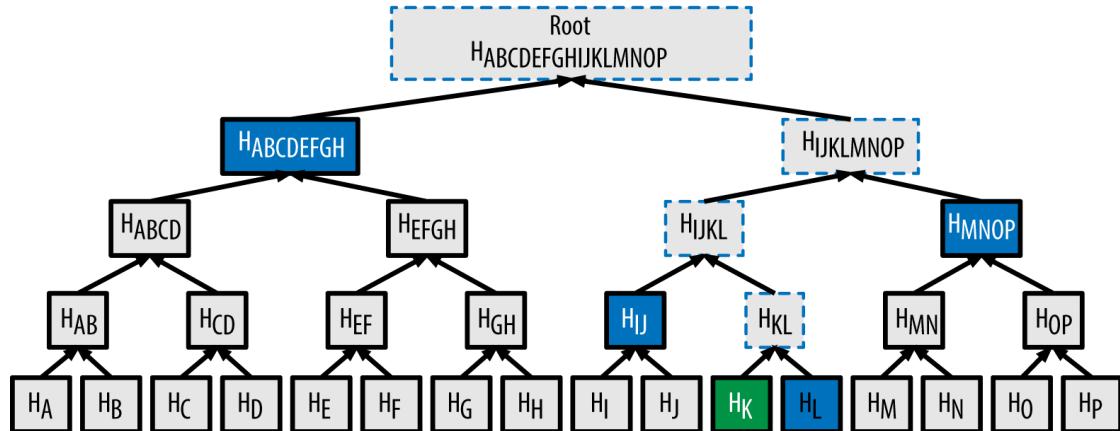
A négy tranzakcióból fát építő módszer általánosítható bármilyen méretű fa felépítésére. A bitcoinnál gyakori, hogy egy blokk több száz, vagy akár több ezer tranzakciót is tartalmaz. Ezek pontosan ugyanilyen módon kerülnek összefoglalásra, és egyetlen 32 bájtos adatelem, a Merkle-gyökér áll elő belőlük. Az alábbi 7-4. ábrán egy 16 tranzakcióból felépülő fa látható. Meg kell jegyezni, hogy az ábrán a gyökér ugyan nagyobbnak látszik, mint levél csomópontjai, méretük azonban pontosan ugyanaz, minden 32 bájt. Legyen szó egy blokk egyetlen egy, vagy akár több százezer tranzakciójáról, a Merkle-gyökér minden 32 bájtból foglalja össze őket.

Ha bizonyítani szeretnénk, hogy egy adott tranzakció szerepel egy blokkban, akkor csupán arra van szükség, hogy a csomópont egy $\log_2(N)$ 32 bájtos hasht hozzon létre, amely hitelesítési útvonalat vagy úgynevezett *Merkle-utat* alkot, és összeköti az adott tranzakciót a fa gyökérével. Ez különösen akkor fontos, amikor a tranzakciók száma nagy, mivel a tranzakciók számának 2-es alapú logaritmusa sokkal lassabban növekszik. Íly módon a Bitcoin csomópontok képesek arra, hogy tíz vagy tizenkét hashból (320-384 bájtból) álló hatékony útvonalat hozzanak létre, amellyel bizonyítható egy tranzakció blokkon belüli jelenléte, akár egy megabajt méretű blokk többezer tranzakciójába között is.



7-4. ábra: Nagyszámú adatelemet összefoglaló Merkle-fa

Az alábbi 7-5. ábrán látható példában a csomópont úgy tudja bizonyítani, hogy a blokk tartalmazza a K tranzakció-t, hogy csupán négy bájt hash hosszúságú (összesen 128 bájtos) Merkle-utat hoz létre. Az útvonal a következő négy hashből áll: H_L , H_{IJ} , H_{MNOP} és $H_{ABCDEFGH}$. A hitelesítési útvonalat alkotó négy hash érték ismeretében bármelyik csomópont bizonyítani tudja, hogy H_K (ami az ábrán zölddel szerepel) benne van a Merkle-fában. Ehhez további három hash-párt kell kiszámítania: H_{KL} , H_{IJKL} és $H_{IJKLMNOP}$, melyek elvezetik a Merkle-gyökérhez (az ábrán ezt pontozott vonal szemlélteti).



7-5. ábra: Merkle-út egy adatelem jelenlétének a bizonyítására

A 7-1. példában szereplő program egy Merkle-fa létrehozásának folyamatát szemlélteti, a levél csomópontokból a gyökér irányába haladva. A példa a libbitcoin könyvtárat és néhány segédfüggvényt használ.

7-1. példa: Egy Merkle-fa felépítése

```

$ cat merkle.cpp
#include <bitcoin/bitcoin.hpp>

bc::hash_digest create_merkle(bc::hash_list& merkle)
{
    // Stop if hash list is empty.
    if (merkle.empty())
        return bc::null_hash;
    else if (merkle.size() == 1)
        return merkle[0];

    // While there is more than 1 hash in the list, keep looping...
    while (merkle.size() > 1)
    {
        // If number of hashes is odd, duplicate last hash in the list.
        if (merkle.size() % 2 != 0)
            merkle.push_back(merkle.back());
        // List size is now even.
        assert(merkle.size() % 2 == 0);

        // New hash list.
        bc::hash_list new_merkle;
        // Loop through hashes 2 at a time.
        for (auto it = merkle.begin(); it != merkle.end(); it += 2)
        {
            // Join both current hashes together (concatenate).
            bc::data_chunk concat_data(bc::hash_size * 2);
            new_merkle.push_back(bc::hash{concat_data});
        }
        merkle = new_merkle;
    }
}
  
```

A 7-2. példa a 7-1. példa lefordítását és futtatasának az eredményét mutatja.

7-2. példa: A Merkle példaprogram lefordítása és futtatása

```
$ # A merkle.cpp lefordítása
$ g++ -o merkle merkle.cpp $(pkg-config --cflags --libs libbitcoin)
$ # A merkle végrehajtható program futtatása
$ ./merkle
Aktuális Merkle hash lista:
32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d27006
30861db96905c8dc8b99398ca1cd5bd5b84ac3264a4e1b3e65afa1bcee7540c4
```

Aktuális Merkle hash lista:
d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

Result: d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

A Merkle-fák hatékonysága a méret növekedésével válik nyilvánvalóvá. A 7-3. táblázat mutatja, hogy mennyi adatcserére van szükség egy Merke útvonalon, ha bizonyítani szeretnénk, hogy az adott tranzakció része egy blokknak.

7-3. táblázat: A Merkle-fák hatékonysága

Tranzakciók száma	Blokk kb. mérete	Útvonal mérete (hash db)	Útvonal mérete (bájtokban)
16 tranzakció	4 kilobájt	4 hash	128 bájt
512 tranzakció	128 kilobájt	9 hash	288 bájt
2048 tranzakció	512 kilobájt	11 hash	352 bájt
65'535 tranzakció	16 megabájt	16 hash	512 bájt

Ahogy a táblázatban látható: még egy blokk mérete gyorsan növekszik, a 16 tranzakciót tartalmazó blokk 4 kilobájtról a 65,535 tranzakciónak megfelelő 16 megabájtra, addig a tranzakció jelenlétének az igazolásához szükséges Merkle-út sokkal lassabban, 128 bájtról csak 512 bájtra növekszik. A Merkle-fákkal megoldható, hogy egy csomópont csak a blokkfejlécet (80 bájt/blokk) töltse le, és mégis képes legyen azonosítani egy tranzakció blokkon belüli jelenlétét úgy, hogy a Merkle-útnak minden összes csomópontnak csak egy kis részét keresi vissza a teljes csomópontból. Ehhez csupán a teljes blokkláncnak, ami több gigabájt méretű, csupán egy elenyésző részét kell tárolnia vagy letöltenie. Az úgynevezett Simplified Payment Verification (Egyszerűsített Fizetési Ellenőrzés) vagy SPV csomópontok olyan csomópontok, amelyek a teljes blokklánc letöltése nélkül, Merkle-utak segítségével ellenőrzik a tranzakciók jelenlétét.

A Merkle-fák és az egyszerűsített fizetés ellenőrzés (Simplified Payment Verification (SPV))

Az úgynevezett Simplified Payment Verification csomópontok széles körben használják a Merkle-fákat. A SPV csomópontokban nincs meg az összes tranzakció, és teljes blokkokat sem töltenek le, csupán blokkfejléceket. Ahhoz, hogy a hitelesítési útvonal, vagy Merkle-út használatával ellenőrizni tudják egy tranzakció blokkon belüli jelenlétét, nem szükséges a blokkon belüli összes tranzakció letöltése.

Vegyük például egy SPV csomópontot, amely a bejövő fizetésekkel egy adott, a pénztárcájában megtalálható cím után érdeklődik. Ekkor, a SPV csomópont létrehoz egy szűrőt a peerjeiben. A szűrő a tranzakciók küldését csak a szóban forgó címre korlátozza. Amikor megjelenik egy, a szűrőnek megfelelő tranzakció, akkor a peer csomópont egy `merkleblock` üzenet segítségével elküldi az adott blokkot. A `merkleblock` üzenet egyrészt a blokkfejlécet, másrészt a keresett tranzakciót a blokk Merkle-gyökerével összekötő Merkle-utat tartalmazza. Az SPV csomópont ezt a Merkle-utat használja arra is, hogy összekösse a tranzakciót a blokkal, illetve, hogy ellenőrizze a tranzakció jelenlétét a blokkban. Az SPV csomópont a blokkfejlécet is használja, hogy a blokkot összekapcsolja a lánc további részével. Ez a tranzakció és a blokk, illetve a blokk és a blokklánc közötti két kapcsolat bizonyítja azt, hogy a tranzakció rögzítésre került a blokkláncban. Minden összevetve, az SPV csomópont kevesebb, mint egy kilobájt adat formájában megkapja a blokkfejlécet, illetve a Merkle-utat. Ez az adatmennyiség több mint ezerszer kisebb egy teljes blokknál (ami jelenleg körülbelül 1 megabájt).

8. Bányászat és konszenzus

Bevezetés

A bányászat az a folyamat, amely a pénzkészletet új bitcoinokkal bővíti. A bányászat emellett védi a bitcoin rendszert a csalásuktól vagy ugyanannak a bitcoin összegnek a többszöri elköltésétől, azaz az ún. kettős költéstől is. A bányászok feldolgozó kapacitást biztosítanak a bitcoin hálózatnak, ezért cserébe bitcoinokat kaphatnak jutalomul.

A bányászok ellenőrzik az új tranzakciókat és a globális főkönyvbe rögzítik őket. minden 10 percben egy új blokk kerül „kibányászásra”, amely az utolsó blokk kibányászása óta előfordult tranzakciókat tartalmazza, vagyis a blokkláncot ezekkel a tranzakciókkal bővíti. A blokkba foglalt, és a blokklánchoz hozzáadott tranzakciók „megerősített” tranzakciók. A megerősítés teszi lehetővé az új bitcoin tulajdonosok számára, hogy elköltsek az ilyen tranzakciókban kapott bitcoinjaikat.

A bányászok kétféle jutalmat kapnak a bányászatért: az új blokkokkal létrejövő új érméket, és a blokkban lévő tranzakciók tranzakciós díjait. Ahhoz, hogy megkaphassák ezt a jutalmat, a bányászoknak egy kriptográfiai hash algoritmust kapcsolatos bonyolult matematikai problémát kell megoldaniuk. A probléma megoldása, az ún. munkabizonyíték (Proof-of-Work) beépül az új blokkba, és bizonyítékul szolgál arra nézve, hogy a bányász jelentős számítási munkát végzett. A bitcoin biztonságát az egymással versenyző bányászok alapozzák meg, akik a munkabizonyíték előállítása révén jutalomhoz és a tranzakciók blokkláncban történő rögzítésének a jogához jutnak.

Az új érmék előállításának folyamatát azért hívjuk bányászatnak, mert a jutalom úgy lett megalkotva, hogy a nemesfémek bányászatához hasonlóan a hozadék egyre kisebb legyen. A bitcoin pénz-kibocsátása bányászattal történik, hasonlóan ahhoz, ahogy egy központi bank bankjegyek nyomtatásával új pénzt hoz létre. A bányászok által a blokkhoz hozzáadott új bitcoinok mennyisége kb. négy évenként (Pontosabban 210 ezer blokkonként) a felére csökken. 2009 januárjában 50 bitcoin/blokk értékkel indult a folyamat, ez 2012 novemberében 25 bitcoin/blokkra csökkent. A jutalom valamikor 2016-ban fog újra feleződni, és 12.5 bitcoin lesz. Ennek a képletnek az alapján a bitcoin bányászat jutalma exponenciálisan csökkeni fog, egészen 2140-ig, amikorra az összes bitcoin (20'999'999.98 BTC) kibocsátásra kerül. 2140 után több új bitcoin már nem kerül forgalomba.

A bitcoin bányászok a tranzakciós díjakat is megkapják. Mindegyik tranzakció tartalmazhat tranzakciós díjat, a tranzakció bemenetei és kimenetei közötti különbség formájában. A győztes bitcoin bányász jut hozzá a nyertes blokk tranzakcióiban lévő „apró” pénzhez. Manapság a díjak a bitcoin bányászok bevételéinek 0.5 %-át vagy még kisebb hányadát alkotják, a bevétel túlnyomó többsége az újonnan „vert” bitcoinokból származik. De ahogyan a jutalom idővel csökken majd, a blokkonkénti tranzakciók száma pedig nő, úgy fog a bitcoin bányászok bevételének egyre nagyobb hányada a díjakból származni. 2140 után a bitcoin bányászok összes bevételét a tranzakciós díjak fogják jelenteni.

A „bányászat” szó kicsit félrevezető. Azáltal, hogy a nemesfémek bányászatának a képet idézi föl, a bányászatért kapott jutalmat, a blokkonként létrejövő új bitcoinokat állítja a figyelem középpontjába. Noha a bányászokat ez a jutalom mobilizálja, a bányászat elsődleges célja nem a jutalom vagy az új érmék előállítása. Ha a bányászatot csak az érmék előállítására

szolgáló folyamatnak tekintjük, akkor nem szabad a folyamat eszközét (össztönzőjét) a folyamat céljával összekevernünk. A bányászat jelenti a decentralizált elszámolóház fő folyamatát. A bányászat során kerülnek ellenőrzésre és elszámolásra a tranzakciók. A bányászat teremti meg a bitcoin rendszer biztonságát és teszi lehetővé, hogy az egész hálózatban központi szervezet nélkül létrejöjjön a konszenzus.

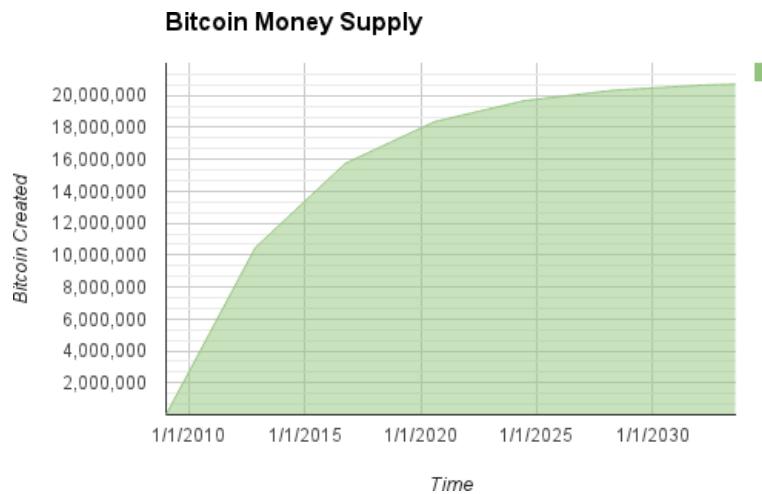
A bányászat az az újítás, ami a bitcoint különlegessé teszi: ez az a decentralizált biztonsági mechanizmus, ami a peer-to-peer digitális pénz alapja. Az újonnan kibocsátott érmékkel kapott jutalom és a tranzakciós díjak olyan össztönzést jelentenek, amely a bányászok tetteit a hálózat biztonságával hangolja össze, és egyúttal a pénzkibocsátást is megoldja.

Ebben a fejezetben először a bányászatot mint pénz kibocsátó mechanizmust fogjuk vizsgálni, majd megnézzük a bányászat legfontosabb funkcióját: a decentralizált konszenzus létrejöttének módját, amely megteremti a bitcoin biztonságát.

A bitcoin gazdaság és a pénz kibocsátás

A bitcoinokat az új blokkok létrehozása során „verik”, fix és egyre csökkenő ütemben. Az átlagosan 10 percenként létrejövő új blokkok mindegyikében teljesen új bitcoinok vannak, melyek a semmiből jönnek létre. minden 210 ezer blokk után, azaz kb. négy évente a pénz kibocsátás mértéke 50 %-kal csökken. A hálózat működésének első négy éve során mindegyik új blokk 50 új bitcoin tartalmazott.

2012 novemberében az új bitcoinok kibocsátási üteme 25 bitcoin/blokkra csökkent, és valamikor 2016-ban, a 420'000-ik blokknál fog csökkenni ismét, és 12.5 bitcoin lesz. Az új érmék kibocsátási üteme 64 „felezés” révén, exponenciálisan csökken, egészen a 13'230'000-ik blokkig (melyet valamikor 2137-ben fognak kibányászni), amikor is a jutalom eléri a legkisebb pénzegységet, az 1 Satoshiit. Végül a 13.44 milliomodik blokk után, kb. 2140-re a kibocsátott bitcoinok mennyisége 2'099'999'997'690'000 Satoshi, azaz kb. 21 millió bitcoin lesz. Ezt követően a blokkok már nem fognak új bitcoinokat tartalmazni, és a bányászok jutalma kizárolag a tranzakciós díjakból fog származni. A 8-1. ábra mutatja a forgalomban lévő bitcoinokat az idő függvényében. Látható a kibocsátás csökkenő üteme.



8-1. ábra: A bitcoin mennyiségét az idő függvényében egy mértanilag csökkenő pénzkibocsátási ütem jellemzi



A kibányászott érmék száma adja a bitcoinok számának *felső határát*. A gyakorlatban a bányászoknak nem kötelező a blokk után járó teljes jutalom kibányászása. Ilyen blokkok már eddig is előfordultak, és a jövőben is lehetnek ilyenek, emiatt az össze bitcoin száma kevesebb lesz, mint az elvi határ.

A 8-1. példaprogramban kiszámítjuk a forgalomba kerülő összes bitcoin számát.

8-1. példa: Egy script, amely azt számítja ki, hogy összesen hány bitcoin fog forgalomba kerülni

```
$ cat max_money.py
# Original block reward for miners was 50 BTC
start_block_reward = 50
# 210000 is around every 4 years with a 10 minute block interval
reward_interval = 210000

def max_money():
    # 50 BTC = 50 0000 0000 Satoshis
    current_reward = 50 * 10**8
    total = 0
    while current_reward > 0:
        total += reward_interval * current_reward
        current_reward /= 2
    return total

print "Total BTC to ever be created:", max_money(), "Satoshis"
```

A 8-2. példa mutatja a script futtatása során kapott kimenetet.

8-2. példa: A max_money.py script futtatása

```
$ python max_money.py
Az összes, valaha létrejövő BTC: 209999997690000 Satoshi
```

A véges és egyre csökkenő mértékű kibocsájtás egy véges pénzkészletet hoz létre, amely ellenálló az inflációval szemben. A hagyományos papírpénzekkel szemben, melyek korlátlanul nyomtathatók, a bitcoin nyomtatással soha sem lesz inflálható.

Deflációs pénz

A fix és egyre csökkenő pénzkibocsájtás egyik legfontosabb és legvitatottabb következménye az, hogy a pénz óhatatlanul *deflációs* tendenciát rejt magában. A defláció az a jelenség, melynek során a pénz értéke nő, mivel a kereslet és a kínálat közötti egyensúly hiánya felhajtja a pénz értékét (és váltási árfolyamát). Az árak deflációja, vagyis az infláció ellentéte azt jelenti, hogy a pénznak idővel egyre nagyobb lesz a vásárlóértéke.

Sok közigazdász szerint egy deflációs gazdaság katasztrófát jelent, és mindenáron el kell kerülni. Ennek az az oka, hogy a gyors defláció időszakában az emberek inkább felhalmozzák a pénzt, ahelyett hogy elköltenék, mert azt remélik, hogy az árak csökkenni fognak. Ilyen jelenség bontakozott ki Japán „Elveszett Évtizede” során, mikor a kereslet teljes összeomlása egy deflációs spirálba taszította a japán pénzt.

A bitcoin szakértők szerint a defláció önmagában nem rossz. Szerintük a deflációt eddig

azért hozták kapcsolatba a kereslet összeomlásával, mert az általunk ismert deflációra ez az egyetlen példa. A papírpénz korlátlan nyomtathatósága mellett nagyon nehéz deflációs spirálba kerülni, kivéve, ha teljesen összeomlik a kereslet és nincs pénznyomtatási hajlandóság. A bitcoin deflációját nem a kereslet összeomlása okozza, hanem az előre megjósolható kibocsátás.

A gyakorlatban bebizonyosodott, hogy a deflációs pénz által okozott felhalmozási ösztön árleszállításokkal legyőzhető, ha ennek mértéke nagyobb, mint a vevő felhalmozási ösztöne. Mivel az eladó szintén érdekelt a pénz felhalmozásban, a leszállított ár jelenti azt az egyensúlyi árat, amely mellett a két felhalmozási ösztön kiegyenlíti egymást. A legtöbb eladó 30%-os árleszállítással különösebb nehézségek nélkül le tudja győzni a felhalmozási ösztönt, és bevételt tud generálni. A jövő zenéje, hogy a pénz deflációs jellege valóban problémát jelent-e, ha a deflációt nem a gyors gazdaságcsökkenés váltja ki.

Decentralizált konszenzus

Az előző fejezetben a blokkláncot, vagyis az összes tranzakciót tartalmazó publikus globális főkönyvet vizsgáltuk, amelyet a bitcoin rendszer valamennyi résztvevője a tulajdon hiteles okmányaként fogad el.

De hogyan lehet a hálózaton belül kölcsönös bizalom nélkül egyetlen univerzális „igazságban” megállapodni arról, hogy kinek mije van? Az összes hagyományos fizetési rendszer egy bizalmi modellen alapul, melyben egy központi szervezet nyújtja az elszámolási szolgáltatást, melynek során alapjában véve a tranzakciók ellenőrzése és elszámolása történik. A bitcoinban nincs ilyen központi szervezet, de mégis minden csomópontnak egy teljes másolata van a főkönyvről, melyben hiteles okmányként megbízhat. A blokkláncot nem egy központi szervezet hozza létre, hanem egymástól függetlenül a hálózat csomópontjai állítják össze. Valamiféleképpen a hálózat csomópontjai képesek a nem biztonságos hálózati összeköttetéseken továbbított információk alapján ugyanarra a következetesre jutni és ugyanazt a főkönyvi példányt összeállítani, mint a többiek. Ebben a fejezetben azt a folyamatot vizsgájuk meg, amellyel a bitcoin hálózat központi szervezet nélkül globális konszenzust ér el.

Satoshi Nakamoto fő felfedezése a *konszenzus kialakulásának* decentralizált mechanizmusa volt. Ez a konszenzus nem explicit módon jön létre – nincsenek választások vagy rögzített időpillanatok, amikor konszenzus van. A közmegegyezés inkább a sok ezer, egyszerű szabályokat követő, független csomópontok aszinkron kölcsönhatásai révén alakul ki. A bitcoin esetén a pénz, a tranzakciók, a pénz küldés vagy a biztonság nem valamilyen központi szervezettől függ és nem a bizalomra épül, hanem ebből a felfedezésből adódik.

A bitcoin decentralizált konszenzusa négy folyamat kölcsönhatásának az eredményeképpen jön létre. A négy folyamat egymástól függetlenül megy végbe a hálózat csomópontjain:

- minden egyes tranzakció egymástól független ellenőrzése. Ezt számos kritérium alapján a teljes csomópontok végzik.
- ezeknek a tranzakciónak az új blokkokban történő egyesítése. Ezt a bányász csomópontok egymástól függetlenül végzik. A bányász csomópontok a munkabizonyíték algoritmus segítségével igazolják, hogy elvégeztek bizonyos számításokat.

- az új blokk ellenőrzése és lánca szervezése, melyet az egyes csomópontok egymástól függetlenül végeznek.
- a legtöbb összesített munkabizonyíték számítást tartalmazó lánca kiválasztása. Ezt minden egyes csomópont a többiről függetlenül végzi.

A következő néhány részben megvizsgáljuk ezeket a folyamatokat, valamint megnézzük, hogy a kölcsönhatásai hogyan teremtik meg a hálózat egészében megjelenő közmegegyezést, amellyel bármely bitcoin csomópont képes a saját hiteles, megbízható, publikus és globális főkönyvének az összeállítására.

A tranzakciók egymástól független ellenőrzése

Az [5. fejezetben](#) láttuk, hogy egy pénztárca az UTXO-k összegyűjtésével állítja elő a tranzakciókat. Ennek során először megadja a megfelelő zárolást feloldó scripteket, majd új kimeneteket hoz létre, melyek egy új tulajdonoshoz vannak hozzárendelve. Az így előálló tranzakciót ezután elküldi a bitcoin hálózat szomszédos csomópontjainak, hogy a tranzakció az egész bitcoin hálózatban szétterjedhessen.

Mielőtt azonban egy bitcoin csomópont továbbítaná a tranzakciókat a szomszédainak, először ellenőrzi őket. Ez biztosítja, hogy csak érvényes tranzakciók terjedjenek tova a hálózatban, az érvénytelen tranzakciókat pedig már az első csomópont elvesse.

A csomópontok az egyes tranzakciót kritériumok hosszú sora alapján ellenőrizik:

- helyes-e a tranzakció szintaxisa és adatstruktúrája
- sem a bemenetek, sem a kimenetek listája nem lehet üres
- a tranzakció mérete bájtokban kisebb-e a `MAX_BLOCK_SIZE`-nál
- az egyes kimenetek értéke, valamint ezek összege a megengedett tartományon belül van-e (kevesebb-e, mint 21 M érme, és több-e, mint 0)
- semelyik bemenet hashe sem lehet 0, N=-1 (a coinbase tranzakciót nem kell továbbküldeni)
- az `nLockTime` kisebb vagy egyenlő-e `INT_MAX`-nál
- a tranzakció mérete bájtokban nagyobb vagy egyenlő-e 100-nál
- a tranzakcióban lévő aláírási műveletek száma kevesebb-e, mint az aláírások max. számára vonatkozó határ
- a zárolást feloldó script (`scriptSig`) csak számokat helyezhet a veremre, a zároló scriptnek (`scriptPubkey`) pedig meg kell felelnie az `isStandard` formátumoknak (itt történik a „nem szabványos” tranzakciók elvetése)
- a memória poolban vagy a fő ág egy blokkjában kell legyen egy ennek megfelelő tranzakció
- minden egyes bemenetre: ha a hivatkozott kimenetek léteznek a memória poolban lévő bármelyik másik tranzakcióban, akkor a tranzakció elvetése
- minden egyes bemenetre: annak az ellenőrzése, hogy létezik-e a fő ágban vagy a tranzakció poolban a hivatkozott kimeneti tranzakció. Ha a kimeneti tranzakció bármelyik bemenetnél hiányzik, akkor egy árva tranzakcióról van szó. A tranzakciót az árva tranzakciók listájához adja hozzá, ha a poolban még nincs meg az illeszkedő tranzakció
- minden egyes bemenetre: ha a hivatkozott kimeneti tranzakció egy coinbase kimenet, akkor van-e már legalább `COINBASE_MATURITY` (100) megerősítése
- minden egyes bemenetre: a hivatkozott kimenet létezik-e és elkötötlen-e

- a hivatkozott kimeneti tranzakciók alapján a bemeneti értékük megállapítása, és annak az ellenőrzése, hogy mindegyik bemeneti érték, valamint az összegük is a megengedett értéktartományban van-e (21 M érménél kevesebb, 0-nál több)
- a tranzakció elvetése, ha a bemeneti értéke összege < a kimeneti értékek összege
- a tranzakció elvetése, ha a tranzakciós díj túl kicsi ahhoz, hogy a tranzakció bekerülhessen egy üres blokkba
- minden egyes bemenetre: a zárolást feloldó script és a neki megfelelő kimeneti zároló script megfelel-e egymásnak

Ezek a feltételek a bitcoin referencia kliens következő függvényeiben vannak részletesen leírva: `AcceptToMemoryPool`, `CheckTransaction`, és `CheckInputs`. A feltételek idővel változhatnak, pl. ha újfajta Dos szolgáltatás megtagadási támadások kezelésére van szükség, vagy arra, hogy a szabályok lazítása révén többféle tranzakció típus legyen kezelhető.

Mivel a többi csomóponttól függetlenül mindegyik csomópont ellenőri a tranzakciókat azok beérkezésekor, és a továbbításuk előtt, mindegyik csomópont felépíti az érvényes (de még nem megerősített) új tranzakciók készletét, az ún. *tranzakció pool*, vagy másnéven *memória pool* vagy *mempool*.

Bányász csomópontok

A bitcoin hálózat bizonyos csomópontjai speciális csomópontok, az ún. *bányászok*. Az [1. fejezetben](#) bemutattuk Jinget, a bitcoin bányászt, aki Sanghajban számítástechnikát tanul. Jing úgy jut bitcoinokhoz, hogy egy „bányász platformot” üzemeltet, amely egy bitcoin bányászatra szolgáló speciális számítógép hardver. Jing speciális bányász hardvere összeköttetésben áll egy teljes bitcoin csomópontot futtató szerverrel. Jingtől eltérően nemelyik bányász teljes csomópont nélkül bányászik, amint azt a „[Bánytársaságok \(Mining Pools\)](#)” részben látni fogjuk. Jing csomópontja a hálózat többi csomópontjához hasonlóan megkapja és továbbítja a hálózat megerősítetten tranzakcióit. Jing csomópontja azonban új blokkokba is egyesíti ezeket a tranzakciókat.

Jing csomópontja a többi csomóponthoz hasonlóan szintén észleli a bitcoin hálózaton belül továbbított új blokkokat. De egy új blokk érkezése a bányász csomópont számára speciális jelentőségű. A bányászok közötti versengést lényegében az új blokk szétterjedése állítja le, mert ez felel meg a győztes kihirdetésének. Egy bányász számára egy új blokk érkezése azt jelenti, hogy valaki más nyerte meg a versenyt, ő pedig veszített. De az egyik versenyforduló vége egyúttal a következő forduló kezdete. Az új blokk nem csak egy kockás zászló, amely a verseny végét jelzi, hanem egy startpisztoly is, mely a következő blokkért folyó versenyt indítja.

A tranzakciók blokkokba gyűjtése

A bitcoin csomópontokban a tranzakciók az ellenőrzés után bekerülnek a *memória poolokba*, másnéven *tranzakció készletekbe*. A tranzakciók itt várakoznak arra, hogy bekerüljenek egy blokkba (kibányásszák őket). Jing csomópontja a többi csomóponthoz hasonlóan összegyűjt, ellenőri, és továbbítja az új tranzakciókat. De a többi csomópontról eltérően Jing csomópontja ezekből a tranzakciókból egy *blokk jelöltet* is létrehoz.

Kövessük a annak a blokknak az útját, amely akkor keletkezett, amikor Alíz egy csésze kávét vett Bob kávézójában (lásd a „[Vegyünk egy csésze kávét](#)” rész). Alíz tranzakciója a 277'316-

ik blokkba lett befoglalva. A fejezetben szereplő fogalmak szemléltetése érdekében tegyük fel, hogy a blokkot Jing bányagépe bányászta ki, és kövessük Alíz tranzakcióját, amint részévé válik ennek az új blokknak.

Jing bányász csomópontja egy helyi példányt tart fönn a blokkláncból, vagyis azokból a blokkokból, melyek bitcoin rendszer kezdete, 2009 óta képződtek. Mikor Alíz megvette a csésze kávét, Jing csomópontja már egészen a 277'314-ik blokkig összeállította a láncot. Jing csomópontja figyelte a tranzakciókat, megpróbált előállítani egy új blokkot, és egyúttal figyelte a többi csomópont által előállított blokkokat is. Miközben Jing csomópontja bányászott, megjött a 277'315-ik blokk a bitcoin hálózattól. Az új blokk érkezése jelezte a 277'315-ik blokkért történő versengés végét, és a 277'316-ik előállításáért folytatott verseny kezdetét.

Az előző 10 percben, míg Jing csomópontja a 277'315-ik blokk megoldását kereste, az új blokk előállításának előkészületeként tranzakciókat is gyűjtött. Mostanra már pár száz tranzakció gyűlt össze a memóriában. Amikor Jing csomópontja megkapta a 277'315-ik blokkot és ellenőrizte azt, a memóriában lévő tranzakciókat is ellenőrizte, és eltávolította közülük azokat, melyek szerepeltek a 277'315- blokkban. A memóriában maradt tranzakciók megerősítetlenek, és arra várnak, hogy egy új blokkba foglalják őket.

Jing csomópontja egy új üres blokkot állít elő, a 277'316-ik blokkot. Ez a blokk csak egy létrehozandó jelölt, amely még nem érvényes, mivel nem tartalmaz érvényes munkabizonyítékot. A blokk csak akkor válik érvényessé, ha a bányásznak sikerül egy megoldást találnia a munkabizonyíték algoritmusra .

Tranzakció életkor, díjak és prioritás

Jing bitcoin csomópontja kiszámítja az összes, memóriában lévő tranzakció prioritását, és a létrehozandó blokk jelölthöz először a legmagasabb prioritású tranzakciót adja hozzá. A tranzakciók prioritása a bemeneteikben szereplő UTXO-k „életkorától” függ. Ez lehetővé teszi, hogy a régi és nagy értékű bemenetek elsőbbséget élvezzenek az újabb és kisebb bemenetekkel szemben. Az elsőbbséget élvező/priorizált tranzakciók díj nélkül küldhetők, ha elég hely van a blokkban.

A tranzakció prioritása úgy számítható ki, hogy összeadjuk az egyes bemenetek értékének és életkorának szorzatait, és az összeget elosztjuk a tranzakció teljes méretével:

$$\text{Prioritás} = \frac{\text{Összeg} \cdot (\text{Bemenet}_\text{értéke} * \text{Bemenet}_\text{életkora})}{\text{Tranzakció}_\text{méret}}$$

A fenti egyenletben a bemenet értéke alap egységekben, Satoshiban (1/100M bitcoinban) van megadva. Az UTXO életkora az UTXO-nak a blokkláncban történő rögzítése óta képződött blokkok számával egyenlő. Az életkor azt méri, hogy milyen „mélyen” van a blokk a blokkláncban. A tranzakció mérete bájtokban van megadva.

Egy tranzakció akkor számít „magas prioritásúnak”, ha a prioritása nagyobb, mint 57'600'000, ami annak felel meg meg, mint ha 1 bitcoin (100M Satoshi), melynek életkora 1 nap (144 blokk) egy 250 bájt méretű tranzakcióból szerepelne.

$$\text{Magas}_\text{prioritás} > 100'000'000 \text{ Satoshi} * 144 \text{ blokk} / 250 \text{ bájt} = 57'600'000$$

A blokk első 50 kilobájtja a magas prioritású tranzakciók számára van fenntartva. Jing csomópontja kitölti az első 50 kilobájtot, és ebben a díjtól függetlenül a legmagasabb prioritású tranzakciókat helyezi el először. Ennek megfelelően a magas prioritású tranzakciók akkor is feldolgozásra kerülnek, ha a bennük szereplő tranzakciós díj nulla.

Ezután Jing csomópontja a blokk maradék részét azokkal a tranzakciókkal tölti fel, amelyek legalább a minimális tranzakciós díjat tartalmazzák, előnyben részesítve azokat, melyeknél egy kilobájtra vonatkoztatva a legmagasabb a tranzakciós díj. A feltöltés egészen a max. blokkméretig (a kódban `MAX_BLOCK_SIZE`) tart.

Ha marad még hely a blokkban, akkor Jing bányász csomópontja a maradék helyet tranzakciós díj nélküli tranzakciókkal tölti fel. Némelyik bányász a tranzakciós díj nélküli tranzakciókat „ahogy esik, úgy puffan” módon kezeli. Más bányászok viszont akár teljesen ki is hagyhatják a létrehozandó blokk jelöltből ezeket a tranzakciókat.

A blokk felöltése után a memóriában maradó tranzakciók a következő blokkba kerülhetnek be. A memóriában maradó tranzakció bemenetei „öregebbekké” válnak, mivel az általuk elköltött UTXO az új blokk miatt mélyebbre került a blokkláncban. Mivel a tranzakció prioritása függ a bemeneteinek az életkorától, a memóriában maradó tranzakciók egyre régebbiek lesznek, és emiatt nő a prioritásuk. Végül a tranzakciós díjat nem tartalmazó tranzakciók is elég magas prioritásúvá válhatnak ahhoz, hogy díjtalansára befoglalásra kerülhessenek egy blokkba.

A bitcoin tranzakcióknak nincs lejáratú idejük. Egy jelenleg érvényes tranzakció az idők végezetéig érvényes marad. Mivel azonban a tranzakciót a hálózat csak egyszer továbbítja, csak addig marad fenn, amíg benne van egy bányász csomópont memóriájában. Ha a bányász csomópontot újraindítják, a memóriájában lévő tartalom törlődik, mivel a memória csak egy átmeneti, nem tartós tárolási forma. Ha a hálózat egy érvényes tranzakciót küldött szét, de a tranzakció nem hajtódik végre, akkor ennek az lehet az oka, hogy már nincs meg egyetlen bányász memóriájában sem. A pénztárca szoftverek ilyen esetben vagy változatlanul újraküldik a tranzakciót, vagy magasabb tranzakciós díjjal ismét előállítják és elküldik őket, amíg végül ésszerű idő alatt meg nem történik a végrehajtásuk.

Mikor Jing csomópontja összeszedte a memóriában lévő tranzakciókat, a jövendő blokk 418 tranzakciót tartalmazott, és a tranzakciós díj összesen 0.09094928 bitcoin volt. A blokkláncban a 277'316-ik blokk a Bitcoin Core kliens parancsos felületével a következőképpen nézhető meg:

```
$ bitcoin-cli getblockhash 277316
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4

$ bitcoin-cli getblock
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

8-3. példa: A 277'316-ik blokk

A generáló tranzakció

A blokk első tranzakciója egy különleges tranzakció, melyet *generáló tranzakciónak* vagy *coinbase tranzakciónak* hívnak. Ezt a tranzakciót Jing csomópontja hozza létre, és a bányászatért járó jutalmat tartalmazza. Jing csomópontja a generáló tranzakciót a saját pénztárcájába történő kifizetésként hozza létre: „Kifizetés Jing címére 25.09094928 bitcoin értékben”. A blokk kibányászáért kapott összes jutalmat a coinbase jutalom (25 új bitcoin) és a blokkba befoglalt tranzakciók tranzakciós díjainak (0.09094928) összege adja.

```
$ bitcoin-cli getrawtransaction  
d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f 1
```

8-4. példa: Generáló tranzakció

```

        ],
        "vout" : [
            {
                "value" : 25.09094928,
                "n" : 0,
                "scriptPubKey" : {
                    "asm" :
"02aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21OP_CHECK
SIG",
                    "hex" :
"2102aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac",
                    "reqSigs" : 1,
                    "type" : "pubkey",
                    "addresses" : [
                        "1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N"
                    ]
                }
            }
        ],
        "blockhash" :
"0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
        "confirmations" : 35566,
        "time" : 1388185914,
        "blocktime" : 1388185914
    }
}

```

A szabályos tranzakciókkal ellentétben a generáló tranzakciók a bemenetükön nem fogyasztanak (nem költenek el) UTXO-kat. Csak egy bemenetük van, a *coinbase*, amely a semmiből állít elő új bitcoinokat. A generáló tranzakciónak egy kimenete van, melyben a bányász saját bitcoin címére történő kifizetés áll. A generáló tranzakció kimenete 25.09094928 bitcoin küld a bányász bitcoin címére, ebben az esetben az 1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N címre.

A coinbase jutalom és tranzakciós díjak

A generáló tranzakció előállításához Jing csomópontjának először ki kell számítania a teljes tranzakciós díjat. Ehhez a blokkban szereplő 418 tranzakció összes bemenetének összegéből le kell vonnia a 418 tranzakció kimeneteinek összegét:

$$\text{Teljes_díj} = \text{Összeg(Bemenetek)} - \text{Összeg(Kimenetek)}$$

A 277'316-ik blokkban a tranzakciós díj 0.09094928 bitcoin volt.

Ezután Jing csomópontja kiszámítja a blokkért járó jutalmat. A jutalom a blokk magasságától függ, kezdetben 50 bitcoin volt, és minden 210'000 blokk után feleződik. Mivel ennek a blokknak a magassága 277'316, 25 bitcoin a jutalom.

A számítás a Bitcoin Core kliens `GetBlockValue` függvényében látható, amint azt a 8-5. példa mutatja:

8-5. példa: A blokk jutalom kiszámítása – GetBlockValue függvény, Bitcoin Core kliens, main.cpp, 1305. sor

```
int64_t GetBlockValue(int nHeight, int64_t nFees)
{
    int64_t nSubsidy = 50 * COIN;
    int halvings = nHeight / Params().SubsidyHalvingInterval();

    // Force block reward to zero when right shift is undefined.
    if (halvings >= 64)
        return nFees;

    // Subsidy is cut in half every 210,000 blocks which will occur
    // approximately every 4 years.
    nSubsidy >>= halvings;

    return nSubsidy + nFees;
}
```

A kezdeti támogatás Satoshiban az 50 és a COIN konstans (100,000,000 Satoshi) szorzata. A kezdeti jutalmat (nSubsidy) ez állítja be 5 milliárd Satoshira.

Ezt követően a függvény kiszámítja, hogy hány darab feleződés történt, mégpedig úgy, hogy az aktuális blokk magasságát elosztja a felezési intervallummal (SubsidyHalvingInterval). A 277'316-ik blokk esetében az eredmény 1.

A feleződések értéke max. 64 lehet, emiatt 64-nél több feleződés esetén a kód nulla jutalmat ad (csak a tranzakciós díjat adja vissza).

Ezután a függvény minden egyes feleződésnek megfelelően jobbra léptetéssel 2-vel osztja a jutalmat (nSubsidy). A 277'316-ik blokk esetében ez azt jelenti, hogy az 5 milliárd Satoshiis jutalmat egyszer lépteti jobbra (egy felezés van), az eredmény pedig 2.5 milliárd Satoshi, vagyis 25 bitcoin. A 2-vel való osztás azért a bináris jobbra léptetéssel történik, mert ez hatékonyabb, mint egy egésszel vagy valós számmal történő osztás.

Végül a coinbase jutalmat (nSubsidy) összeadja a tranzakciós díjjal (nFees), és ezt az összeget adja vissza eredményként.

A generáló tranzakció felépítése

Ezen számítások után Jing csomópontja előállítja a generáló tranzakciót, amellyel kifizet magának 25.09094928 bitcoint.

Amint azt a 8-4. példa mutatja, ennek a tranzakciónak különleges formátuma van. A tranzakció bemenetén nem az elkölthető, előző UTXO-k vannak megadva, hanem egy „coinbase” bemenetet. A tranzakciós bemeneteket az [5-3. táblázatban](#) vizsgáltuk. Hasonlítsunk össze egy szokásos tranzakció bemenetét a generáló tranzakció bemenetével. Egy szokásos tranzakció szerkezetét a 8-1. táblázat mutatja, míg a 8-2. táblázat a generáló tranzakció bemeneteinek a szerkezetét mutatja.

8-1. táblázat: Egy „közönséges” tranzakció egyik bemenetének szerkezete

Méret	Mező	Leírás
32 bájt	Tranzakció hash	Mutató arra a tranzakcióról, amely az elköltendő UTXO-t tartalmazza
4 bájt	Output Index	Az elköltendő UTXO indexe, az első 0
1-9 bájt (VarInt)	A zárolást megszüntető script mérete	A zárolást megszüntető script mérete bájtokban
Változó	A zárolást megszüntető script	Az UTXO-t zároló script feltételeit kielégítő script
4 bájt	Sorszám	Tx-helyettesítő lehetőség, Jelenleg letiltva, 0xFFFFFFFF

8-2. táblázat: A generáló tranzakció bemenetének a szerkezete

Méret	Mező	Leírás
32 bájt	Tranzakció hash	Az összes bit nulla: nem hivatkozik tranzakció hashre
4 bájt	Output Index	Az összes bit egy: 0xFFFFFFFF
1-9 bájt (VarInt)	Coinbase adat méret	Coinbase adathossz, 2 és 100 bájt között
Változó	Coinbase adat	Tetszőleges adat, a v2 blokkokban az extra nonce-t és a bányász címkéket tartalmazza, a v2 blokkokban a blokk magassággal kell kezdődnie
4 bájt	Sorszám	0xFFFFFFFF

A generáló tranzakciójában az első két mező olyan értékeket tartalmaz, amely nem UTXO hivatkozásnak felel meg. A „Tranzakció hash” helyett az első mező mind a 32 bájtja nullával van feltöltve. Az „Output index” 4 bájtja 0xFF (255). Az „Zárolást megszüntető script” helyén a coinbase adat található, amely a bányászok által használt adatmező.

Coinbase adatok

A generáló tranzakcióknak nincs zárolást feloldó script (ún. `scriptSig`) mezőjük, hanem a mező coinbase adatokat tartalmaz, melyek hossza 2 és 100 bájt között van. Az első néhány bájt kivételével a coinbase adatok a bányász által tetszőlegesen használhatók.

Például a genezis blokkban Satoshi Nakamoto ezt a szöveget helyezte el a coinbase adatmezőbe: „The Times 03/Jan/2009 Chancellor on brink of second bailout for banks” („The Times, 2009. jan. 3., A pénzügyminiszter hajlik a bankok második kimentésére”). Ily módon a mezőt a dátum bizonyítására és egyúttal egy üzenet továbbítására használta. Jelenleg a bányászok a coinbase adatként adják meg az extra nonce értékét, amint azt a következőkben látni fogjuk.

A coinbase első néhány bájta korábban tetszőleges lehetett, de ez most már nem így van. A BIP0034 (Bitcoin Improvement Proposal 34) szerint a 2. verziójú blokkoknál (amelyeknél a verzió mező 2-re van állítva) a coinbase mező elején a blokk magasságot kell megadni egy „push” script utasítással.

A 277'316. blokkban azt látjuk, hogy a coinbase mezőben (amely a tranzakciós bemenet „Zárolást megszüntető scriptje” vagy `scriptSig`-je) a `03443b0403858402062f503253482f` hexadecimális szám áll (lásd 8-4. példa). Dekódoljuk ezt az értéket!

Az első bájt, a `03` arra utasítja a script végrehajtó mechanizmust, hogy a következő 3 bájtot helyezze a script vermére (lásd az [A. függeléket](#)). A következő 3 bájt, a `0x443b04` a blokk magasság, ahol a legkisebb helyiértékű bájt áll legelöl. A bájtok sorrendjének megcserélése után `0x043b44` lesz az eredmény, ami decimálisan 277'316.

A következő néhány hexadecimális számjegy (`03858402062`) az *extra nonce*-t kódolja (lásd „[Az extra nonce megoldás](#)” részt), amely a megfelelő munkabizonyíték előállításához használt véletlen érték.

A coinbase adat utolsó része (`2f503253482f`) a `/P2SH/` string ASCII kódja, ami azt jelzi, hogy a blokkot kibányászó csomópont támogatja a BIP0016-ban definiált Fizetés-a-script-hashnek (P2SH) bővítést. A P2SH opció bevezetése megkövetelte, hogy a bányászok „szavazzanak”, hogy a BIP0016-ot vagy a BIP0017-et támogatják. Azok, akik a BIP0016-ot támogatták, a `/P2SH/`-t tették a coinbase adatmezőbe. Azok, akik a P2SH BIP0017 szerinti megvalósítását támogatták, a `p2sh/CHV`-t tették a coinbase adatmezőbe. A BIP0016 lett győztes, de sok bányász továbbra is beteszí a `/P2SH/` stringet a coinbase mezőbe, így jelezve, hogy támogatja ezt az opciót.

A 8-6. példa az „[Alternatív kliensek, könyvtárak és eszközök](#)” részben bevezetett libbitcoin könyvtárra támaszkodva veszi ki a genezis blokkból a coinbase adatokat és jeleníti meg Satoshi üzenetét. Megjegyezzük, hogy a libbitcoin könyvtár tartalmazza a genezis blokk egy statikus másolatát, ezért a példa program közvetlenül a könyvtárból tudja elővenni a genezis blokkot.

8-6. példa: A genezis blokkban lévő coinbase adatok megjelenítése

```
$ cat satoshi-words.cpp
/*
   Display the genesis block message by Satoshi.
*/
#include <iostream>
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Create genesis block.
    bc::block_type block = bc::genesis_block();
    // Genesis block contains a single coinbase transaction.
    assert(block.transactions.size() == 1);
    // Get first transaction in block (coinbase).
    const bc::transaction_type& coinbase_tx = block.transactions[0];
    // Coinbase tx has a single input.
    assert(coinbase_tx.inputs.size() == 1);
```

```

    const bc::transaction_input_type& coinbase_input =
coinbase_tx.inputs[0];
    // Convert the input script to its raw format.
    const bc::data_chunk& raw_message = save_script(coinbase_input.script);
    // Convert this to an std::string.
    std::string message;
    message.resize(raw_message.size());
    std::copy(raw_message.begin(), raw_message.end(), message.begin());
    // Display the genesis block message.
    std::cout << message << std::endl;
    return 0;
}

```

A kódot a GNU C++ fordítóprogrammal fordítottuk le. Futtatása a 8-7. példa szerint történt.

8-7. példa: A „Satoshi szavai” példaprogram fordítása és futtatása

```

$ # A kód lefordítása
$ g++ -o satoshi-words satoshi-words.cpp $(pkg-config --cflags --libs
libbitcoin)
$ # A végrehajtható program futtatása
$ ./satoshi-words
^D?♦?<GS>^A^DEThe Times 03/Jan/2009 Chancellor on brink of second bailout
for banks (magyarul: A pénzügyminiszter hajlik a bankok második
kimentésére.)

```

A blokkfej előállítása

A blokkfej előállításához a bányász csomópontnak a 8-3. táblázatban látható következő hat mezőt kell kitöltenie:

8-3. táblázat: A blokkfej szerkezete

Méret	Mező	Leírás
4 bájt	Version	A szoftver/protokoll változásokat nyomon követő verziószám
32 bájt	Previous Block Hash	Hivatkozás a blokklánc előző (szülő) blokkjának a hashére
32 bájt	Merkle Root	A blokk tranzakcióihoz tartozó Merkle-fa gyökerének a hashe
4 bájt	Timestamp	Időbényeg: hozzávetőleg mikor jött létre a blokk (a Unix kezdőidő óta eltelt másodpercek)
4 bájt	Difficulty Target	A munkabizonyíték algoritmus által megkövetelt cél nehézségi szint
4 bájt	Nonce	A munkabizonyíték algoritmus által használt számláló

A 277'316-ik blokk kibányászásakor a blokk szerkezetére jellemző verziószám „2” volt, amelyet a legkisebb helyértékű bájt első helyre írásával, 4 bájton a `0x02000000` ábrázol.

Ezután a bányász csomópontnak az „Előző blokk hashét” kell betennie a fejbe. Ez a hálózattól előzőleg kapott 277'315-ik blokk blokkfejének a hashe, melyet Jing csomópontja az ellenőrzés után a 277'316-ikként létrehozandó blokk szülőjének választott. A 277'315-ik blokk blokkfejének a hashe:

`0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569`

A következő lépés a tranzakció összesítése egy Merkle-fa formájában, mely ahhoz szükséges, hogy a Merkle-fa gyökere bekerülhessen a blokkfejbe. A generáló tranzakció a blokk első tranzakciójá. Ezt még 418 további tranzakció követi, vagyis összesen 419 tranzakció van a blokkban. Amint azt a „[Merkle-fák](#)” részben láttuk, a fában páros számú „levél” csomópontnak kell lennie, ezért az utolsó tranzakciót meg kellett duplázni, hogy a blokkban 420 csomópont legyen. Mindegyik csomópont egy tranzakció hashét tartalmazza. A tranzakció hasheket a bányász csomópont ezután párokba rendezi, és létrehozza a fa minden egyes szintjét, míg végül az összes tranzakcióból eljut a fa „gyökerét” alkotó csomóponthoz. A Merkle-fa gyökere az összes tranzakciót egyetlen egy 32 bájtos értékbe sűrít, amelyet a 8-3. példa listájában „merkle root”-ként van kilistázva:

`c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e`

A bányász csomópont ezután egy 4 bájtos időbényeget tesz a blokkfejbe. Az időbényeg Unix „Epoch” időbényegként van kódolva, ami az 1970. január 1. éjfél (UTC/GMT) óta eltelt másodpercek száma. A `1388185914` érték 2013. december 27. 23:11:54 UTC/GMT időnek felel meg.

A csomópont ezután kitölti a megkívánt nehézségi szintet – ettől függ, hogy milyen bonyolult munkabizonyítékra van szükség ahhoz, hogy érvényes legyen a blokk. A bonyolultságot a blokk a „nehézség mértéke” bitekben tárolja, amely a cél nehézségi szintet kódolja mantissza és kitevő formájában. A kitevő 1 bájtos, ezt egy 3 bájtos mantissza követi. A 277'316-ik blokk esetében a nehézségi szinthez tartozó bitek értéke 0x1903a30c. Az első rész, 0x19, egy hexadecimális kitevő, míg a következő rész, a 0x03a30c a mantissza. A cél nehézségi szint magyarázata „[A cél nehézségi szint és a nehézségi szint újraszámítása](#)” részben, míg a „nehézségi szintet megadó bitek” magyarázata „[A nehézségi szint ábrázolása](#)” részben szerepel.

Az utolsó mező a nonce, amelybe 0 kezdőérték kerül.

A mezők kitöltésével a blokkfej teljessé vált, és kezdődhet a bányászat folyamata. A cél az, hogy egy olyan nonce értéket találunk, amelynél a blokkfej hashe kisebb, mint a cél nehézségi szint. A bányász csomópont nonce értékek billióit és trilliót ellenőrzi, hogy egy olyan nonce értéket találjon, amely megfelel ennek a feltételnek.

A blokk kibányászása

Most, hogy Jing csomópontja megkonstruálta az előállítandó blokk jelöltet, Jing hardver „platformján” a sor, hogy „kibányássza” a blokkot, vagyis olyan megoldást találjon a munkabizonyíték algoritmusra, amely a blokkot érvényesé teszi. Könyvünkben sokat tanulmányoztuk, hogy a bitcoin rendszer hogyan használja a kriptográfiai hash függvényeket.. A bitcoin bányászat folyamata az SHA256 függvényt használja.

A bányászat egyszerűen az a folyamat, melynek során a blokkfej hashe egy paraméter megváltoztatása után ismételten kiszámításra kerül, mindaddig, amíg a hash meg nem felel egy adott cél értéknek. A hash függvény eredményét előre nem ismert, és olyan bemenet sem adható meg, amely egy adott hash értéket hoz létre. A hash függvény ezen jellemzői miatt egy adott célt kielégítő hash érték csak úgy állítható elő, ha újra és újra próbálkozunk, és a bement értékét mindaddig változtatjuk, amíg a kívánt hash eredmény véletlenül elő nem áll.

Munkabizonyíték algoritmus

Egy hash algoritmus egy tetszőleges hosszúságú bemenő adatból egy fix hosszúságú, determinisztikus kimenetet állít elő: a bemenet digitális ujjlenyomatát. Egy adott bemenet esetén az eredményként kapott hash mindenkor ugyanaz lesz. Az eredményt bárki könnyen kiszámíthatja és ellenőrizheti, ha lefuttatja ugyanazt a hash algoritmust. A kriptográfiai hash algoritmusok alapvető jellemzője, hogy lényegében lehetetlen két olyan bemenetet találni, amely ugyanazt az ujjlenyomatot állítja elő. Ennek következtében az is lehetetlen, hogy egy adott ujjlenyomathoz találunk egy bemenetet, amely épp ezt az ujjlenyomatot állítja elő. Csak próbálhatni tudunk: egy véletlenszerű bemenetnek kiszámítjuk az ujjlenyomatát, és ellenőrizzük, hogy ez az ujjlenyomat egyezik-e a megadott ujjlenyomattal.

Az SHA256 esetén a bemenet méretétől függetlenül a kimenet minden 256 bit hosszú. A lenti 8-8. példa egy Python interpreterrel számítja ki az „I am Satoshi Nakamoto” kifejezés SHA256 hash értékét.

8-8. példa: SHA256 példa

```
$ python
Python 2.7.1
>>> import hashlib
>>> print hashlib.sha256("I am Satoshi Nakamoto").hexdigest()
5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e
```

A 8-8. példa szerint az "I am Satoshi Nakamoto" kifejezés hashe 5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e. Ez a 256 bites szám a kifejezés hash-e vagy *zanzája*, és a kifejezés minden egyes karakterétől függ az értéke. Már egyetlen egy betű megváltoztatása is teljesen különböző hash értéket eredményez.

Ha megváltoztatjuk ezt a kifejezést, akkor azt várjuk, hogy teljesen különböző hash értékeket kapunk. Próbáljuk ezt ki. Egy egyszerű script (8-9. példa) segítségével tegyünk a kifejezés végére egy számot:

8-9. példa: Nonce növeléssel SHA256 hasheket előállító script

```
$ cat hash_example.py

# example of iterating a nonce in a hashing algorithm's input

import hashlib

text = "I am Satoshi Nakamoto"

# iterate nonce from 0 to 19
for nonce in xrange(20):

    # add the nonce to the end of the text
    input = text + str(nonce)

    # calculate the SHA-256 hash of the input (text+nonce)
    hash = hashlib.sha256(input).hexdigest()

    # show the input and hash result
    print input, '=>', hash
```

Ennek futtatásával azoknak a kifejezéseknek a hashei állíthatók elő, melyek a fenti szöveg és egy szám összefűzésével álltak elő. A szám növelésekor különböző hasheket kapunk, ezt a 8-10. példa mutatja.

8-10. példa: A nonce növeléssel SHA256 hasheket előállító script kimenete

```
$ python hash_example.py
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbab...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
```

```
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

Minden egyes kifejezésnek teljesen más a hash értéke. A hash értékek teljesen véletlenszerűnek látszanak, ugyanakkor az eredmények egy másik számítógépen futó Python-nal pontosan reprodukálhatók, és pontosan ugyanezeket a hash értékeket eredményezik.

Az ilyen célra használt számláló neve: *nonce*. A nonce segítségével befolyásolhatjuk a kriptográfiai függvény kimenetét, ebben az esetben a kifejezés SHA256 ujjlenyomatát.

Egy cél nehézségi szint megadásával állítsuk kihívás elé ezt az algoritmust: próbáljuk egy olyan kifejezést találni, amelynek hashe 0-val kezdődik. Szerencsére, ez nem olyan nehéz! A 8-10. példa szerint az „I am Satoshi Nakamoto13” által előállított hash értéke

0ebc56d59a34f5082aaef3d66b37a661696c2b618e62432727216ba9531041a5, amely megfelel ennek a követelménynek. 13 kísérlet kellett hozzá, hogy megtaláljuk ezt a stringet. A valószínűségszámítás nyelvén, ha a hash függvény kimenete egyenletes eloszlású, akkor várhatóan minden 16 darab hash között lesz egy olyan, amely a 0 hexadecimális számjeggyel kezdődik (mivel a 16 hexadecimális számjegy, 0 .. F között ez az egyik számjegy).

Számszerűsítve mindenzt, egy olyan hash értéket kerestünk, amely kisebb mint

Egy egyszerű hasonlattal élve, képzeljünk el egy olyan játékot, amelyben a játékosok két kocka ismételt feldobását végzik, és próbálnak egy olyat dobni, ahol az összeg egy adott célnál kevesebb. Első körben legyen a cél 12. Ekkor mindegyik dobás megfelelő, kivéve, ha minden kockával 6-ost dobnak. A következő körben legyen a nehézség 11. Ekkor 10-et vagy kevesebbet kell dobniuk a játékosoknak, ami ismét csak könnyű feladat. Néhány körrel később legyen a cél 6. Ekkor a kockadobások több mint felénél az összeg több lesz mint 5, vagyis a dobás sikertelen lesz. Minél kisebb a cél, annál több kockadobásra van szükség a győzelemhez, és a kockadobások száma exponenciálisan nő. Végül, ha a cél 3 (a lehetséges minimum), akkor minden 36 dobásból 1 (2%) fog győzelemhez vezetni.

A 8-10. példában a „nyerő” nonce 13 volt, amit bárki ellenőrizhet, ha hozzáteszi az „I am Satoshi Nakamoto” string végéhez a 13-as számot, és kiszámíthatja az „I am Satoshi Nakamoto13” kifejezés hash értékét. Látni fogja, hogy a hash értéke kisebb, mint a cél. A sikeres eredmény egyúttal egy munkabizonyítékot (Proof-of-Work) jelent, mivel bizonyítja, hogy elvégeztük a fenti nonce megkereséséhez szükséges munkát. Az ellenőrzéséhez csak 1 darab hash kiszámítására van szükség, ugyanakkor a feltételnek megfelelő nonce érték előállításához 13 darab hash kiszámításra volt szükség. Ha alacsonyabb a cél (magasabb a bonyolultság), akkor a cél feltételt kielégítő nonce előállításához sokkal több hash kiszámításra lett volna szükség, de az ellenőrzéséhez továbbra is csak egy hash kiszámítása szükséges. Ha tudjuk a célt, akkor ki tudjuk számítani a bonyolultságot, és ennek megfelelően tudjuk, hogy mennyi munka szükséges ahhoz, hogy egy ilyen nonce értéket találjunk.

A Bitcoin munkabizonyítéka nagyon hasonló a fenti 8-10. példában szereplő munkabizonyítékoz. A bányász előállít egy jelölt blokkot, amelyet tranzakciókkal tölt föl. Ezt követően a bányász kiszámítja a blokkfej hashét, és megvizsgálja, hogy az kisebb-e, mint az aktuális cél. Ha a hash értéke nem kisebb a célnál, akkor a bányász módosítja a nonce-t (általában úgy, hogy megnöveli eggyel), és újra próbálkozik. A bitcoin hálózat jelenlegi bonyolultsági szintje mellett a bányászoknak sok billiószor kell próbálkozniuk ahhoz, hogy egy olyan nonce értéket találjanak, amely elég kicsiny blokkfej hash értéket eredményez.

Az alábbi 8-11. példa egy nagyon leegyszerűsített munkabizonyíték algoritmust (Proof-of-Work algorithm) valósít meg Python-ban:

8-11. példa: Egy egyszerű munkabizonyíték algoritmus

```
$ cat proof-of-work-example.py
#!/usr/bin/env python
# example of proof-of-work algorithm

import hashlib
import time

max_nonce = 2 ** 32 # 4 billion

def proof_of_work(header, difficulty_bits):

    # calculate the difficulty target
    target = 2 ** (256-difficulty_bits)

    for nonce in xrange(max_nonce):
        hash_result = hashlib.sha256(str(header)+str(nonce)).hexdigest()

        # check if this is a valid result, below the target
        if long(hash_result, 16) < target:
            print "Success with nonce %d" % nonce
            print "Hash is %s" % hash_result
            return (hash_result,nonce)

    print "Failed after %d (max_nonce) tries" % nonce
    return nonce

if __name__ == '__main__':

    nonce = 0
    hash_result = ''

    # difficulty from 0 to 31 bits
    for difficulty_bits in xrange(32):

        difficulty = 2 ** difficulty_bits
        print "Difficulty: %ld (%d bits)" % (difficulty, difficulty_bits)

        print "Starting search..."

        # checkpoint the current time
        start_time = time.time()

        # make a new block which includes the hash from the previous block
        # we fake a block of transactions - just a string
        new_block = 'test block with transactions' + hash_result
```

```

# find a valid nonce for the new block
(hash_result, nonce) = proof_of_work(new_block, difficulty_bits)

# checkpoint how long it took to find a result
end_time = time.time()

elapsed_time = end_time - start_time
print "Elapsed Time: %.4f seconds" % elapsed_time

if elapsed_time > 0:

    # estimate the hashes per second
    hash_power = float(long(nonce)/elapsed_time)
    print "Hashing Power: %ld hashes per second" % hash_power

```

A fenti kód futtatásakor beállítható a kívánt nehézség (bitekben, vagyis hogy hány bit legyen a hash elején nulla), és megvizsgálható, hogy mennyi idő szükséges egy megoldáshoz. A 8-12. példa mutatja, hogy egy átlagos laptopon hogyan működik az algoritmus:

8-12. példa: A munkabizonyíték példa futtatása különféle nehézségi szintekre

```

$ python proof-of-work-example.py*
Difficulty: 1 (0 bits)

[...]

Difficulty: 8 (3 bits)
Starting search...
Success with nonce 9
Hash is 1c1c105e65b47142f028a8f93ddf3dabb9260491bc64474738133ce5256cb3c1
Elapsed Time: 0.0004 seconds
Hashing Power: 25065 hashes per second
Difficulty: 16 (4 bits)
Starting search...
Success with nonce 25
Hash is 0f7becfd3bcd1a82e06663c97176add89e7cae0268de46f94e7e11bc3863e148
Elapsed Time: 0.0005 seconds
Hashing Power: 52507 hashes per second
Difficulty: 32 (5 bits)
Starting search...
Success with nonce 36
Hash is 029ae6e5004302a120630adccb808452346ab1cf0b94c5189ba8bac1d47e7903
Elapsed Time: 0.0006 seconds
Hashing Power: 58164 hashes per second

[...]

Difficulty: 4194304 (22 bits)
Starting search...
Success with nonce 1759164
Hash is 0000008bb8f0e731f0496b8e530da984e85fb3cd2bd81882fe8ba3610b6cef3
Elapsed Time: 13.3201 seconds
Hashing Power: 132068 hashes per second
Difficulty: 8388608 (23 bits)
Starting search...
Success with nonce 14214729
Hash is 000001408cf12dbd20fcba6372a223e098d58786c6ff93488a9f74f5df4df0a3
Elapsed Time: 110.1507 seconds
Hashing Power: 129048 hashes per second

```

```
Difficulty: 16777216 (24 bits)
Starting search...
Success with nonce 24586379
Hash is 0000002c3d6b370fccd699708d1b7cb4a94388595171366b944d68b2acce8b95
Elapsed Time: 195.2991 seconds
Hashing Power: 125890 hashes per second
```

[...]

```
Difficulty: 67108864 (26 bits)
Starting search...
Success with nonce 84561291
Hash is 0000001f0ea21e676b6dde5ad429b9d131a9f2b000802ab2f169cbca22b1e21a
Elapsed Time: 665.0949 seconds
Hashing Power: 127141 hashes per second
```

Mint látható, a nehézségi szint 1 bittel történő megnövelése exponenciálisan növeli a megoldás megtalálásához szükséges időt. Ha az egész 256-bites számteret tekintjük, akkor minden egyes alkalommal, amikor egy további bittől követeljük meg, hogy nulla legyen, azzal megfelezzük a keresési tért. A fenti 8-12. példában 84 millió próbálkozás kellett ahhoz, hogy egy olyan nonce értéket találunk, amelynél a hash első 26 bitje nulla. Még 120 ezer hash/másodperc sebességnél is több mint 10 percbe került, hogy egy közönséges laptop-on megtaláljuk ezt a megoldást.

Amikor e sorokat írom, a hálózat olyan blokkot próbál találni, amelynél a blokkfej hash értéke kevesebb, mint

0000000000000004c296e6376db3a241271f43fd3f5de7ba18986e517a243baa7. Mint látják, sok-sok nulla van a hash elején, ami azt jelenti, hogy a megengedhető hash tartomány sokkal kisebb, emiatt sokkal nehezebb egy ennek megfelelő hash értéket találni. Átlagosan több mint 150 ezer billió hash számításra van szükség másodpercenként, hogy a hálózat megtalálja a következő blokkot. Ez szinte megoldhatatlan feladatnak látszik, de szerencsére a hálózat 100 Petahash/sec számítási teljesítménnyel rendelkezik, ami lehetővé teszi, hogy átlagosan 10 perc alatt találjon egy blokkot.

A nehézségi szint ábrázolása

A 8-3. példánál láttuk, hogy a blokkfej tartalmazza a nehézségi célt, olyan jelölésmódban, amit „nehézségi szintet meghatározó biteknek” vagy egyszerűen csak „biteknek” hívunk. A 277'316-ik blokk esetén ez az érték 0x1903a30c. Ez a jelölésmód mantissa/kitevő formátumban fejezi ki a kívánt nehézségi szintet, ahol az első két hexa számjegy a kitevő, a következő hat hexa számjegy pedig a mantissa. Ennek megfelelően ebben a blokkban a kitevő 0x19, a mantissa pedig 0x03a30c.

A kívánt nehézségi szint ebből az ábrázolásból a következő képlettel számítható ki:

$$\text{cél} = \text{mantissa} * 2^{(8 * (\text{kitevő} - 3))}$$

Ha a képletet a 0x1903a30c nehézségi bitekre vonatkozóan használjuk, akkor azt kapjuk, hogy:

$$\text{cél} = 0x03a30c * 2^{(0x08 * (0x19 - 0x03))}$$

$$\Rightarrow \text{cél} = 0x03a30c * 2^{(0x08 * 0x16)}$$

$\Rightarrow \text{c}_e1 = 0x03a30c * 2^{0xB0}$

ami decimálisan:

$$\Rightarrow \text{cél} = 238'348 * 2^{176}$$

```
⇒ cél =  
22'829'202'948'393'929'850'749'706'076'701'368'331'072'452'018'388'575'715'  
328
```

Ezt hexadecimális alakra visszaírva:

Ez azt jelenti, hogy a 277'316 magasságban az a blokk érvényes, melynél a blokkfej hashének értéke kisebb, mint a cél. Binárisan ez a szám több, mint 60 db nullával kezdődik. Ennél a nehézségi szintnél egy olyan bányász, amely másodpercenként 1 millió hash értéket képes kiszámítani (ami másképpen 1 terahash másodpercenként, azaz 1 TH/sec), átlagosan csak minden 8496 blokkonként (vagyis 59 naponta) fog egy megoldást találni.

A cél nehézségi szint és a nehézségi szint újraszámítása

Mint azt fent láttuk, a cél határozza meg a nehézségi szintet, és emiatt közvetlenül befolyásolja, hogy mennyi idő szükséges a munkabizonyíték (Proof-of-Work) algoritmus megoldásához. Ez viszont felveti a következő nyilvánvaló kérdést: ha a nehézségi szint állítható, akkor ki állítja és hogyan?

A bitcoin blokkjai átlagosan 10 percenként állnak elő. Ez a bitcoin szívverése, ami a pénzkibocsátás gyakoriságát és a tranzakciók elszámolását határozza meg. Nem csak rövid távon, hanem hosszú évtizedek során is állandónak kell maradnia. Azt várjuk, hogy idővel gyors ütemben nő majd a számítási kapacitás. Ezen kívül a bányászatban részt vevők száma és az általuk használt berendezések száma szintén állandóan változik. Ha azt szeretnénk, hogy a blokk előállítás ideje 10 perc maradjon, a bányászat nehézségét úgy kell szabályozni, hogy figyelembe vegye ezeket a körülményeket. És valóban, a nehézségi szint egy dinamikus paraméter, amelynek időről időre történő állításával elérhető, hogy teljesüljön a 10 perces blokk előállítási idő. Leegyszerűsítve, a cél nehézségi szint mindenkorral úgy fog beállni, hogy a bányászok teljesítményétől függetlenül 10 perc legyen a blokkok közötti idő.

Hogyan lehet egy teljesen decentralizált hálózatban egy ilyen beállítást elvégezni? A cél nehézségi szint újraszámítása minden teljes csomópontron automatikusan és a többi csomóponttól teljesen függetlenül történik. minden 2016 darab blokk után mindegyik csomópont újraszámítja a nehézségi szintet. A nehézségi szint újraszámítására szolgáló képletben a 2016 darab blokk tényleges előállításához szükséges időt hasonlítják össze a várt 20'160 perces értékkel (ami a kívánatos 10 perces blokk idő esetén két héttel). A ténylegesen eltelt idő és a kívánt idő hányadosának kiszámítása után megtörténik a nehézségi szint szükséges korrekciója (fölfelé vagy lefelé). Leegyszerűsítve: Ha a hálózat átlagosan 10 percnél hamarabb találja meg a blokkokat, akkor a nehézségi szint nő. Ha a blokkok előállítása lassabb a vártnál, akkor a nehézségi szint csökken.

Az egyenlet a következőképpen foglalható össze:

`Új_nehézség = Régi_nehézség * (Az_utolsó_2016_blokk_előállításának_ideje / 20160 perc)`

A 8-13. példa a Bitcoin Core kliensen belül használt kódot mutatja.

8-13. példa: A munkabizonyíték nehézségi szintjének újraszámítása – `GetNextWorkRequired()`, `pow.cpp`, 43-ik sor

```
// Visszalépünk annyival, hogy 14 napnak megfelelő blokkunk legyen
const CBlockIndex* pindexFirst = pindexLast;
for (int i = 0; pindexFirst && i < Params().Interval()-1; i++)
    pindexFirst = pindexFirst->pprev;
assert(pindexFirst);

// Korlátozzuk a módosítás mértékét
int64_t nActualTimespan = pindexLast->GetBlockTime() - pindexFirst-
>GetBlockTime();
LogPrintf(" nActualTimespan = %d before bounds\n", nActualTimespan);
if (nActualTimespan < Params().TargetTimespan()/4)
    nActualTimespan = Params().TargetTimespan()/4;
if (nActualTimespan > Params().TargetTimespan()*4)
    nActualTimespan = Params().TargetTimespan()*4;

// Újraszámítjuk a nehézségi szintet
uint256 bnNew;
uint256 bnOld;
bnNew.SetCompact(pindexLast->nBits);
bnOld = bnNew;
bnNew *= nActualTimespan;
bnNew /= Params().TargetTimespan();

if (bnNew > Params().ProofOfWorkLimit())
    bnNew = Params().ProofOfWorkLimit();
```



Míg a nehézségi szint újraszámítása 2016 blokkonként történik, az eredeti Bitcoin Core kliens egy hiba miatt az előző 2015 blokk idejét veszi figyelembe (nem 2016 darabét, ahogyan azt kellene). Emiatt a nehézségi szint 0.05%-kal magasabb lesz, mint kellene.

Az Interval paramétert (2016 blokk) és a TargetTimespan paramétert (két hét, azaz 1'209'600 másodperc) a `chainparams.cpp` definiálja.

A nehézségi szint nagy ingadozásainak elkerülése érdekében a módosító tényezőnek ciklusonként (2016 blokkonként) 4-nél kisebbnek kell lennie. Ha a kívánt nehézségi szint módosulás több mint négyszeres lenne, akkor 4-re lesz állítva. A további állítások a következő körben fognak megvalósulni, és a következő 2016 blokkban fennmarad az egyensúlyhiány. Emiatt a hash kapacitás és a nehézségi szint közötti nagy eltérések kiegyenlítődéséhez egynél több 2016 blokkos ciklusra lehet szükség.



A teljes bitcoin hálózatban kb. 10 perc szükséges az egész hálózat számára egy blokk előállításához, ami az előző 2016 blokk előállításához szükséges idő alapján, 2016 blokkonként újraszabályozásra kerül.

Figyeljék meg, hogy a cél nehézségi szint független a tranzakciók számától vagy értékétől. Ez azt jelenti, hogy a bitcoin hálózat biztonságának megteremtésre fordított hash kapacitás, vagyis villamos energia szintén teljesen független a tranzakciók számától. Ha a bitcoin elterjedtebbé válik, akkor sem lesz arra szükség, hogy a biztonság érdekében a hash kapacitás a mai szinthez képest tovább növekedjen. A hash kapacitás növekedése olyan piaci erőknek tudható be, mint a jutalomért versengő újabb bányászok megjelenése. Amíg a jutalomért versenyző bányászok között elég sok becsületes bányász van, addig nem lehetséges a hálózat „kisajtítása”, és emiatt a bitcoin biztonságos marad.

A cél nehézségi szint szoros kapcsolatban a villamos energia árával és a bitcoin átváltási árfolyamával, t.i. a bányászok a villanyszámlát hagyományos papírpénzzel fizetik. A nagy kapacitású bányász „farmok” a lehető leggazdaságosabban működnek, a legmodernebb integrált áramköröket (ASIC) használják, és a villamos energiát a lehető leghatékonyabban alakítják át hash számításokká. A bányászokra a legközvetlenebb hatást 1 kWh bitcoinban mért ára jelenti, mivel ez határozza meg a bányászat jövedelmezőségét, vagyis azt, hogy érdemes-e belépni erre a piakra, vagy be kell szüntetni a tevékenységet.

A blokk sikeres kibányászása

Mint azt korábban láttuk, Jing csomópontja előállított egy blokk jelöltet és előkészítette a bányászathoz. Jingnek számos ASIC (ASIC = Application Specific Integrated Circuit, BOÁK, Berendezés Orientált Árakör) alapú bányász berendezése van, amelyekben az integrált áramkörök SHA256 algoritmusok ezreit futtatók egymással párhuzamosan, hihetetlen sebességgel. Ezek a specializált gépek USB-vel kapcsolódnak hozzá a bányász csomóponthoz. A Jing asztali számítógépén futó bányász csomópont elküldi a blokkfejet a bányász hardvernek, ami több billió nonce / másodperc sebességgel elkezdi tesztelni a nonce-okat.

A 277'316-ik blokk bányászatának megkezdése után majdnem 11 perccel az egyik hardver talál egy megoldást, és visszaküldi a bányász csomópontnak. A nonce $4'215'469'401$, melyet a következő blokkhaszt eredményezi:

0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569

amely kisebb, mint a megkívánt cél:

Jing számítógépe azonnal elküldi a blokkot a szomszédos csomópontoknak. Ezek fogadják, ellenőrzik, majd továbbítják az új blokkot. Amint a blokk szétterjed a hálózaton, mindenki csomópont hozzáadja ezt a blokkot a saját blokkláncához, így a blokklánc magassága 277'316-ra nő. Miután a bányász csomópontok megkapták és ellenőrizték az új blokkot, abbahagyják annak blokk keresését, melynek a 277'315-ik blokk a szülője, és azonnal hozzálátnak a lánc következő blokkjának a kiszámításához.

A következő részben azt a folyamatot fogjuk megvizsgálni, amellyel az egyes csomópontok ellenőrizik a blokkot és kiválasztják a leghosszabb láncot, megteremtve ezáltal azt a közmegegyezést, amely a decentralizált blokklánc létrejöttének az alapja.

Az új blokk ellenőrzése

A bitcoin konszenzus mechanizmusának harmadik eleme a blokkok egymástól független ellenőrzése, amely a hálózat minden egyes csomópontján végbemegy. Az újonnan keletkezett blokkok hálózati szétterjedésekor mindegyik csomópont egy tesztsorozatot végez, mielőtt a blokkot továbbítaná a peerjeinek. Ez biztosítja, hogy a hálózatban csak érvényes blokkok terjedhessenek szét. A blokkok független ellenőrzése azt is biztosítja, hogy a becsületes bányászok blokkjai beépülnek a blokkláncba, és a bányász ezáltal hozzájut a jutalmához. A csaló bányászok blokkjait viszont a többiek elvetik, és így a bányász nem csupán a jutalomtól esik el, hanem a munkabizonyíték előállításához elvégzett munkája is pocsékba megy, vagyis a villamos energia költségét nem kompenzálja semmi.

Ha egy csomópontba új blokk érkezik, akkor a csomópont a blokkot egy hosszú feltételezés alapján ellenőrzi. A blokknak az össze feltételt teljesítenie kell, különben a blokkot elveti a csomópont. Ezek a feltételek a Bitcoin Core kliens `CheckBlock` és `CheckBlockHeader` függvényeiben találhatók meg. A feltételek a következők:

- a blokk adatstruktúrája szintaktikusan érvényes
- a blokkfej hashe kisebb, mint a cél nehézségi szint (a munkabizonyíték betartatása)
- a blokk időbelyege kevesebb, mint két órával mutat a jövőbe (óra hibák engedélyezése)
- a blokk mérete a megengedett határok között van
- az első (és csak az első) tranzakció egy coinbase tranzakció
- „[A tranzakciók egymástól független ellenőrzése](#)” tranzakciós ellenőrző lista alapján a blokkban lévő összes tranzakció érvényes

A hálózat mindegyik csomópontja által, a többi csomóponttól függetlenül elvégzett ellenőrzések biztosítják, hogy a bányászok ne csalhassanak. Az előző részben láttuk, hogy egy bányász hogyan tud olyan tranzakciót létrehozni, amely a blokkban létrejött bitcoinokat és a tranzakciós díjakat a bányász saját címére utalja. Miért nem írnak a bányászok ebbe a tranzakcióba több ezer bitcoint a jutalom helyes összege helyett? Mert mindegyik csomópont ugyanazon szabályok szerint ellenőrzi a blokkokat. Egy érvénytelen coinbase tranzakció az egész blokkot érvénytelenné tenné, ami miatt a többi csomópont elvetné a blokkot és a blokk soha nem válna a főkönyv részévé. Egy bányásznak tökéletesen helyes blokkot kell előállítania, amely megfelel a többi csomópont által követett közös szabályoknak, és a munkabizonyíték helyes megoldását tartalmazza. A bányászathoz sok villamos energia szükséges, ami csalás esetén nem térül meg. A blokkok független ellenőrzése emiatt kulcsfontosságú a decentralizált konszenzus megteremtése szempontjából.

A blokklánc összeállítása és kiválasztása

A bitcoin rendszer decentralizált konszenzus mechanizmusának utolsó lépése a blokkok láncokba történő szervezése, és annak a blokkláncnak a kiválasztása, amely a legtöbb munkabizonyítékot tartalmazza. Ha egy csomópont elvégezte a blokk ellenőrzését, akkor megkísérli a blokklánc bővítését, vagyis a blokk és a már létező blokklánc összekapcsolását.

A csomópontok háromféle blokkot tartanak nyilván: a fő blokklánchoz kapcsolódó blokkokat, a fő blokklánc elágazásain lévő blokkokat (másodlagos láncok) és végül azokat a blokkokat, amelyeknek az ismert blokkláncban nincsenek szülei (árva blokkok). Ha valamelyik

ellenőrzési feltétel nem teljesül, akkor az érvénytelen blokkot a csomópont azonnal elveti, emiatt a blokk egyik blokkláncba sem kerül be.

A „fő lánc” egy adott időpillanatban az a blokklánc, amelyben a nehézségi szint kumulált értéke a legnagyobb. A legtöbb esetben ez azonos a legtöbb blokkot tartalmazó láncnal, kivéve, ha két lánc azonos hosszúságú, mert ekkor a több munkabizonyítékot tartalmazó lánc lesz a fő lánc. A fő láncról elágazó láncok blokkjai a fő láncon lévő blokkok „testvérei”. Ezek a blokkok érvényesek ugyan, de nem részei a fő láncnak. Azért tartják meg őket, mert előfordulhat, hogy valamelyik elágazó lánc bővülése miatt az elágazó lánc nehézségi szintje meghaladja a fő láncét. A következő részben („[Blokklánc elágazások](#)”) látni fogjuk, hogy hogyan jönnek létre másodlagos láncok, ha két bányász majdnem egyidőben azonos magasságban lévő blokkokat bányászik ki.

Egy új blokk beérkezése után a csomópont megpróbálja beilleszteni a blokkot a létező blokkláncba. A csomópont megvizsgálja a blokk „előző blokk hashe” mezőjét, amely a blokk szülőjére hivatkozik. Ezután a csomópont a létező blokkláncban megpróbálja megkeresni ezt a szülőt. A legtöbbször a szülő a fő lánc „csúcsán” lesz, vagyis az új blokk a fő láncot fogja meghosszabbítani. Például a 277'316-ik blokk a 277'315-ik, szülő blokk blokk hashére hivatkozik. A legtöbb csomópont a 277'316-ik blokk megérkezésekor már rendelkezik a 277'315-ik blokkal, és a 277'315-ik blokk van a fő láncuk legtetején, vagyis az új blokk beláncolása ezt a láncot fogja kibővíteni.

Néha, amint azt a „[Blokklánc elágazások](#)” részben látni fogjuk, az új blokk nem a fő láncot bővíti ki. Ebben az esetben a csomópont a másodlagos lánchoz kapcsolja hozzá az új blokkot, és azután összehasonlítja a másodlagos lánc és a fő lánc nehézségi szintjét. Ha a másodlagos lánc összegzett nehézsége nagyobb, mint a fő láncé, akkor a csomópont átkonvergál a másodlagos láncra, ami azt jelenti, hogy a másodlagos láncából fő lánc válik, a korábbi fő láncából pedig másodlagos lánc. Ha a csomópont bányász csomópont, akkor az általa létrehozott blokk ezt az új „hosszabb” láncot fogja bővíteni.

Ha egy olyan érvényes blokk érkezik, melynek a létező láncban nincs szülője, akkor a blokkot a csomópont „árvának” tekinti. Az árva blokkokat az árva blokkok készletébe helyezi, és a blokk egészen addig itt fog maradni, amíg meg nem érkezik a szüleje. Ha megérkezett a szülő, és a csomópont beillesztette a szülőt a létező blokkláncba, akkor az árva blokkot kiveszi az árva blokkok készletéből, és összekapcsolja a szülőjével, vagyis a lánc részévé teszi. Árva blokkok általában akkor fordulnak elő, ha egymás után kis időeltéréssel két blokk kerül kibányászásra, és a blokkok fordított sorrendben érkeznek meg (a gyermek előbb, mint a szülő).

A legnagyobb nehézségű lánc kiválasztása révén az összes csomópont előbb-utóbb egyezségre jut. A láncok közötti átmeneti eltéréseket végül a további munkabizonyíték hozzáadása oldja fel, amely bővíti valamelyik lehetséges láncot. A bányász csomópontok a következő blokk kibányászásakor a számítási kapacitásukkal „szavaznak”, mert ők választják ki, hogy melyik láncot akarják bővíteni. Ha sikerül kibányászniuk egy új blokkot, és kibővíteni vele a láncot, akkor maga az új blokk jelenti a szavazatukat.

A következő részben megnézzük, hogy az egymással versenyző láncok közötti eltéréseket (elágazásokat) hogyan oldja föl az, hogy az egyes csomópontok egymástól függetlenül a legnagyobb nehézségű láncot választják.

Blokklánc elágazások

Mivel a blokklánc egy decentralizált adatszerkezet, a különböző példányai nem minden konzisztensek. Az egyes csomópontokhoz a blokkok eltérő időben jutnak el, emiatt a csomópontok más hogy fogják látni a blokkláncot. Ennek feloldása érdekében mindegyik csomópont minden azt a láncot választja és bővíti, amely a legtöbb munkabizonyitéknak felel meg, vagyis a legnagyobb összegzett nehézségi szinttel rendelkező láncot. A lánc blokkjaiban lévő nehézségi szint összegzése révén a csomópont kiszámítja, hogy az adott láncnak a létrehozásához összesen mennyi munkabizonyitékra volt szükség. Ha a hálózat mindegyik csomópontja a legnagyobb összegzett nehézségű láncot választja, a globális bitcoin hálózat előbb-utóbb konzisztens állapotba kerül. Az elágazások csupán átmeneti inkonzisztenciát jelentenek a blokklánc különféle változatai között. Az elágazások az átkonvergálás révén oldódnak meg, vagyis azáltal, hogy valamelyik elágazás újabb blokkokkal bővül.

A következő néhány ábrán egy „elágazási” eseményt követünk nyomon a hálózatban. Az ábra a globális bitcoin hálózat egy egyszerűsített ábrázolása. A valóságban a bitcoin hálózat nem földrajzi szerveződésű, hanem egymással kapcsolatban lévő csomópontok hálózata, melyek földrajzilag nagyon távol is lehetnek egymástól. A földrajzi topológia egy egyszerűsítés, amelyet az elágazás szemléltetése érdekében használunk. A valódi bitcoin hálózatban a csomópontok közötti „távolságot” a csomópontok közötti „ugrásokkal” („hops”) mérjük, nem pedig a földrajzi helyzet alapján. A szemléltetés kedvéért a különböző blokkokat különböző színekkel jelöltük.

Az lenti első ábrán (8-2. ábra) a hálózat a blokkláncot egységesnek látja. A kék blokkok a fő lánc legtetején vannak.



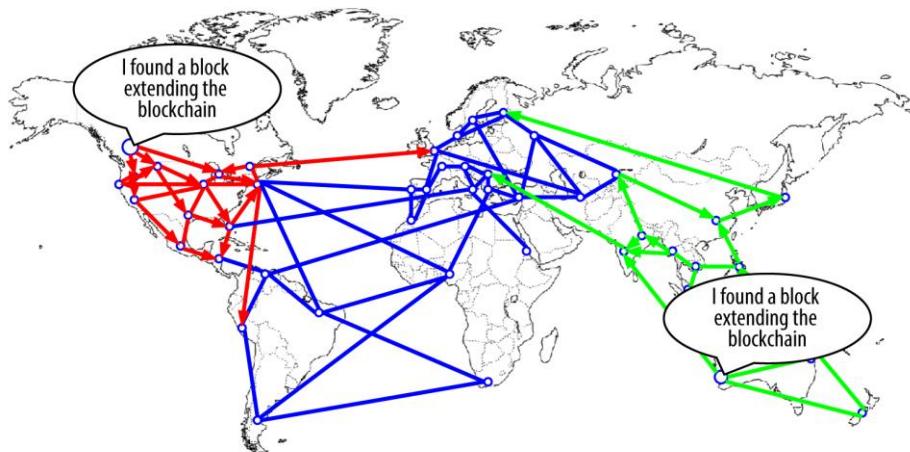
8-2. ábra: Egy blokklánc elágazás szemléltetése – az elágazás előtti állapot

„Elágazás” akkor fordul elő, ha két blokkjelölt versenyez egymással, hogy melyik van a leghosszabb blokkláncon. Normális körülmények között ez akkor fordul elő, ha két bányász egymáshoz képest viszonylag kis időkülönbséggel egy-egy új blokkot állított elő. Mindkét bányász azonnal közvetíti a „nyerő” blokkot, mihelyt sikerült megoldania a munkabizonyíték algoritmust, és a blokkok a közvetlen szomszédaikon keresztül kezdenek szétterjedni a hálózatban. Mindegyik csomópont, amelyik megkapja az érvényes blokkot, beépíti azt a blokkláncába, vagyis bővíti a saját blokkláncát egy blokkal. Ha a csomópont később egy másik blokkot lát, amely ugyanazt a szülőt bővíti, akkor a második blokkot egy másodlagos

láncra helyezi. Ennek eredményeképpen bizonyos csomópontok az egyik blokkot fogják „látni” először, míg a többiek a másikat, és a blokklánc két egymással versengő változata jön létre.

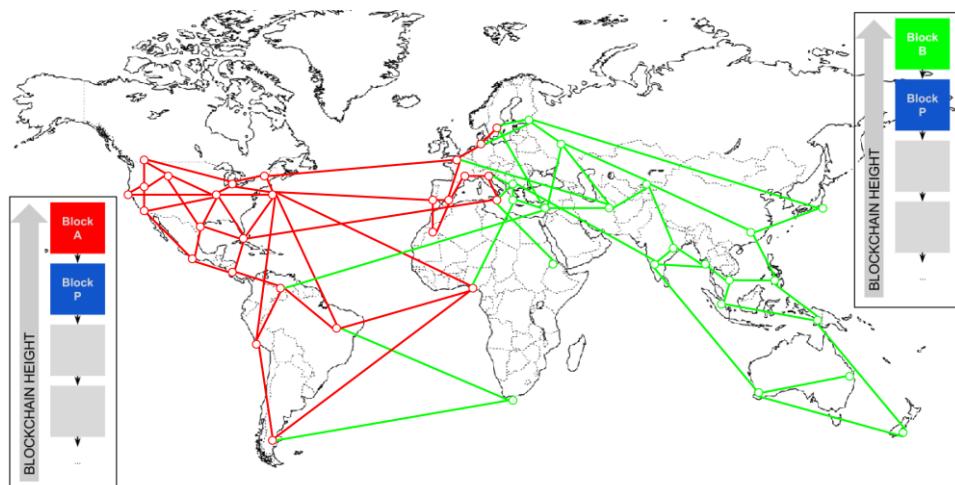
A 8-3. ábrán két bányászt látunk, akik majdnem egyidőben két különböző blokkot bányásztak ki. Mindkét kibányászott blokk a kék blokk gyermeke, vagyis a láncot a kék blokk tetején bővíti. Hogy az események jobban nyomon követhetők legyenek, az egyik blokkot egy Kanadából származó piros blokként ábrázoltuk, míg a másikat egy Ausztráliából származó zöld blokként.

Tegyük fel például, hogy a kanadai bányász a „piros” blokkhoz talált egy olyan munkabizonyítékot, amely a „kék” szülő blokkláncra épül. Majdnem ugyanekkor egy ausztrál bányász, aki szintén a „kék” blokkláncot szeretné bővíteni, talál egy megoldást a „zöld” blokkjához. Két lehetséges blokkunk van tehát, a „piros”, ami Kanadából indul, és a „zöld”, ami Ausztráliából. Mindkét blokk érvényes, mindenki blokk érvényes munkabizonyítékot tartalmaz, mindenki ugyanazt a születéstől bővíti. Valószínűleg minden blokkban ugyanazok a tranzakciók vannak, de a tranzakciók sorrendjében előfordulhatnak különbségek.



8-3. ábra: Egy blokklánc elágazás szemléltetése – egyszerre két blokk jött létre

A két blokk szétterjedése során némelyik csomóponthoz a „piros” blokk jut el először, míg másokhoz a „zöld”. Amint azt a 8-4. ábra mutatja, a hálózat két részre szakad, és minden rész másképpen látja a blokkláncot: az egyiknél a piros blokk van a blokklánc tetején, a másiknál a zöld.



8-4. ábra: Egy blokklánc elágazás szemléltetése – a két blokk szétterjedése során a hálózat két részre szakad

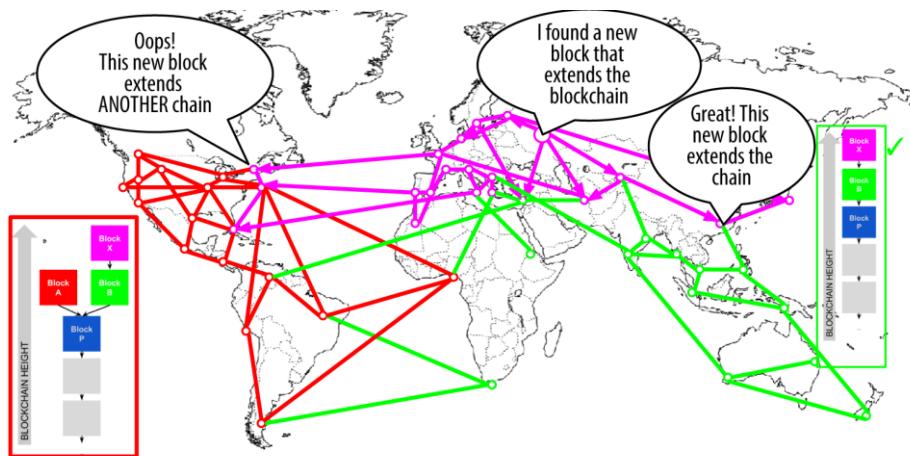
Ettől a pillanattól kezdve a bitcoin hálózat azon csomópontjai, melyek a kanadai csomóponthoz vannak topológiaileg a legközelebb, a „piros” blokkról fognak először tudomást szerezni, és a legnagyobb nehézségi szint összeggel rendelkező láncba a „piros” blokkot teszik be a lánc utolsó tagjaként (kék-piros), a „zöld” blokkot pedig, amely egy kicsit később érkezik, figyelmen kívül hagyják. Közben az ausztráliai csomóponthoz közelebb lévő csomópontok a „zöld” blokkot fogják nyertesnek tekinteni, és ezzel bővítik a blokkláncukat (kék-zöld), és elhanyagolják a néhány másodperccel ezután megérkező „pirosat”. Azok a bányászok, melyek a „pirosat” látták meg először, olyan blokk jelölteket hoznak létre, melyekben a „piros” a szülő blokk, és ezekkel a blokkokkal kezdik el a munkabizonyíték algoritmus megoldását. A „zöld” blokkot elfogadó bányászok viszont a „zöldre” épülő blokkal próbálják meg az új blokk létrehozását.

Az elágazások majdnem minden egy blokkon belül megoldódnak. A hálózat hash kapacitásának egyik része a „piros” blokkra épülő blokkot szeretne létrehozni, a másik része a „zöldre” épülőt. Még ha a hash kapacitás közel egyforma lenne is, akkor is valószínűtlen, hogy a két bányászcsoporthoz ismét közel egyszerre talál egy-egy megoldást. Ha valamelyik csoporthoz talál egy megoldást, az ezúttal zavartalanul szétterjed a hálózatban, mielőtt a másik csoporthoz találna egy megoldást. Mondjuk, hogy azok a bányászok, akik a „zöld” blokkra építettek, találtak egy „rózsaszín” új blokkot, amely ezt a láncot bővíti ki (vagyis kék-zöld-rózsaszín). Azonnal továbbítják a blokkot a többi csomópontnak, és a blokkot az egész hálózat érvényes megoldásnak látja, amint azt a 8-5. ábra mutatja.



8-5. ábra: Egy blokklánc elágazás szemléltetése – az elágazás egy újabb blokkal bővül

Azok a csomópontok, melyek az előző körben a „zöldet” tekintették győztesnek, egyszerűen kibővítenek a blokkláncukat egy blokkal. Azok a csomópontok azonban, melyek a „pirosat” tekintették győztesnek, most két láncot látnak: egy kék-zöld-rózsaszín láncot és egy kék-piros láncot. Ezek a csomópontok a kék-zöld-rózsaszín láncra váltanak át, és a továbbiakban ez lesz a fő lánc, a kék-piros láncot pedig másodlagos lánctársnak tekintik, lásd a 8-6. ábrát. Lánc átkonvergálás történik, mivel a csomópontok egy részének meg kell változtatnia a blokkláncról alkotott képét, hogy a hosszabb láncban lévő, új munkabizonyítékot tartalmazó blokkot be tudja építeni. Azok a bányászok, akik a kék-piros lánc bővítésén dolgoztak, most félbeszakítják ezt a munkát, mivel a blokkjuk, amelyen dolgoztak, „árvává” vált, hiszen a „piros” szülő blokk már nincs a leghosszabb láncban. A „piros” blokkban lévő tranzakciók ismét sorbaállnak, hogy bekerülhessenek a következő blokkba, mivel a blokk már nem a leghosszabb láncban van. Az egész hálózat átkonvergál a kék-zöld-rózsaszín blokkláncra, amelyben a „rózsaszín” a lánc utolsó eleme. Az összes bányász olyan blokk jelöltekben kezd dolgozni, amelyekben a „rózsaszín” blokk a szülő, és a blokk a kék-zöld-rózsaszín láncot fogja bővíteni.



8-6. ábra: Egy blokklánc elágazás szemléltetése – a hálózat átkonvergál a leghosszabb láncra

Elméletileg lehetséges, hogy egy elágazás két blokkon keresztül tartson, ha a hálózat két ellentétes „végén” két bányász majdnem egyszerre két blokkot talál. Ennek a bekövetkezési

valószínűsége nagyon kicsi. Míg egy blokkos elágazások hetente előfordulnak, a két blokkos elágazások nagyon ritkák.

A bitcoin 10 perces blokkideje tervezési kompromisszumot jelent a gyors megerősítési idő (tranzakciók elszámolása) és az elágazások létrejötte között. A kisebb blokkidő megyyorsítaná a tranzakciók feldolgozását, de gyakrabban lennének a blokklánc elágazások, míg a nagyobb blokkidő csökkentené az elágazások számát, de lassítaná az elszámolást.

Bányászat és versenyfutás a hash kapacitásban

A bitcoin bányászatban nagy a verseny. Amióta csak létezik a bitcoin, a hash kapacitás minden évben exponenciálisan nőtt. Néhány éve a növekedés a teljes technológiaváltást tükrözte, pl. 2010-ben és 2011-ben, amikor sok bányász CPU-ról GPU-ra (Graphical Processing Unit) és FPGA-ra (Field Programmable Gate Array) állt át. 2013-ban az ASIC-ok (Application Specific Integrated Circuits) bányászati alkalmazása újabb hatalmas ugrást okozott a bányász kapacitásban, mert az SHA256 függvény közvetlenül a bányászatra specializált szilícium integrált áramköri chipeken lett megvalósítva. Az ilyen chipekből összeállított első eszközöknek több kapacitása volt, mint az egész bitcoin hálózatnak 2010-ben.

A következő lista a bitcoin hálózat teljes hash kapacitását mutatja, a hálózat működésének első öt évében:

2009

0.5 MH/sec–8 MH/sec (16-szoros növekedés)

2010

8 MH/sec–116 GH/sec (14,500-szoros növekedés)

2011

16 GH/sec–9 TH/sec (562-szoros növekedés)

2012

9 TH/sec–23 TH/sec (2.5-szoros növekedés)

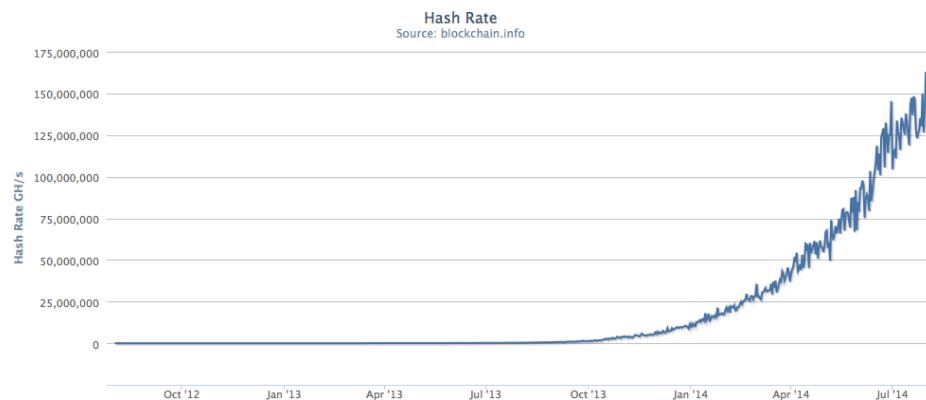
2013

23 TH/sec–10 PH/sec (450-szoros növekedés)

2014

10 PH/sec–150 PH/sec augusztusban (15-szörös növekedés)

A 8-7. ábrán a bitcoin hálózat hash kapacitásának a növekedése látható az elmúlt két évben. Látható, hogy a bányászok közötti versengés és a bitcoin elterjedése miatt a hash kapacitás (a hálózatban végezhető hashek száma másodpercenként) exponenciálisan növekedett.



8-7. ábra: A teljes hash kapacitás az elmúlt két évben, gigahash/sec-ben

A bitcoin bányászat hash kapacitásának növekedésével együtt a nehézségi szint is nőtt. A 8-8. ábrán a nehézségi szintet egy arányszám ábrázolja, amely az aktuális nehézségi szint és a minimális nehézségi szint hányadosa (a minimális nehézségi szint az első blokk nehézségi szintjének felel meg):



8-8. ábra: A bitcoin bányászat nehézsége az elmúlt két évben

Az elmúlt két évben az ASIC bányász chippekben lévő tranzisztorok egyre kisebbek és kisebbek lettek, és ma már megközelítik a gyártható legkisebb méreteket, ami jelenleg 22 nanométer (nm). Az ASIC gyártók szeretnék a CPU gyártási technológiát alkalmazni, amely 16 nm-es vonalvastagságot használ, mert a bányászat jövedelmezősége az iparát rendkívül gyorsan hajtja előre. A bitcoin bányászatban nem várhatók további óriási ugrások, mert az iparág eljutott a leghatékonyabb eszközökig. A Moore szabály azt mondja ki, hogy az elemek sűrűsége 18 havonta megduplázódik. A hálózat bányászkapacitása ugyanakkor exponenciális ütemben tovább fog nőni, mivel az egyre kisebb méretű tranzisztorok helyett most az egyre nagyobb energiahatékonyságra tevődik át a verseny. Már nem arról szól a történet, hogy mennyit lehet bányászni egy chippel, hanem arról, hogy hány chip helyezhető el egy épületen belül úgy, hogy a hőleadás és a tápellátás megfelelő legyen.

Az extra nonce megoldás

2012 óta a bitcoin bányászat fejlődése megoldotta a blokkfej szerkezetének egy alapvető korlátját. A bitcoin korai napjaiban úgy lehetett egy blokkot kibányászni, hogy a bányász addig növelte a nonce értékét, amíg az eredményül kapott hash értéke kisebb nem lett a célnál. De közben a nehézségi szint annyira megnőtt, hogy a bányászok gyakran anélkül lépkedtek végig mind a 4 milliárd értéken, hogy megoldást találtak volna. Ezt azonban könnyű volt orvosolni azzal, hogy az eltelt időnek megfelelően módosították a blokkfejben lévő időbélyeget. Mivel az időbélyeg a blokk része, a változtatás lehetővé tette, hogy a bányász ismét végigpróbálja a nonce értékeit, és más eredményeket kapjon. Amikor azonban a bányász hardverek sebessége meghaladta a 4 GH/sec-et, egyre nehezebb volt ennek a módszernek az alkalmazása, mert a nonce értékek végigvizsgálásához 1 másodpercnél rövidebb időre volt szükség. Mikor az ASIC bányász berendezések kezdték megközelíteni, ill. túllépni az 1TH/sec sebességet, a bányász szoftverben az érvényes blokk előállításához szükséges további nonce értékekkel kellett helyet találni. Az időbélyeg módosítható egy kicsit, de ha túlságosan előreállítják, akkor a blokk érvénytelen lesz. A blokkfejben egy új „módosíthatósági” forrásra volt szükség. A megoldás az lett, hogy a coinbase tranzakció lett az extra nonce értékek forrása. Mivel a coinbase script 2 és 100 bájt közötti adat tárolásra képes, a bányászok ezt a helyet kezdték használni extra nonce helyként, ami lehetővé tette a számukra, hogy az érvényes blokkok előállításához sokkal nagyobb értéktartományt vizsgálhassanak át a blokkfejben. A coinbase tranzakció benne van a Merkle-fában, ami azt jelenti, hogy a coinbase script bármilyen megváltoztatása esetén a Merkle-gyökér is megváltozik. 8 bájt extra nonce és a „hagyományos” nonce 4 bájtja 2^{96} lehetőség átvizsgálását teszi lehetővé másodpercenként, az időbélyeg módosítása nélkül. A jövőben a bányászok először ezeket a lehetőségeket fogják kimeríteni, és csak ez után nyúlnak az időbélyeg módosításának módszeréhez. A coinbase scriptben van még további hely is, ha szükség lenne az extra nonce térr kibővítésére.

Bánytársaságok (Mining Pools)

Ebben a rendkívül versengő környezetben a magányos bányászoknak (az ún. szingli bányászoknak) esélyük sincs a jutalomra. Annak a valószínűsége, hogy találnak egy blokkot, amely majd fedezí az energia és hardver költségeiket, annyira csekély, mint ha szerencséjátékot játszanának, például olyan, mint ha lottóznának. Még a kereskedelmi forgalomban lévő leggyorsabb ASIC bányász rendszerek sem tudják felvenni a versenyt azokkal a rendszerekkel, amelyek ilyen ASIC chipek tízezreit zsúfolják be hatalmas épületekbe, és vízerőművek olcsó energiáját használják. A bányászok ezért bányatársaságokat alkotnak, közösen bányásznak, és a jutalmat a sok ezer résztvevő között osztják el. Ha egy bányász bányatársaságban bányászik, akkor a teljes jutalomnak csak egy kis részét kapja meg, de minden nap kap valami jutalmat, ami csökkenti a bizonytalanságot.

Nézzünk egy konkrét példát. Tegyük fel, hogy egy bányász vett egy bányagépet, amelynek összkapacitása 6000 GH/sec, azaz 6 TH/sec. 2014 augusztusában egy ilyen berendezés kb. 10'000 USD-be került. A hardver fogyasztása 3 kW * 24 óra, azaz 72 kWh naponta, ami kb. napi 8 USD kiadást jelent. A jelenlegi bitcoin nehézségi szintnél a bányász szingli-módban kb. 155 naponként (5 havonta) tud kibányászni egy blokkot. Ha a bányász az 5 hónap alatt talál egy blokkot, akkor a jutalma 25 bitcoin, ami 600 \$-os árfolyammal számolva kb. 15'000 USD egyszeri kifizetésnek felel meg, ami fedezí a hardver és villamos energia költségeket, és kb. 3000 USD hasznot eredményez. Az viszont, hogy talál-e a bányász 5 hónap alatt egy blokkot, a szerencsén múlik. Lehet, hogy két blokkot is talál, és nagyon nagy haszonra tesz

szert. Az is lehet, hogy 10 hónap alatt sem talál egyetlen egy blokkot sem, és veszteséges lesz. Még súlyosabb, hogy a munkabizonyíték algoritmus nehézségi szintje valószínűleg jelentősen nöni fog ezen időszak alatt, figyelembe véve a hash kapacitás jelenlegi növekedését, ami azt jelenti, hogy a bányásznak max. 6 hónapja van arra, hogy nullszaldót érjen el, mert ez alatt a hardvere gyakorlatilag elavul, és egy nagyobb kapacitású, hatékonyabb hardvert kell beszereznie. Ha azonban a bányász egy bányatársaságban vesz részt, akkor nem kell az 5 havonkénti egyszeri, 15'000 dolláros alkalomra várnia, hanem heti 500 és 750 dollár közötti összeget fog keresni. A bányatársaság kifizetései révén amortizálni tudja a hardverköltséget és ki tudja fizetni a villamos energiát anélkül, hogy hatalmas kockázatot vállalna. A hardver 6 – 9 hónap múlva még mindig elavul, és a kockázat még mindig magas, de a vizsgált időszak alatt a bevétel kiszámítható és biztos.

A bányatársaságok speciális protokollok segítségével több száz ill. több ezer bányász munkáját hangolják össze. Az egyes bányászok egy számlaszámot nyitnak a bányatársaság szerverén, majd ezt követően úgy állítják be a berendezéseiket, hogy azok ehhez a szerverhez kapcsolódjanak. A bányászok a bányászat során a hardverükkel ehhez a szerverhez kapcsolódnak, a szerver pedig összehangolja a munkájukat a többi bányászával. Ez azt jelenti, hogy a bányatársaság tagjai közösen bányásznak ki egy blokkot, és osztognak a jutalomban.

Egy blokk sikeres kibányászásakor a jutalom nem az egyes bányászokhoz, hanem a bányatársaság bitcoin címére kerül. A szerver bizonyos időközönként, ha a jutalom meghalad egy bizonyos határt, kifizeti a bányászokat. A bányatársaság általában egy pár százalékos díjat számít fel a szolgáltatásáért.

A bányatársaságban részt vevő bányászok megosztják egymás között a következő blokk utáni kutatómunkát, és „részvényeket” kapnak a munkájukért. A bányatársaságon belül beállított nehézségi szint általában 1000-szer kisebb, mint a bitcoin hálózat aktuális nehézségi szintje. Ha egy bányász talál egy blokkot, amely megfelel ennek a kisebb nehézségi szintnek, akkor kap egy részvényt. Ha valamelyik bányász sikerrel jár, és kibányászik egy blokkot, akkor a jutalom a bányatársasághoz került, és a bányászok között annak arányában kerül felosztásra, hogy ki hány részvénnyel járult hozzá a munkához.

A bányatársaságok bárki előtt nyitva állnak, legyen profi vagy amatőr, óriási vagy pirinyó hash kapacitással. A bányatársaságok egyes tagjainak csak egy kis teljesítményű kütüje van, míg mások egy teli garázs nagy teljesítményű készülékkel rendelkeznek. Némelyik bányász berendezéseinek pár kilowatt a fogyasztása, míg mások egy egész adatközponttal bányászkodnak, melynek pár megawatt a fogyasztása. Hogyan tudja a bányatársaság lemérni, hogy ki mennyivel járul hozzá a közös munkához, hogyan tudja megakadályozni, hogy a bányászok csaljanak? A választ a bitcoin munkabizonyíték rendszere jelenti, amely alacsonyabb nehézségi szintre van beállítva. Ennél az alacsonyabb nehézségi szintnél még a legkisebb hash kapacitással rendelkező bányász is elég gyakran jut részvényekhez, és emiatt mégéri neki, hogy részt vegyen a bányatársaság munkájában. Azzal, hogy a bányatársaság alacsonyabba állítja be a részvényhez jutás nehézségi szintjét, mérni tudja az egyes bányászok által elvégzett munka nagyságát. Ha valamelyik bányász talál egy olyan blokkfej hash értéket, amely kisebb a bányatársaság által megszabott nehézségi szintnél, akkor ezzel bizonyítani tudja, hogy elvégezte a munkát, ami ennek a hashnek az előállításához kellett. De ami ennél is fontosabb, a részvények keresése statisztikailag is mérhető formában hozzájárul ahhoz a munkához, amely a hálózati küszöbértéknél kisebb hashű blokk előállítására irányul. A bányászok ezrei közül, akik minél kisebb hasheket próbálnak előállítani, az egyiknek végül

sikerülni fog egy olyan hasht találni, amely elég kicsiny ahhoz, hogy kielégítse a bitcoin hálózat által felállított célt.

Térjünk vissza a kockajáték hasonlathoz. Ha a játéknak az a célja, hogy négynél kevesebbet dobunk (a teljes hálózatra vonatkozó cél), akkor a bányatársaság egy könnyebb célt ad meg, és pl. megszámolja, hogy hányszor sikerült a bányatársaságban részt vevő játékosoknak 8-nál kevesebbet dobnia. Ha egy játékos 8-nál kevesebbet dob (a bányatársaság által felállított cél), akkor a játékos kap egy részvényt, de nem nyeri meg a játékot, mivel a játék még nem érte el a célját (hogy az összeg 4-nél kevesebb legyen). A társaságban lévő játékosok a könnyebb célt gyakrabban elérik, és szabályosabb időközönként jutnak részvényekhez, még ha nem is érik el a nehezebb célt, a játék megnyerését. Néha valamelyik játékosnak sikerül olyat dobni, hogy a kockák összege négynél kevesebb legyen, vagyis megnyeri a játékot a társaságnak. Ekkor a bevétel a játékosok annak az arányában osztják el egymás között, hogy ki hány részvénnyel rendelkezik. A 8-nál kisebb cél ugyan nem volt nyerő, de jól mérte a játékban résztvevő játékosok kockadobásainak a számát.

Hasonlóképpen, a bányatársaság által beállított nehézségi szint esetén az egyes bányászok gyakran találnak olyan blokkfejeket, amelyek hash értéke kisebb ennél, és így részvényekhez jutnak. Néha egy olyan blokkfej áll elő amelynek hashe kisebb a bitcoin hálózat által megkövetelt célnál, vagyis a blokk érvényes lesz, és az egész társaság nyer.

Felügyelt bányatársaságok (Managed pools)

A legtöbb bányatársaság „felügyelt”, ami azt jelenti, hogy van egy egyén vagy szervezet, aki/amely a bányatársaság szerverét, a pool szervert futtatja. A szerver tulajdonosát *pool operátornak* hívják, és a bányászoktól a bevételből pár százalékos díjat szed.

A pool szerver speciális szoftvert és pool bányászprotokollt futtat, amely összehangolja a bányászok tevékenységét. A pool szerver egy vagy több teljes csomóponthoz is kapcsolódik, és közvetlen hozzáférése van a teljes blokklánc adatbázishoz. Ez lehetővé teszi, hogy a pool szerver a bányászok nevében ellenőrizze a tranzakciókat és a blokkokat, és mentesítse őket a teljes csomópont futtatásának terhétől. A bányászok szempontjából ez nagyon fontos szempont, mivel egy teljes csomóponthoz egy külön számítógépre van szükség, melynek legalább 15 – 20 Gbájtos tárolója (diszk) és legalább 2 Gbájt memóriája (RAM) van. Ez kívül a teljes csomópontokon futó bitcoin szoftver monitorozást igényel, karban kell tartani és gyakran kell verziót váltást végezni. Bármilyen leállás, amit a karbantartás hiánya vagy az erőforrás hiány okoz, hátrányosan érinti a bányászok profitját. Sok bányász amiatt csatlakozik egy felügyelt bányatársasághoz, mert nagy előnyt jelent, hogy egy teljes csomópont futtatása nélkül is bányászkodni tud.

A bányatársaság bányászai egy bányász protokoll segítségével kapcsolódnak a pool szerverhez, pl. Stratum (STM) vagy GetBlockTemplate (GBT) segítségével. Egy régebbi szabvány, melynek GetWork a neve, 2012 vége óta elavultnak számít, mivel 4 GH/s felett nehézkesen támogatja a bányászatot. Mind az STM, mind a GBT protokoll *blokk sablonokat* hoz létre, amelyek a létrehozandó blokk blokkfejének sablonját tartalmazzák. A pool szerver a tranzakciók összesítésével előállítja a létrehozandó blokk jelöltet, beleteszi a coinbase tranzakciót (az extra nonce hellyel), kiszámítja a Merkle-gyökeret, és hozzákapcsolja az előző blokk hashét. Az előállítandó blokk jelölt fejét ezután sablonként elküldi a pool bányászainak. Mindegyik bányász ezt a sablont használja, persze kisebb nehézségi szinttel, mint ami a

bitcoin hálózatban fennáll, és a sikeres eredményeket visszaküldi a pool szervernek, hogy részvényeket kaphasson.

P2Pool

A felügyelt poolok megteremtik a lehetőséget annak, hogy a pool operátor csaljon, hiszen az operátor a tagok által végzett munkát arra használhatja, hogy kettős költést kísérjen meg, vagy hogy érvénytelenné tegyen egy blokkot (lásd „[A konszenzus elleni támadások](#)”). Ezen kívül a központosított pool szerverek egy egy-pontos meghibásodási lehetőséget jelentenek. Ha a pool szerver leáll, vagy DoS támadás éri, akkor a pool bányászai nem tudnak tovább bányászni. 2011-ben ezen kérdések megoldására egy új bányászati módszert javasoltak és fejlesztettek ki: a P2Pool egy peer-to-peer bányász protokoll, melynek nincs operátora.

A P2Pool decentralizálja a pool szerver feladatait oly módon, hogy egy párhuzamos, blokklánc-szerű rendszert valósít meg, az ún. *részvényláncot* (sharechain-t). A részvénylánc a bitcoin blokkláncnál alacsonyabb nehézségi szinttel rendelkező lánc. A részvénylánc lehetővé teszi a bányászok decentralizált poolban történő együttműködését, mert a részvénylánon 30 másodpercenként áll elő egy részvényeknek megfelelő blokk. A részvénylánc minden egyes blokkja rögzíti a munkában résztvevő bányászok arányos jutalmait, és az előző részvényblokkból tovább viszi az addigi eredményeket. Ha valamelyik részvényblokk eléri a bitcoin hálózat nehézségi szintjét, akkor továbbításra kerül a bitcoin hálózatba és bekerül a blokkláncba. A jutalmat azok a bányászok kapják, akik létrehozták a nyerő blokk előtti részvényeket. Lényegében ahelyett, hogy egy pool szerver tartaná nyilván a pool bányászainak részvényeit és jutalmát, a részvénylánc biztosítja a bányászok számára, hogy a bitcoin blokklánc konszenzus mechanizmusához hasonló, decentralizált konszenzus mechanizmus segítségével tartsák nyilván a részvényeket.

A P2Pool bányászat bonyolultabb, mint a bányatársaságokban történő bányászat, mivel a P2Pool bányászathoz egy külön számítógépre van szükség, amelyen elégges a diszk hely, a memória, és az internet sávszélesség ahhoz, hogy támogatni tudjon egy teljes bitcoin csomópontot valamint a P2Pool csomópont szoftverét. A P2Pool bányászok a bányagépeiket a helyi P2Pool csomópontjukkal kötik össze, ami egy pool szerver funkcióit szimulálja, vagyis blokk sablonokat küld a bányász hardvernek. Egy P2Poolban mindegyik bányász maga állítja össze a blokk jelöltjét, és a tranzakciókból épp úgy képezi a blokkot, mint a szingli bányászok, de ezt követően a részvénylánon közösen bányászik a többi bányással. A P2Pool egy hibrid módszer, amelynek előnye, hogy tagjai gyakrabban jutnak jövedelemhez, mint a szingli bányászok, de mindez anélkül valósul meg, hogy egy pool operátor túl sok felügyeleti joghoz jutna.

Mostanában a P2Poolokban jelentősen nőtt a részvétel, mivel a bányatársaságok koncentrációja megközelítette azt a szintet, ami már felveti az 51%-os támadással kapcsolatos aggodalmakat (lásd „[A konszenzus elleni támadások](#)”). A P2Pool protokoll továbbfejlesztése várhatóan szükségtelenné teszi majd a teljes csomópont futtatását, ami még könnyebbé teszi majd a decentralizált bányászatot.

Noha a P2Pool csökkenti a bányatársaságok operátorainak a hatalmát, maga a részvénylánc viszont sebezhető az 51%-os támadásokkal szemben. A P2Pool sokkal szélesebb elterjedése sem oldja meg a bitcoin 51%-os támadásokkal kapcsolatos problémáját. A P2Pool inkább csak ellenállóbbá teszi a teljes rendszert, mert diverzifikálja a bányászati ökoszisztemát.

A konszenzus elleni támadások

A bitcoin konszenzus mechanizmusa elméletben támadható, ha a bányászok (vagy poolok) a hash kapacitásaikat tisztességtelen vagy ártalmas célra használják. Mint láttuk, a konszenzus mechanizmus azon alapul, hogy a bányászok többsége becsületesen, önzetlenül viselkedik. De ha egy bányásznak vagy egy bányász csoportnak sikerül megszereznie a bányász kapacitás jelentős részét, akkor a konszenzus mechanizmus ellen indított támadással szétzilálhatja a bitcoin hálózat biztonságát és rendelkezésre állását.

Fontos megjegyeznünk, hogy konszenzus elleni támadásokkal csak a jövőbeli, vagy legfeljebb a közelműlt eseményei befolyásolhatók (néhányszor tíz blokk). A bitcoin főkönyve az idő műlásával egyre ellenállóbb. Bizonyos „mélységen” túl a blokkok teljesen immúnisak, még egy olyan tartós, konszenzus elleni támadás esetén is, amely elágazást hoz létre. A konszenzus elleni támadások nem érintik a titkos kulcsok és az aláíró algoritmus (ECDSA) biztonságát. Egy konszenzus elleni támadással nem lehet bitcoinokat lopni, vagy aláírás nélkül bitcoinokat elkölni, átirányítani őket vagy más módon megváltoztatni a régebbi tranzakciókat vagy bejegyzéseket. A konszenzus elleni támadások csak a legutóbbi blokkokat érintik, és megakadályozhatják további blokkok előállítását, vagyis DoS (denial-of-service, szolgáltatás megtagadási) támadást valósíthatnak meg.

A konszenzus mechanizmus elleni egyik ilyen támadás forgatókönyve az ún. „51 %-os támadás”. Ennél a forgatókönyvnél a hálózati hash kapacitás többségével (51%-val) rendelkező bányászok összejátszanak egymással. Mivel a támadók képesek arra, hogy a blokkok többségét ők bányásszák ki, ezért tudatosan „elágazásokat” képesek létrehozni a blokkláncban, és kétszer is el tudják költeni ugyanazt a bitcoint, vagy DoS támadást képesek indítani bizonyos tranzakciók vagy címek ellen. Egy elágazás/kétszeres költés úgy valósul meg, hogy a támadó az előzőleg már megerősített blokkokat érvénytelenné teszi, mégpedig úgy, hogy a blokkok előtt egy elágazást hoz létre, és egy alternatív láncra történő átkonvergálást kényszerít ki. Megfelelő hash kapacitással a támadó egymás után akár 6 vagy még több blokkot érvénytelenné tud tenni, ezáltal azok a tranzakciók, amelyeket már változatlannak tekintettek (6 megerősítés), érvénytelenné válnak. Megjegyzem, hogy a kettős költés csak a támadó saját tranzakciójánál valósítható meg, mert a támadó ezeket tudja szabályosan aláírni. A saját tranzakciók újból elköltése akkor rentabilis, ha a támadó így egy visszafordíthatatlan kifizetéshez vagy áruhoz jut, anélkül, hogy fizetne érte.

Nézzünk az 51 %-os támadásra egy gyakorlati példát. Az első fejezetben láttunk egy Alíz és Bob közötti tranzakciót, amellyel Alíz vett egy csésze kávét. Bob, a kávéház tulajdonosa szívesen elfogadta a kávé árát anélkül is, hogy megvárta volna a tranzakció megerősítését (egy blokk kibányászását), mert a kettős költés veszélye egy csésze kávé esetén viszonylag csekély, és fontosabb a gyors kiszolgálás. Ez hasonló ahhoz a gyakorlathoz, ahogy a legtöbb kávéházban 25 \$ alatti összegeknél a hitelkártyás fizetési módot aláírás nélkül is elfogadják, mivel az utólagos visszaszámlázás veszélye kicsi, míg az aláírás okozta késedelem költsége viszonylag nagy. Ezzel szemben ha egy költségesebb tételel adnak el bitcoinért, akkor a kettős költés veszélye nagyobb, mivel a vevő szétküldhet egy másik tranzakciót, amely ugyanazokat a bemeneteket (UTXO) költi el, és a kereskedő hoppon marad. Kettős költés kétféleképpen valósulhat meg: vagy úgy, hogy a tranzakció még nincs megerősítve, vagy úgy, hogy a támadó egy blokklánc elágazás segítségével számos blokkot „visszacsinál”. Egy 51 %-os támadás esetén a támadó egy új láncban másodszor is képes elkölni a saját tranzakciójában lévő bitcoinokat, és így érvényteleníteni tudja a régi lánc ezeknek megfelelő tranzakciójait.

Példákban a rosszindulatú támadó, Mallory elmegy Carol galériájába, és vásárol egy gyönyörű szárnyas oltárt, amely Satoshi Nakomatót Prométeuszként ábrázolja. Carol „A nagy tűz” című képet, mely 250'000\$-ba kerül, bitcoinért adja el Malllory-nak. Carol ahelyett, hogy várna hat vagy hét megerősítésre, becsomagolja a képet, és egy megerősítés után átadja Mallory-nak. Mallory bűntársa, Paul egy nagy bányatársaságot üzemeltet. Amint Mallory tranzakciója bekerül a blokkláncba, Paul azonnal indít egy 51%-os támadást. Paul úgy befolyásolja a bányatársaság munkáját, hogy ugyanazon a blokkmagasságon, amelyen a Mallory tranzakcióját tartalmazó blokk van, újrabányásszanak egy blokkot. A régi blokkot az új blokk fogja helyettesíteni, és Mallory Carolnak történő fizetsége helyett egy olyan tranzakciót fog tartalmazni, amely ugyanazokat a bemeneteket költi el, de ezúttal Mallory számára. A másodszori költéshez tartozó tranzakció ugyanazokat az UTXO-kat fogyasztja el, és Mallory pénztárcájába fizeti vissza őket, vagyis Mallory ahelyett hogy fizetne, lényegében megtartja a bitcoinokat. Paul ezután egy újabb blokkot állítat elő a bányatársasággal, hogy a kettős költést tartalmazó lánc hosszabb legyen, mint az eredeti lánc (vagyis egy elágazást okoz a Mallory tranzakcióját tartalmazó blokk alatt). Ha a blokklánc elágazás az új (hosszabb) lánc javára dől el, akkor a második tranzakció fog az eredeti, Carolnak történő fizetség helyébe lépni. Carol ott áll fizetség és szárnyas oltár nélkül. Azok a bányászok, akik Paul bányatársaságában dolgoznak, egyáltalán nem tudnak a kettős költési kísérletről, mivel a bányászat automatizált, és nem tudnak minden tranzakciót vagy blokkot nyomon követni.

Ha egy kereskedő védekezni szeretne az efféle támadások ellen, akkor nagy értékű tételek esetén várnia kell legalább hat megerősítésre, mielőtt a terméket odaadná a vevőnek. Egy másik lehetőség, ha az eladó egy több aláírást megkívánó (multisig) letéti számlát használ, és a pénz beérkezése után ismét csak jó néhány megerősítést megvár. Minél több a megerősítés, annál nehezebb a tranzakciót egy 51 %-os támadással érvénytelennek tenni. A nagy értékű tételeknél bitcoinnal fizetni még így is kényelmes és hatékony, még akkor is, ha a vevőnek 24 órát kell várnia a szállításra, ami 144 megerősítést jelent.

A kettős költésen kívül van egy másik forgatókönyv, amelynél a konszenzus elleni támadással bizonyos bitcoin szereplők (adott bitcoin címek) működése hiúsítható meg. Az a támadó, aki rendelkezik a többségi bányász kapacitással, egyszerűen figyelmen kívül hagy bizonyos tranzakciókat. Ha ezek a tranzakciók bekerültek egy másik bányász által előállított blokkba, akkor a támadó szándékosan elágazást hoz létre, és újrabányássza a blokkot, ezáltal ismét meghiúsítja a tranzakciót. Ezzel a támadással tartós szolgáltatás megtagadás (DoS) érhető el egy adott címmel vagy címekkel szemben, ha a támadó rendelkezik a bányász kapacitás többségével.

A neve ellenére az 51 %-os támadáshoz nem szükséges, hogy a támadó ténylegesen rendelkezzen a hash kapacitás 51%-ával. Egy ilyen támadás már kisebb kapacitásnál is megkísérelhető. Az 51%-os határ egyszerűen csak azt a szintet jelzi, amelynél az ilyen támadás majdnem minden sikeres lesz. A konszenzus elleni támadás lényegében a következő blokkért folytatott kötélhúzás, melyben valószínűleg az „erősebb” csoport győz. Kevesebb hash kapacitásnál a sikeres valószínűsége is kisebb, mivel a többi bányász a blokkok előállítását „becsületes” módon végzi. Úgy is lehet nézni a dolgot, hogy minél több hash kapacitása van a támadónak, annál hosszabb elágazást tud szándékosan létrehozni, vagyis annál több, a közelmúltban létrejött blokkot tud érvényteleníteni, vagy annál több jövőbeli blokkot tud befolyásolni. Biztonsági kutatók statisztikai modellezéssel megállapították, hogy különféle konszenzus elleni támadások már akár 30 %-os hash kapacitás esetén is lehetségesek.

A teljes hash kapacitás nagy mértékű növekedése a bitcoint vitathatatlanul támadhatatlanná tette az egyes bányászokkal szemben. Egy szingli bányász még a teljes hálózati kapacitás 1 %-a felett sem rendelkezik. Ugyanakkor a bányatársaságok miatt megjelenő központosítás magával hozta annak a veszélyét, hogy a pool operátora haszonszerzési céllal támadást indít. A felügyelt poolok operátora képes az előállítandó blokkok befolyásolására, és képes azt is eldönteni, hogy mely tranzakciók kerüljenek be a blokkba. Ha a pool operátora korlátozott mértékben és ügyesen él vissza ezzel az erővel, akkor anélkül profitálhat egy konszenzus elleni támadásból, hogy lebukna.

Vannak olyan támadók is, akiket nem a haszonszerzés vágya hajt. Az egyik ilyen lehetséges forgatókönyvben a támadó célja a bitcoin hálózat szétzilálása, anélkül, hogy profitálna ebből a bomlasztásból. A bitcoin hálózat megbénításához hihetetlen nagy befektetésre és titkos tervekre van szükség, de egy nagy költségvetésű, vélhetően valamelyik kormány által finanszírozott támadó még erre is képes lehet. Az is elképzelhető, hogy egy pénzes támadó egyszerre többféle módon ássa alá a konszenzus mechanizmust: bányász hardvert halmoz fel, megvesztegeti a pool operátorokat, és DoS támadást indít a többi pool ellen. Elméletileg mindegyik forgatókönyv lehetséges, de a bitcoin hash kapacitásának exponenciális növekedése mellett ez az út egyre kevésbé járható.

Kétségtelen tény, hogy egy konszenzus elleni komoly támadás megingatná a bitcoinba vetett rövid távú bizalmat, és jelentős árcsökkenéssel járna. A bitcoin hálózat és szoftver viszont állandóan fejlődik, emiatt a konszenzus elleni támadásokra a bitcoin közösség azonnal megfelelő ellenintézkedéseket tud tenni, ami a bitcoint még ellenállóbbá, egészségebbé és robusztusabbá teszi.

9. Alternatív blokkláncok, fizetőeszközök, alkalmazások

A bitcoin létrejöttét közel 20 év kutatómunka előzte meg az elosztott rendszerek és fizetőeszközök világában. A bitcoinnal egy forradalmian új technológia jött létre: a munkabizonyítékon (Proof-of-Work) alapuló elosztott közmegegyezés. A bitcoinnak ez a központi jelentőségű vívmánya egy sereg egyéb újításnak is teret nyitott a fizetőeszközök, a pénzügyi szolgáltatások, a közigazdaság, az elosztott rendszerek, a választási rendszerek, a kormányzás, és a különféle szerződések terén.

Ebben a fejezetben azokkal az innovatív eljárásokkal (pl. alternatív lánkok, fizetőeszközök, alkalmazások) foglalkozunk, amelyek a bitcoin, illetve a blokklánc 2009-es bevezetése óta születtek. Meg fogjuk vizsgálni az alternatív érméket, azaz az *alt-coinokat*, melyek ugyanolyan alapokon nyugvó digitális pénzek, mint a bitcoin, de teljesen különböző blokkláncot és p2p hálózatot használnak.

A fejezetben megemlített alt-coinoknál legalább 50-szer több marad majd említés nélkül (kiváltva ezek fejlesztőinek és felhasználóinak haragját). Célunk nem az alt-coinok értékelése vagy minősítése, sőt még csak nem is a „legjelentősebb” fejlesztések a valamelyen szubjektív értékítélen alapuló kiemelése, hanem az, hogy szemléltessük az ökoszisztema változatosságát és gazdagságát, kiemelve azon fejlesztéseket, amelyek valamilyen újdonságot, vagy jelentős eltérést hoztak az addigiakhoz képest. Az alt-coinok néhány igazán érdekes példája pénzügyi szemszögből tulajdonképpen abszolút bukásnak minősíthető. Sokszor mégis pont ettől válnak igazán érdekessé, tanulmányozásra méltóvá. Ugyanakkor hangsúlyozni kell, hogy az itt leírtak semmiféleképpen sem tekinthetők befektetői kisokosnak.

Szinte naponta jelennek meg új fejlesztések, így szinte biztos, hogy néhány fontos alt-coin kimarad a felsorolásunkból, és az sincs kizárvá, hogy pont az, amelyik megváltoztatja majd a történelmet. Az innovációk gyorsasága teszi vibrálóvá és izgalmasá ezt a szférát, és egyben garantálja, hogy ez a fejezet már a nyomtatásba kerülések kor idejétmúlt lesz.

Az alternatív fizetőeszközök és blokkláncok osztályozása

A bitcoin egy nyílt forráskódú projekt, amelynek kódja számos egyéb szoftver alapjául szolgált. A bitcoin forráskódjából származó leggyakoribb szoftverek az alternatív, decentralizált fizetőeszközök, vagyis az ú.n. *alt-coinok*, melyek a bitcoinnal megegyező építőelemeket használnak a különböző digitális fizetőeszközök megvalósításához.

A bitcoin blokkláncára egy egész sereg protokollt épül. Ezek a *meta-coinok*, *meta-chaineik*, vagy *blokklánc alkalmazások*, melyek a blokkláncot egyfajta alkalmazási platformként használják, vagy a bitcoin protokollt további protokoll rétegekkel bővízik. Ilyen például a Colored Coins, a Mastercoin, az NXT és a Counterparty.

Az alábbiakban néhány jelentősebb alt-coin fogunk megvizsgálni, többek között a Litecoint, a Dogecoint, a Freicoint, a Primecoint, a Peercoint, a Darkcoint, és a Zerocoint. Felhozott példáinkat történelmi okokból, vagy egy-egy specifikus újítás miatt tartjuk jelentősnek, semmiképp sem azért, mert ezek az általunk „legjobbnak”, vagy legértékesebbnek vélt alt-coinok.

Az alt-coinokon kívül számos olyan alternatív blokklánc implementáció is van, amelyek nem igazán felelnek meg a „coin” kategóriának, így ezeket a fejlesztéseket *alt-chain*-eknek hívom a továbbiakban. Ezek az alt-chainek szerződéseket, név regisztrációkat, és egyéb alkalmazásokat valósítanak meg valamilyen közegyeztetést teremtő algoritmussal és egy osztott blokkláncnal. Az alt-chainek ugyanazokat az építő elemeket használják, mint a bitcoin, és néha az is előfordul, hogy fizetőeszközöként, vagy egyfajta fizetési rendszerként használják őket, de az elsődleges céluk nem az, hogy fizetőeszközöként funkcionáljanak. Az alt-chainek közül a Namecoin, az Ethereum, és az NXT-t fogjuk részletesebben megvizsgálni.

Végül meg kell említeni azokat a bitcoin versenytársakat is, amelyek felkínálják ugyan a digitális fizetőeszköz, vagy digitális fizetési hálózat lehetőségét, de nem használnak semmilyen decentralizált főkönyvet vagy munkabizonyítékon (Proof-of-Work) alapuló közmegegyezési mechanizmust. Ide tartozik például a Ripple, illetve társai. Ezek a nem-blokklánc alapú technológiák kívül esnek e könyv keretein, így jelen fejezetben sem foglalkozunk velük.

Meta-coin platformok

A meta-coinok és meta-chainek olyan szoftver rétegek, amelyek a bitcoinra lettek ráépítve, azaz esetükben vagy egyfajta „fizetőeszköz a fizetőeszközben” implementációról, vagy a bitcoin rendszerén belüli platformról/protokollról van szó. Ezek a funkcionális rétegek további tulajdonságokkal és képességekkel bővítik a bitcoin protokollt, oly módon, hogy a bitcoin tranzakciókba és címekbe egyéb kiegészítő adatokat kódolnak. Az első ilyen típusú meta-coin implementációk különféle „hack”-eket használtak a bitcoin blokklánc meta-adatokkal történő bővítésére, így például bitcoin címekbe kódoltak adatokat, vagy felhasználatlan tranzakciós mezőket (például a tranzakció szekvencia mezőjét) hasznosítottak a járulékos protokollal kapcsolatos meta-adatok kódolására. Mióta bevezetésre került az OP_RETURN műveleti kód, a meta-coinok közvetlenebb módon tudnak a blokkláncba meta-adatokat kódolni, és a legtöbb meta-coin ennek a használatára állt át.

Színezett érmék (Colored Coins)

Az úgynevezett *Színezett Érmék* (Colored Coins) egy olyan meta-protokoll, amely a bitcoin egy kis mennyiségehez rendel hozzá további információkat. Ezeknek a színezett érméknek az a célja, hogy a megjelölt bitcoint egyéb értékkel is felruházzák. Például képzeljünk el egy 1 dolláros bankjegyet, amelyre rápecsételjük azt a feliratot, hogy „Ez az Hiper-Szuper Kft. 1 egységnyi részvénnyei elismervénye”. Ekképp az 1 dolláros bankjegy két funkciót is betölt: bankjegy és egyben részvénnyei elismervény. Mivel elismervényi aspektusa értékesebb fizetőeszközi minőségénél, így valószínűsíthető, hogy nem fogunk cukorkát vásárolni vele. Fizetőeszközi minősége tehát már nem tűnik hasznosnak. A Színezett Érmék hasonló elven működnek, azaz a bitcoin egy adott és parányi részét egy másik minőséggel is felruházzák. A terminusban a „color” (szín) kifejezés arra utal, hogy egy adott egységet egy speciális jelentéssel, attribútummal látunk el. Az attribútum jelzi a specifikus minőség jelenlétéit az így megjelölt egységen. A „színmegjelölést”, mint attribútumot természetesen metaforikusan értjük, azaz a Colored Coinok nem tartalmaznak színeket.

A Colored Coinok kezelésére speciális „pénztárca” programok szolgálnak, amelyek lajstromba veszik és értelmezik azokat a meta-adatokat, amelyeket a „colored” bitcoinokhoz csatoltak. Egy ilyen a pénztárca használatával a felhasználó egy adott mennyiségi „színtelen” (attribútum nélküli) bitcoint egy speciális jelentésű címke hozzáadásával színezett érmévé

konvertálhat. Ez a címke jelenthet részvényses elismervényt, kuponokat, ingatlan tulajdont, árucikket, beváltható pénzt, és még sok minden mást. Csak a színezett érme felhasználónin műlik, hogy milyen specifikus minőséggel ruházzák fel az adott „színnel” (attribútummal) ellátott coinokat. Ahhoz, hogy egy sima érme színezett érmévé váljon, arra van szükség, hogy a felhasználó meghatározza a hozzá társított meta-adatokat, például kibocsájtás típusát, osztható, vagy osztathatatlan voltát, szimbólumát, leírását és egyéb adatokat. Miután egy sima coin színezett érmévé vált, értékesíthetővé, oszthatóvá, illetve egyesíthetővé válik, és akár jutalékok kifizetésére is alkalmas. A színezett érmék visszaállíthatóak sima coinokká is, úgy, hogy megfosztjuk őket speciális minőségüktől, és visszaállítjuk „sima” bitcoin névértéküket.

Mindennek szemléltetéseképpen létrehoztunk 20 db „MasterBTC” attribútummal ellátott színezett érmét, kupont, amelyek a jelen könyv egy-egy ingyenes példányának felelnek meg, amint azt a 9-1. példa mutatja. Ezt követően a MasterBTC jelzéssel ellátott összes egység (vagyis az ilyen attribútummal ellátott colored coinok) értékesíthetőek vagy továbbadhatóak minden olyan bitcoin felhasználónak, aki rendelkezik a kezeléséhez szükséges pénztárca programmal. Egy adott felhasználó a megszerzett colored coint más felhasználónak továbbutalhatja, vagy a kibocsájtónál beválthatja a könyv egy ingyenes példányára. Az ismertetett példa [itt](#) érhető el.

9-1. példa: A jelen könyv egy ingyenes példányához kupont biztosító colored coin meta-adat profilia

```
{
  "source_addresses": [
    "3NpZmvSPLmN2cVFw1pY7gxEAVPCVfnWfVD"
  ],
  "contract_url":
"https://www.coinprism.info/asset/3NpZmvSPLmN2cVFw1pY7gxEAVPCVfnWfVD",
  "name_short": "MasterBTC",
  "name": "Free copy of \"Mastering Bitcoin\"",
  "issuer": "Andreas M. Antonopoulos",
  "description": "This token is redeemable for a free copy of the book
\"Mastering Bitcoin\"",
  "description_mime": "text/x-markdown; charset=UTF-8",
  "type": "Other",
  "divisibility": 0,
  "link_to_website": false,
  "icon_url": null,
  "image_url": null,
  "version": "1.0"
}
```

Mastercoin

A Mastercoin egy olyan bitcoinra illesztett protokoll, amely a bitcoin rendszerét kiterjesztve, különféle alkalmazásokhoz biztosít platformot. A Mastercoin elsősorban nem fizetési eszköz, jóllehet a Mastercoin tranzakciókhoz az MST megjelölésű fizetőeszközt használja. A Mastercoind legjobban talán úgy lehetne meghatározni, mint fizetőeszközök, vagyontárgyak, decentralizált eszközök cseréjét, vagy például szerződések és persze még számos más dolog létrehozását támogató platformot. A Mastercoin egy olyasféle alkalmazási protokoll, amely a bitcoin pénzügyi tranzakciós transzport-rétegéhez van hozzáillesztve, hasonlóképpen a TCP-n futó HTTP-hez.

A Mastercoin elsősorban azokon a tranzakciókon keresztül operál, amelyek egy speciális bitcoin címről, vagy címre érkeznek (ezt „exodus” címmek hívjuk,

`1EXoDusjGwvnjZUyKkxZ4UHeF77z6A5S4P`), hasonlóképpen ahhoz, ahogy a HTTP is egy speciális portot (80 port) használ arra, hogy megkülönböztesse az általa szállított forgalmat a TCP egyéb forgalmától. A Mastercoin protokoll a tranzakciók meta-adatait a speciális exodus címmel és többszörös aláírások használatával kódolja. A Mastercoin a jövőben fokozatosan az OP_RETURN bitcoin operátor használatára tér át.

Counterparty

A Counterparty szintén egy bitcoinra illesztett protokoll, amely fizetőeszközök, átruházható zsetonok, pénzügyi eszközök, és decentralizált vagyontárgyak cseréinek, illetve egyéb speciális alkalmazásoknak a gyűjtőtára. A Counterparty implementációja elsősorban a bitcoin script nyelv OP_RETURN operátorát használja a meta-adatok rögzítésére. Ezek járulékos jelentéssel bővítik ki a bitcoin tranzakciókat. A Counterparty az XCP jelzést használja a Counterparty tranzakciók során.

Alt-coinok

Az alt-coinok túlnyomó többségének az alapját a bitcoin forráskódja jelentette. Ezek az ú.n. „elágazások” („forks”). Némelyikük a „semmiből” lett implementálva, azaz a blokklánc modellre támaszkodik, de anélkül, hogy bármit is felhasználva a bitcoin forráskódjából. Az alt-coinok és az alt-chainek (melyekre a következő részben térünk ki részletesebben) a blokklánc technológia teljesen különálló implementációi, tehát mindegyik változat a saját blokkláncát használja. A terminusok (alt-coin, alt-chain) közötti különbség arra utal, hogy míg az alt-coinok elsősorban fizetőeszközök funkcionálnak, addig az alt-chain-eket általában más céllal használják.

Szigorúan véve, a bitcoin kód első jelentős alt-coinja igazából nem is alt-coin volt, hanem egy Namecoin nevű alt-chain, de erről a következő részben fogunk részletesebben írni.

A bejelentés időpontja alapján az első alt-coinnak az *IXCoin* minősíthető, amelyet 2011 augusztusában indítottak útjára. Az IXCoinban csak néhány bitcoin paramétert változtattak meg, például felgyorsították a fizetőeszköz „előállítását”, 96 érmére növelte a blokkokért járó jutalmat.

A *Tenebrix* 2011 szeptemberében indult útjára. A Tenebrix volt az első olyan digitális pénz, amely egy alternatív munkabizonyíték (Proof-of-Work) rendszeren alapuló algoritmust, az úgynevezett scrypt-et implementálta. A scrypt algoritmus eredetileg jelszavak megerősítésére szolgált (hogy ellenálljanak a nyers erővel (brute-force) történő töréseknek). A hivatalos közlemény szerint a Tenebrix kifejlesztésénél az volt a vezérelv, hogy egy olyan coint hozzanak létre, amely ellenálló mind a GPU, mind az ASIC típusú bányászattal szemben, mert memória-igényes algoritmust használ. A Tenebrix, mint fizetőeszköz ugyan nem könyvelt el különösebb sikereket, de megalapozta a Litecoind, amely óriási népszerűségre tett szert, egyben klónok százai előtt nyitotta meg az utat.

A *Litecoin* azon kívül, hogy a scrypt-et használja munkabizonyíték (Proof-of-Work) algoritmusnak, egy gyorsabb blokk-generálási időt valósított meg, és 2,5 percre csökkentette a bitcoin 10 perces intervallumát. A Litecoind úgy is szokták emlegetni, hogy ez az „ezüst, ha a bitcoin az arany”, és nem vitás, hogy alternatív fizetőeszközöként a bitcoin egy könnyűsűlyű versenytársa. A gyorsabb megerősítési időnek, illetve a maximum 84 millió érme

összmennyiségnek köszönhetően, számos Litecoin hívő gondolja úgy, hogy az „ezüst érme” a kereskedelmi tranzakciók területén a bitcoinnál jobb megoldást jelent.

A bitcoinra vagy Litecoin-ra támaszkodó különféle alt-coin sarjak száma 2011 és 2012 folyamán tovább nőtt, és 2013 elején már közel 20 alt-coin versenyzett a piaci pozíószerzésért. 2013 végére a különféle alt-coinok száma megtízszerződött, elérve a 200-as határt, így 2013 vitathatatlanul az „alt-coinok éve” volt. Természetesen a növekedés folytatódott, 2014-ben már közel 500 alt-coin létezéséről tudunk. Napjainkban az alt-coinok több mint fele Litecoin klón.

Egy alt-coin létrehozása viszonylag egyszerű feladat, ezért is van már belőlük 500-nál is több. Legtöbbük csak nagyon kis eltérést mutat a bitcoinhoz képest, és igazából semmi említésre méltót sem tartalmaz. Sok csak arra a célra szolgál, hogy a fejlesztőjét gazdagabbá tegye. A „pump-and-dump” sémák (manipulatív módon felvinni az alt-coin árát, majd hirtelen elárásztani vele a piacot) és az utánzók között azért természetesen van néhány figyelemre méltó kivétel, és igazán fontos innováció is. Ezek az alt-coinok vagy egy egészen eltérő megközelítést alkalmaznak, vagy a bitcoin tervezési mintáját jelentős újítással bővítik. Alapvetően három olyan területet tudunk elkülöníteni, amelyekben az említett alt-coinok változást hoztak a bitcoinhoz képest:

- Eltérő pénzügyi irányelvek
- Eltérő munkabizonyíték (Proof-of-Work) rendszer, vagy eltérő konszenzus mechanizmus
- Különleges jellemzők, például fokozottabb anonimitás

Az alt-coinok és alt-chain-ek grafikus idővonala az alábbi linken tekinthető meg:
<http://mapofcoins.com>.

Egy alt-coin kiértékelése

Hogyan lehet eldönteni, hogy a rengeteg alt-coin közül melyek a valójában figyelemre méltó fejlesztések? Vannak alt-coinok, amelyek arra tesznek kísérletet, hogy széles körben, fizetőeszközöként kerüljenek felhasználásra. Mások inkább laboratóriumi próbálkozások, melyek különböző jellegzetességekkel, vagy pénzügyi modellekkel kísérleteznek, számos fejlesztés pedig kizárolag a fejlesztőik gyors meggazdagodásának a céljából született. Az alt-coinok kiértékeléséhez azok sajátos jellegzetességeit és piaci paramétereit vettem figyelembe.

Íme pár kérdés, amit érdemes felenni egy alt-coinnal kapcsolatban, amikor azt a bitcoinnal összevetve vizsgáljuk:

- Hozott-e valamilyen jelentős innovációt az adott alt-coin?
- Elégé különbözik-e az adott alt-coin a bitcointól ahhoz, hogy átcسáítsa a bitcoin felhasználókat?
- Az adott alt-coin kapcsolódik-e egy érdekes piaci szektorhoz vagy alkalmazáshoz?
- Képes-e elegendő bányász figyelmét felkelteni ahhoz, hogy védve legyen a támadásokkal szemben?

Néhány megvizsgálandó piaci és pénzügyi tényező:

- Mekkora az adott alt-coin teljes piaci kapitalizációja?
- Mennyi alt-coin becsült felhasználónak/pénztárcáinak a száma?
- Mennyi az alt-coin elfogadó kereskedők száma?
- Naponta hány darab alt-coin tranzakció megy végbe?
- Mekkora a napi tranzakciók értéke?

Könyvünk jelen fejezetében elsősorban az alt-coinok technikai jellegzetességeire és potenciális innovációira koncentrálunk, tehát az imént felsorolt kérdések első felére.

Pénzügyi paraméterű alternatívák: Litecoin, Dogecoin, Freicoin

Pénzügyi paraméterei alapján a bitcoin egy deflációs, fix kibocsátású pénz. Mennyisége 21 millió egységre (vagy 2100 billió kisebb egységre) korlátozódik, kibocsátásának sebessége szabályosan (mértani sorozat szerint) csökkenő, és „szívverésének” üteme - a tranzakciók visszaigazolásának és a blokkok legenerálásának az ideje - 10 perc. Számos alt-coin változtatta meg ezt a három paramétert abból a célból, hogy egy eltérő monetáris keretrendszert hozzon létre. A több száz ilyen típusú alt-coin közül a leginkább figyelemre méltó fejlesztések a következők:

Litecoin

A Litecoin, amely az egyik legelső alt-coin volt 2011-ben, a bitcoin utáni a második legsikeresebb digitális fizetőeszköz. Elsődleges innovációi a scrypt, mint Proof-of-Work algoritmus alkalmazása (Tenebrix-től örökölt újítás), illetve a gyorsabb/könnyebb fizetőeszköz paraméterek.

- Blokk-generálási idő: 2,5 perc
- Maximális érme mennyiség: 84 millió egység, 2140-ig.
- Az egyezményes algoritmus: scrypt Proof-of-Work
- Piaci kapitalizáció: 160 millió amerikai dollár 2014 közepén

Dogecoin

A Dogecoin 2013 decemberében indult útjára, mint a Litecoin egyik „elágazása”. A Dogecoin azért érdekes, mert monetáris politikája gyors kibocsátást ír elő, továbbá a fizetőeszköz maximális érmemennyisége elég magas ahhoz, hogy felhasználót fizetésre, illetve költésre motiválja. Abból a szempontból is figyelemre méltó, hogy eredetileg csak viccnek szánták, azonban viszonylag gyorsan nagy népszerűségre tett szert, és egy relatív nagy és aktív felhasználói közösséggel is rendelkezett, 2014-es gyors hanyatlását megelőzően.

- Blokk-generálási idő: 60 másodperc
- Maximális érme mennyiség: 100'000'000'000 (100 milliárd) Doge, 2015-ig.
- Az egyezményes algoritmus: scrypt Proof-of-Work
- Piaci kapitalizáció: 12 millió amerikai dollár 2014 közepén

Freicoин

A Freicoин 2012 júliusában kezdte meg pályafutását. Ez egy *inflálódó fizetőeszköz*, ami azt jelenti, hogy az érték tárolásához negatív kamatláb tartozik. A Freicoинban tárolt érték után évente 4,5 % díj kerül felszámításra, ami serkenti a fogyasztást, és visszatartja a pénz felhalmozását. A Freicoин azért érdekes, mert egy olyan monetáris politikát implementál, amely pontosan a bitcoin deflációs politikájának az ellentéte. A Freicoин mint fizetőeszköz nem igazán lett sikeres, jelentősége inkább abban van, hogy jól szemlélteti az alt-coinok képviselte monetáris politikák változatosságát.

- Blokk-generálási idő: 10 perc
- Maximális érme mennyisége: 100 millió érme, 2140-ig.
- Konszenzus algoritmus: SHA256 Proof-of-Work
- Piaci kapitalizáció: 130'000 amerikai dollár 2014 közepén

A konszenzusrendszer újításai: Peercoin, Myriad, Blackcoin, Vericoин, NXT

A bitcoin konszenzus mechanizmusa munkabizonyíték (Proof-of-Work) alapú, és az SHA256 algoritmust használja. Az első alt-coinok a scrypt-et használták alternatív Proof-of-Work algoritmusként, ekképp az egyébként ASIC centralizációra hajlamos bányászat CPU-barát folyamattá vált. A scrypt alkalmazása után a konszenzus mechanizmus lett az áttörő innovációk fő területe. Alt-coinok garmadánál figyelhető meg a változatos algoritmus implementáció, például a scrypt, a scrypt-N, a Skein, a Groestl, az SHA3, az X11, a Blake, és mások adoptációja. Néhány alt-coin egyszerre több algoritmust is alkalmaz a Proof-of-Work rendszerhez. 2013 folyamán a Proof-of-Work rendszernek egy olyan alternatívával is találkoztunk, amelyet *kockázati bizonyítéknak* (Proof-of-Stake) hívnak, és amely számos modern alt-coin alapját képzi.

A kockázati bizonyíték (Proof-of-Stake) egy olyan rendszer, amelynél a fizetőeszköz tulajdonosai leköthetik fizetőeszközüköt, mint egyfajta kamatozó biztosítékot. A letéti bizonylathoz (Certificate of Deposit azaz CD) hasonlóan a résztvevők leköthetik megtakarításaiuk egy részét, ezáltal nyereségre tehetnek szert új fizetőeszközök formájában (kamatkifizetésként) és tranzakciós díjak formájában.

Peercoin

A Peercoин 2012 augusztusában indították útjára. Ez volt az első olyan alt-coin, amely az új fizetőeszköz kibocsátásához a munkabizonyíték (Proof-of-Work) és a kockázati bizonyíték (Proof-of-Stake) algoritmusának a keverékét használta.

- Blokk-generálási idő: 10 perc
- Konszenzus algoritmus: (Hibrid) Proof-of-Stake, kezdetben pedig Proof-of-Work rendszer
- Piaci kapitalizáció: 14 millió amerikai dollár 2014 közepén

Myriad

A Myriad 2014 februárjában jelent meg, és azért említésre méltó, mert, öt különböző Proof-of-Work algoritmust alkalmaz (SHA256d, Scrypt, Qubit, Skein és Myriad-Groestl) egyszerre, és a nehézségi szint a bányászati részvétel függvényében minden algoritmusnál más és más

lehet. A Myriad mögött meghúzódó szándék az, hogy az alt-coint éppúgy immúnissá tegye az ASIC specializációval és központosítással szemben, mint az egyesített támadásokkal szemben, mivel több bányász algoritmust egyidejűleg kellene támadni.

- Blokk-generálási idő: átlagosan 30 másodperc (2,5perces iránycél bányászalgoritmus típusonként)
- Maximális érme mennyisége: 2 milliárd, 2024-ig
- Konszenzus algoritmus: több algoritmust alkalmazó Proof-of-Work
- Piaci kapitalizáció: 120'000 amerikai dollár 2014 közepén

Blackcoin

A Blackcoin 2014 februárjában jelent meg, és a kockázati (Proof-of-Stake) algoritmust használ konszenzus algoritmusként. Azért érdemel figyelmet, mert bevezette a „multipools”-t, azaz a bányásztársulatoknak egy olyan típusát, amely automatikusan váltani tud a különböző alt-coinok között, attól függően, hogy melyiknél rentábilisabb a bányászat.

- Blokk-generálási idő: 1 perc
- Konszenzus algoritmus: Proof-of-Stake
- Piaci kapitalizáció: 3,7 millió amerikai dollár 2014 közepén

VeriCoin

A VeriCoin 2014 májusában jelent meg. Kockázati (Proof-of-Stake) algoritmust használ a közmegegyezésre, de olyan változó kamatlábbal, amely a piaci kínálat és kereslet szerint, dinamikusan alakul. Ugyancsak ez volt az első olyan alt-coin, amely a pénztárcából történő bitcoinos fizetésekkel automatikusan bitcoinra váltódik át.

- Blokk-generálási idő: 1 perc
- Konszenzus algoritmus: Proof-of-Stake
- Piaci kapitalizáció: 1,1 millió amerikai dollár 2014 első felében

NXT

A NXT (ejtsd: „Next”) egy „színtiszta” Proof-of-Stake alt-coin, legalábbis abban az értelemben, hogy nem alkalmazza a Proof-of-Work bányászatot. Az NXT egy a „semmiből” implementált digitális fizetőeszköz, tehát sem nem a bitcoinnak, sem más alt-coinnak nem az elágazása. Az NXT-nek számos fejlett jellemzője van, pl. a név regisztráció (a Namecoinhoz hasonlóan), decentralizált eszközök cserelehetősége (a Colored Coinokkal megegyezően), beépített decentralizált és biztonságos üzenetváltás (akkáncs a Bitmessage-nél), és kockázat delegálás (a Proof-of-Stake másokra történő átruházása). A NXT hívei a „következő-generációs”, vagy 2.0 kriptopénznek is hívják ezt az alt-coint.

- Blokk-generálási idő: 1 perc
- Konszenzus algoritmus: Proof-of-Stake
- Piaci kapitalizáció: 30 millió amerikai dollár 2014 közepén

Kettős-célú bányászati innovációk: Primecoin, Curecoin, Gridcoin

A bitcoin Proof-of-Work algoritmusának egy célja van: biztosítani a bitcoin hálózat biztonságát. A hagyományos kifizető rendszerek védelméhez hasonlítva a bányászat költségei nem is olyan jelentősek. Ennek ellenére sokak részéről érkezett olyan kritika, hogy a bányászat „költséges”. Az alt-coinok következő csoportja erre próbál megoldást találni. A kettős-célú Proof-of-Work algoritmus egyrészt valamilyen „hasznos” problémát old meg, másrészt a hálózat védelmére is használja a Proof-of-Work rendszert. Ha a fizetőeszköz védelmére szolgáló algoritmus egy külső felhasználással is rendelkezik, akkor ennek az a kockázata, hogy ez a külső hatás is befolyásolni fogja a kereslet- kínálat görbéjét.

Primecoin

A Primecoin 2013 júliusában látott napvilágot. Proof-of-Work algoritmusára prímszámok után kutat, nevezetesen Cunningham- és ikerprím láncok után. A prímszámok a tudomány számos területén hasznosak. A Primecoin blokklánca a munkabizonyíték algoritmussal feltárt prímszámokat tartalmazza, eképp a tudományos felfedezéseknek egyfajta nyilvános tárhelye, emellett egyben a tranzakciók nyilvános jegyzéke is.

- Blokk-generálási idő: 1 perc
- Konszenzus algoritmus: prímszám láncok előállítása révén megvalósított Proof-of-Work rendszer
- Piaci kapitalizáció: 1,3 millió amerikai dollár 2014 közepén

Curecoin

A Curecoin 2013 májusában jelent meg. Az SHA256 Proof-of-Work algoritmust kombinálja a fehérjék feltekeredésének a tanulmányozásával a „Folding@Home” projekten keresztül. A fehérjék feltekeredésének modellezése a fehérjék közötti biokémiai kölcsönhatások számításigényes szimulációját igényli, és a segítségével betegségek új gyógymódjai fedezhetők fel.

- Blokk-generálási idő: 10 perc
- Konszenzus algoritmus: fehérjék feltekeredésének a kutatásával megvalósított Proof-of-Work rendszer
- Piaci kapitalizáció: 58'000 amerikai dollár 2014 közepén

Gridcoin

A Gridcoint 2013 októberében mutatták be. A scrypt-alapú Proof-of-Work rendszert a nyílt forrású BOINC grid-computing-ban való részvételért cserébe adott jutalom egészít ki. A BOINC egy nyílt forráskódú protokoll, amely tudományos kutatások számításainál használható, és amely lehetővé teszi a résztvevők számára, hogy saját számítási kapacitásukat megosszák egy széles körű, akadémiai kutatást szolgáló számítási kapacitással. A Gridcoin a BOINC-ot, mint általános számítási platformot használja, tehát nem olyan specifikus problémák megoldására, mint amilyen például a prímszámok keresése, vagy a fehérjék feltekeredésének a szimulációja.

- Blokk-generálási idő: 150 másodperc

- Konszenzus algoritmus: BOINC grid computing alapján megvalósított, jutalommal kiegészített Proof-of-Work rendszer
- Piaci kapitalizáció: 122'000 amerikai dollár 2014 közepén

Anonimitás-fókuszú alt-coinok: CryptoNote, Bytecoin, Monero, Zerocash/Zerocoin, Darkcoin

A bitcoin, gyakran hibásan, anonim fizetőeszközként is jellemzik, holott adat elemzéssel viszonylag egyszerűen felderíthető egy bitcoin cím mögött megbújó személyazonosság, és a bitcoin címek összekapcsolása révén átfogó képet kaphatunk egy adott felhasználó bitcoin költekezési szokásairól. Számos olyan alt-coin született, amely kifejezetten ennek kérdésnek a megoldására szolgál, és amelyik kifejezetten a fokozott anonimitásra fókuszál. A legelső ilyen próbálkozás valószínűleg a *Zerocoin* volt, amely a bitcoinra épülő, és az anonimitást megőrző meta-coin protokol. A Zerocoint 2013-ban az IEEE Biztonsági és Titkosítási (Security and Privacy) szimpóziumán mutatták be. A Zerocoin egy teljesen külön alt-coinként, Zerocash néven kerül majd megvalósításra, evvel kapcsolatban a könyv írásának az idején is folynak a fejlesztések. Szintén az anonimitásra fókuszáló alternatíva a 2013 októberében egy dolgozatban bemutatott *CryptoNote*. A CryptoNote egy olyan alapvető technológia, amelyet számos alt-coin elágazás használ, és amelyről részletesebben alább lesz szó. A Zerocash-en és a Cryptonote-on kívül még egy sereg független és anonim alt-coint ismerünk, például a Darkcoint, amely lopakodó (stealth) titkos címekkel, vagy tranzakciók összekeverésével valósítja meg az anonimitást.

Zerocoin/Zerocash

A Zerocoin a digitális fizetőeszköz anonimitásának elméleti megközelítése. A fejlesztést 2013-ban mutatták be a Johns Hopkins Egyetem kutatói. A Zerocash a Zerocoin alt-coin megvalósítása, és jelenleg is folyik a fejlesztése.

CryptoNote

A CryptoNote egy olyan referencia alt-coin implementáció, amely egy 2013 októberében bemutatott digitális anonim pénz alapjait biztosítja. Úgy lett kifejlesztve, hogy változatos implementációkhöz szolgálhasson elágazásként, illetve rendelkezik egy beépített periodikus „reset” mechanizmussal is, amely meggátolja, hogy a CryptoNote, mint fizetőeszköz kerüljön felhasználásra. A CryptoNote-ból rengeteg alt-coin született. Ide tartozik például a Bytecoin (BCN), az Aeon (AEON), a Boolberry (BBR), a duckNote (DUCK), a Fantomcoin (FCN), a Monero (XMR), a MonetaVerde (MCN) és a Quazarcoin (QCN). A CryptoNote azért is figyelemre méltó, mert nem a bitcoin egy elágazása, hanem teljes egészében egy újszerű kriptopénz implementáció

Bytecoin

A Bytecoin volt az első CryptoNote-on alapuló implementáció, azaz a CryptoNote technológiáján alapuló első életképes anonim fizetőeszköz. A Bytecoin 2012 júliusában jelent meg. Itt kell megemlítenünk, hogy volt egy korábbi Bytecoin nevű alt-coin is BTE rövidítéssel, míg a CryptoNote derivatíva fizetőeszközének BCN a rövidítése. A Bytecoin a Cryptonight Proof-of-Work algoritmust használja, amely legalább 2 MB RAM-ot igényel bányász példányonként, ezáltal lehetetlenné teszi a GPU vagy az ASIC típusú bányászatot. A

Bytecoin örökölte a CryptoNote-ból a ring-signatures-t, a belinkelhetetlen tranzakciókat és a blokklánc-elemzőkkel szembeni ellenállóképességét az anonimitás megőrzése érdekében.

- Blokk-generálási idő: 2 perc
- Maximális érme mennyisége: 184 milliárd BCN
- Konszenzus algoritmus: Cryptonight Proof-of-Work
- Piaci kapitalizáció: 3 millió amerikai dollár 2014 közepén

Monero

A Monero is egy CryptoNote implementáció. A kibocsátási görbéje a Bytecoinnál egy árnyalatnyival előnyösebb, mivel az érmék 80% az első négy év során kerül kibocsátásra. Anonimitásra vonatkozó tulajdonságait a CryptoNote-tól örökölte, így azzal megegyeznek.

- Blokk-generálási idő: 1 perc
- Maximális érme mennyisége: 18.4 millió XMR
- Konszenzus algoritmus: Cryptonight Proof-of-Work
- Piaci kapitalizáció: 5 millió amerikai dollár 2014 közepén

Darkcoin

A Darkcoin 2014 januárjában indult útjára. Anonim fizetőeszközt implementál egy DarkSend-nek nevezett újra-keverő (re-mixing) protokollt használatával. A Darkcoin azért is érdekes, mert 11 különböző hash funkciót (blake, bmw, groestl, jh, keccak, skein, luffa, cubehash, shavite, simd, echo) használ a Proof-of-Work algoritmushoz.

- Blokk-generálási idő: 2.5 perc
- Maximális érme mennyisége: 22 millió DRK
- Konszenzus algoritmus: több menetes, több algoritmust használó Proof-of-work
- Piaci kapitalizáció: 19 millió amerikai dollár 2014 közepén

Nem fizetőeszközöként használt alt-chainek

Az alt-chainek a blokklánc, mint ötlet alternatív implementációi, és elsősorban nem fizetőeszközként kerülnek alkalmazásra. Számos közülük fizetőeszközt is tartalmaz, de ez a fizetőeszköz inkább mint szimbólum/zseton kerül alkalmazásra, egyéb javak, vagy például egy szerződés szétosztásánál. Más szóval: a fizetőeszköz nem a platform lényegi kérdése, csak egy amolyan másodlagos adottsága.

Namecoin

A Namecoin volt a bitcoin kód első elágazása. A Namecoin egy olyan decentralizált kulcsnyilvántartási és transzfer platform, amely blokkláncot használ. Az Internet domain-név regisztrációs rendszeréhez hasonló, globális domain-név nyilvántartást támogat. A Namecoint jelenleg a .bit gyökérdomain esetében alternatív Domain Név Szolgáltatóként (DNS) alkalmazzák. A Namecoin más névtérben lévő nevek és értékpárok nyilvántartására is alkalmas, például e-mail címek, titkosítási kódok, SSL tanúsítványok, file aláírások, szavazói regiszterek, és számtalan egyéb dolog tárolására.

A Namecoin rendszer a namecoin fizetőeszközt is tartalmazza (szimbóluma NMC), amely a név-regisztrációk és név-cserék tranzakciós díjainak a megfizetésére használható. Jelenleg egy névregrisztráció díja 0,01 NMC, azaz körülbelül 1 cent. A bitcoinhoz hasonlóan, a díjakat a Namecoin bányászok kapják.

A Namecoin alap paraméterei megegyeznek a bitcoinéval:

- Blokk-generálási idő: 10 perc
- Maximális érme mennyisége: 21 millió NMC, 2140-ig.
- Konszenzus algoritmus: SHA256 Proof-of-Work
- Piaci kapitalizáció: 10 millió amerikai dollár 2014 közepén

A Namecoin névtere nem korlátozott, bárki, bárhogyan és bármilyen névteret használhat. Mindamellett, néhány bizonyos névtér rendelkezik egy jóváhagyott specifikációval, így amikor a blokkláncból kerül beolvasásra, a szoftver tudja, hogyan olvassa ki és hajtsa végre. Ha a név nem jólt formált, akkor a használt szoftvertől függetlenül hibát fogunk kapni. Íme, néhány példa a kedvelt névterekből:

- a `d/` a `.bit` domain névtér domain-neve
- az `id/` a személyes adatok, mint például e-mail cím, PGP kulcsok, stb. jegyzékének a névtere
- az `u/` egy további, jobban strukturált specifikáció személyazonosságok tárolására (openspecs alapú)

A Namecoin kliens nagyon hasonlít a Bitcoin Core kliensre, lévén, hogy ugyanabból a forráskóból származik. Az installálásnál a kliens a Namecoin blokkláncnak egy teljes másolatát letölti, majd ezek után készen áll a nevek lekérdezésére és regisztrációjára. Hárrom fő parancsa van:

`name_new`

egy név lekérdezése vagy pre-regisztrálása

`name_firstupdate`

egy név regisztrálása és a regisztráció publikálása

`name_update`

a részletek módosítása, vagy egy névregisztráció frissítése

Például, a `mastering-bitcoin.bit` domain-név regisztrálására a `name_new` parancssal a következőképpen lehetséges:

```
$ namecoind name_new d/mastering-bitcoin
[
    "21cbab5b1241c6d1a6ad70a2416b3124eb883ac38e423e5ff591d1968eb6664a",
    "a05555e0fc56c023"
]
```

A `name_new` parancs egy név-igényt regisztrál, azáltal, hogy létrehozza a névnek egy random kulccsal ellátott hashét. A `name_new` által visszaküldött két string: a hash és a random kulcs (a05555e0fc56c023 a fenti példában), amelynek a felhasználásával a név regisztráció nyilvánossá tehető. Miután az adott igény regisztrálva lett a Namecoin blokkláncában, nyilvános regisztrációjával is konvertálható a `name_firstupdate` parancssal, és a random kulcs megadásával:

```
$ namecoind name_firstupdate d/mastering-bitcoin a05555e0fc56c023 {"map": {"www": {"ip": "1.2.3.4"}}, "b7a2e59c0a26e5e2664948946ebeca1260985c2f616ba579e6bc7f35ec234b01}
```

A fenti példa `www.mastering-bitcoin.bit` domain-nevet az 1.2.3.4. IP címhez rendeli hozzá. A visszaküldött hash a tranzakció id-je, amellyel a regisztráció nyomon követhető. A regisztrált nevek a `name_list` parancs futtatásával nézhetők meg:

```
$ namecoind name_list
[
  {
    "name" : "d/mastering-bitcoin",
    "value" : {"map: {www: {ip:1.2.3.4}}}",
    "address" : "NCccBXrRUahAGrisBA1BLPWQfSrups8Geh",
    "expires_in" : 35929
  }
]
```

A Namecoin regisztrációkat minden 36'000 blokk után (kb. 200-250 naponként) frissíteni kell. A `name_update` parancsnak nincs díja, így a Namecoin domainok megújítása ingyenes. Szerény díj ellenében egy webes interfészen keresztül egy harmadik fél is kezelheti a regisztrációt, valamint az automatikus frissítést és megújítást. Ha egy harmadik fél végzi a műveleteket, akkor szükségtelen, hogy a Namecoin klienst mindenki saját maga futtassa, de ezzel elvész a Namecoin által kínált, decentralizált névnyilvántartás feletti független felügyelet.

Ethereum

Az Ethereum egy blokklánc-nyilvántartáson alapuló, Turing-teljes szerződés feldolgozó és végrehajtó platform. Az Ethereum nem egy bitcoin klón, hanem egy teljesen független elvi alapokon álló implementáció. Az Ethereumnak van saját fizetőeszköze, az úgynevezett *éter* (ether), amely a *szerződések* végrehajtásához szükséges. A szerződéseket az Ethereum blokklánc rögzíti. A szerződések alacsony szintű, bájtkód-szerű, Turing-teljes nyelven vannak ábrázolva. A szerződések lényegében olyan programok, amelyek az egyes csomópontron az Ethereum rendszerén belül futnak. Az Ethereum szerződések adatokat tárolhatnak, ether kifizetéseket küldhetnek és fogadhatnak, ethereket tárolhatnak, illetve végtelen számú (innen a Turing-teljesség) kiszámítható műveleteket végezhetnek, és decentralizált autonóm ügynökként funkcionálhatnak.

Az Ethereummal viszonylag bonyolult rendszerek valósíthatók meg, akár olyanok is, amelyek önmaguk is alt-chain implementációk. Például, alább egy Namecoinhoz hasonló név-regisztrációs szerződést láthatunk Ethereumban megírva (Pontosabban, egy olyan magas szintű nyelven, amely lefordítható Ethereum kódra):

```
if !contract.storage[msg.data[0]]: # Már foglalt a kulcs?  
    # Nem: akkor foglald le!  
    contract.storage[msg.data[0]] = msg.data[1]  
    return(1)  
else:  
    return(0) // Különben ne csinálj semmit
```

A fizetőeszközök jövője

Összességében, a digitális fizetőeszközök jövője még a bitcoinénál is fényesebb. A bitcoin egy teljesen új formájú decentralizált szervezetet és közmegegyezést vezetett be, ami egyben hihetetlen innovációk százainak is teret nyitott. Ezek az innovációk minden bizonnyal hatással lesznek a gazdaság számos szektorára, az elosztott rendszerektől a pénzügyön, a közigazdaságtanon, a fizetőeszközökön, és a központi bankokon keresztül egészen az államháztartásig és a cégek szabályozásáig. Mostantól számos olyan emberi tevékenység válhat decentralizálttá, amelyhez korábban a hiteles és megbízható működés érdekében egy központosított intézményre vagy szervezetre volt szükség. A blokklánc és a közmegegyezés innovációja jelentősen csökkenteni fogja a nagy rendszerek létrehozásának és koordinációjának a költségét, egyben elkerülhető vele a hatalomkoncentráció, a korrupció és a túlszabályozás csapdája.

10. A bitcoin biztonsági kérdései

A bitcoin biztonságos kezelése azért nehéz, mert a bitcoin nem olyan, mint egy bankszámla, vagyis nem egy elvont hivatkozást jelent valamelyen értékre. A bitcoin inkább a digitális pénzhez vagy az aranyhoz hasonlít. Bizonyára hallották már azt a mondást, hogy „A tulajdon a törvény kilenc tizede”. Nos, a bitcoin esetén a tulajdon a törvény tíz tizedét jelenti. A bitcoinok zárolását megszüntető kulcsok birtoklása egyenértékű azzal, mint ha készpénzünk vagy egy darab nemesfémünk lenne. Ha elveszítjük vagy elhagyjuk a kulcsot, vagy ellopják tölünk, ill. ha véletlenül rossz összeget küldünk valakinek, akkor nincs semmilyen mentségünk, pont úgy járunk, mint ha egy forgalmas járdára készpénzt szórnánk.

A bitcoinnak viszont vannak olyan adottságai, amellyel az arany vagy a bankszámlák nem rendelkeznek. A kulcsokat tároló bitcoin pénztárcáról a többi állományhoz hasonlóan biztonsági másolat készíthető. A másolat több példányban tárolható, vagy akár papírra is kinyomtatható, ha fizikai biztonsági másolatra van szükség. A készpénzről, az aranyról vagy a bankszámlákról nem lehet „biztonsági másolatot” készíteni. A bitcoin annyira különbözik minden más őt megelőző dolgotól, hogy a bitcoin biztonsági kérdései is újfajta megközelítést tesznek szükségessé.

Biztonsági alapelvek

A bitcoin legfontosabb sajátossága a decentralizáció, ennek pedig fontos biztonsági következményei vannak. Egy centralizált modell, pl. egy hagyományos banki hálózat a hozzáférési jogokra és ezek vizsgálatára alapoz, hogy a rossz fiúkat a rendszertől távol tartsa. Ezkel szemben egy decentralizált rendszerben a felelősség és a rendelkezési jog a végfelhasználók kezében van. Mivel a hálózat biztonsága nem a hozzáférési jogokon, hanem a munkabizonyitékokon alapul, a hálózat nyílt is lehet, és a bitcoin forgalmához nincs szükség titkosításra.

Egy hagyományos kifizető hálózatban, pl. egy hitelkártya rendszerben a „fizetség” valójában nem egyszeri, mivel tartalmazza a felhasználó privát azonosítóját (a hitelkártya számot). Ennek az azonosítónak a birtokában bárki képes a pénz újból „lehívására”, és a tulajdonostól történő ismételt levonására. Emiatt a kifizető hálózatot a végpontok között titkosítással kell biztonságossá tenni, és biztosítani kell, hogy a forgalmat semmilyen lehallgatás vagy közbenső személy se zavarhassa meg, sem menet közben, sem a tárolás során. Ha egy rossz fiú jut be a rendszerbe, akkor kompromittálni tudja az aktuális tranzakciót és új tranzakciókat tud létrehozni. De ami még ennél is rosszabb, a felhasználó adatai is kompromittálódnak, a személyazonosságának az ellopása fenyegeti, és lépésekkel kell tennie, hogy megakadályozza a kompromittálódott számlák jogtalan használatát.

A bitcoin teljesen más. Egy bitcoin tranzakcióval csak egy adott érték továbbítható egy adott fogadó félnek, a tranzakciót pedig nem lehet meghamisítani vagy módosítani. Semmilyen privát adatot sem lehet megtudni belőle, például a tranzakcióban részt vevő felek személyazonosságát, és nem lehet vele további kifizetéseket eszközölni. Ezért a bitcoin hálózatban nincs szükség titkosításra vagy a lehallgatás elleni védelemre. A bitcoin tranzakciók publikus csatornákon, pl. nem biztonságos Wifi vagy Bluetooth csatornákon is továbbíthatók anélkül, hogy ez bármilyen negatív hatással lenne a biztonságra.

A bitcoin decentralizált biztonsági modellje a végfelhasználók kezébe sok hatalmat ad. De a hatalommal felelősség is jár: biztosítani kell a kulcsok titkosságát. A legtöbb felhasználó számára ez nem egyszerű, különösen ha olyan általános célú számítástechnikai eszközökről van szó, mint az Internethez kapcsolódó okostelefonok vagy notebook-ok. A bitcoin decentralizált modellje megakadályozza ugyan a hitelkártyák esetében tapasztalt tömeges kompromittálódást, de sok végfelhasználó képtelen a kulcsok megfelelő védelmére, és a hackerek egyesével levadásszák őket.

Bitcoin rendszerek biztonságos fejlesztése

A bitcoin rendszerek fejlesztői számára a decentralizáció a legfontosabb alapelve. A legtöbb fejlesztő már találkozott a centralizált biztonsági modellekkel, és emiatt hajlamos arra, hogy az ilyen modelleket alkalmazza a bitcoin fejlesztései során is, ez pedig katasztrofális következményekkel jár.

A bitcoin biztonsága a kulcsok fölötti decentralizált rendelkezési jogon és a bányászok által végzett független tranzakció ellenőrzésén alapul. Ha szeretnénk kihasználni a bitcoin nyújtotta biztonságot, akkor biztosítanunk kell a bitcoin biztonsági modelljének megfelelő működést. Egyszerűen szólva: a fejlesztő ne vegye el a felhasználó kulesök feletti rendelkezési jogát, és a tranzakciókat ne vigye a blokkláncon kívülre.

Például sok korai bitcoin pénzváltóban a felhasználók pénzét egyetlen egy „forró” pénztárcában gyűjtötték össze, melynek kulcsait egyetlen egy szerveren tárolták. Az ilyen felépítés megfosztotta a felhasználókat a felügyeleti joguktól, és a kulcsok fölötti felügyeleti jogot egyetlen rendszerbe koncentrált. Sok ilyen rendszert feltörtek, ami rendkívül hátrányos következményekkel járt ezen rendszerek felhasználói számára.

Egy másik szokásos hiba a tranzakciók „blokkláncon kívüli” kezelése, azzal a hibás szándékkal, hogy csökkentsék a tranzakciós díjakat és felgyorsítsák a tranzakciók feldolgozását. Egy ilyen „blokkláncon kívüli” rendszer a tranzakciókat egy saját központosított főkönyvben tárolja, és a tranzakciókat csak néha szinkronizálja a bitcoin blokkláncával. Ez a gyakorlat ismét csak a bitcoin decentralizált biztonságát helyettesíti egy saját, centralizált módszerrel. Ha a tranzakciók a blokkláncon kívül vannak, akkor a nem elég biztonságos központosított főkönyvek meghamisíthatóak, és a pénzeszközök észrevétlenül elszivárogthatatóak.

Jól gondolja meg, hogy kiemeli-e a pénzeszközök kezelését a bitcoin decentralizált biztonsági környezetéből. Ha így döntene, akkor készüljön fel arra, hogy a működés biztonságát költséges beruházásokkal, számos hozzáférési réteggel és könyvvizsgálókkal kell megteremtenie (ahogyan ezt a hagyományos bankok teszik). Még rendelkezésre is áll a pénz és a tudás egy ilyen robusztus biztonsági modell megteremtésére, az ilyen felépítés csupán a hagyományos pénzügyi hálózatok törékeny modelljét utánozza, amelyben súlyos problémát jelent a személyazonosság ellopása, a korrupció és a hűtlen kezelés. Ha szeretné a bitcoin decentralizált modelljének előnyeit élvezni, akkor el kell kerülnie azt a kísértést, melyet a centralizált architektúrák/modellek jelentenek. Lehet, hogy ezek közismertebbek, de végső soron a bitcoin biztonságát sodorják veszélybe.

A bizalom alapja

A hagyományos biztonsági rendszerek egy olyan fogalmon alapulnak, melynek neve a *bizalom alapja* (*root of trust*). Ez nem más, mint egy biztonságos mag, amelyen az egész rendszer vagy alkalmazás biztonsága alapul. A biztonsági megoldások koncentrikus körök formájában, a hagymahéjakhoz hasonlóan e köré épülnek, és a biztonságot a kiindulóponttól kifelé terjesztik ki. minden egyes réteg a megbízhatóbb belső rétegre épül, és hozzáférési jogokat, digitális aláírásokat, titkosítást és más biztonsági alapelemeket használ. Az egyre bonyolultabb szoftver rendszerek egyre valószínűbb, hogy hibákat fognak tartalmazni, ami biztonsági szempontból kiszolgáltatottá teszi őket. Emiatt minél bonyolultabb egy szoftver rendszer, annál nehezebb a biztonságát garantálni. A bizalom alapja révén biztosítható, hogy leginkább a rendszer legkevésbé bonyolult, és emiatt legkevésbé támadható részében bízzunk, és a bonyolultabb szoftvert e köré építük. Ez a biztonsági architektúra különféle szinteken ismétlődik: a bizalom alapját először egyetlen egy rendszer hardverén belül teremti meg, majd a bizalomnak ezt az alapját terjeszti ki az operációs rendszeren keresztül a magasabb szintű rendszer-szolgáltatásokra, és végül a számos szerverre, melyek a csökkenő bizalom koncentrikus köreit alkotják.

A bitcoin biztonsági architektúrája nem ilyen. A bitcoin esetében a konszenzus-rendszer teremti meg a biztonságos főkönyvet, amely teljesen decentralizált. A blokkláncban a genezis blokk a bizalom alapja, és az ellenőrzések révén jön létre a bizalmi lánc egészen az aktuális blokkig bezárólag. A bitcoin rendszerekben a blokklánc használható, és a blokkláncot kell használni a bizalom alapjaként. Ha ön bonyolult bitcoin alkalmazásokat tervez, melyek sok különböző rendszeren futó szolgáltatásból épülnek föl, akkor gondosan meg kell vizsgálnia, hogy a biztonsági rendszeren belül hol van a rendszer biztonságának a kiindulópontja. Végső soron csak egyetlen dologban szabad explicit módon megbíznia: a teljesen ellenőrzött blokkláncban. Ha az ön alkalmazása explicit vagy implicit módon a blokkláncon kívül bármiben másban megbízik, abból problémák adódhatnak, mivel támadható pontokat teremt a rendszeren belül. Az alkalmazás biztonságának vizsgálatára jó módszert jelent minden egyes összetevő külön történő vizsgálata, és annak a képzelt esetnek a kiértékelése, hogy mi történne, ha a vizsgált összetevő teljesen kompromittálódna, és a rossz fiúk fennhatósága alá kerülne. Sorban egymás után tekintse át az alkalmazásá elemeit, és vizsgálja meg, hogy milyen hatással lenne a biztonságra, ha az illető elem kompromittálódna. Ha az elemek kompromittálódását követően az alkalmazás már nem biztonságos, akkor ez azt mutatja, hogy ön implicit módon megbízott ezekben az elemekben. Egy nem sérülékeny bitcoin alkalmazás csak a bitcoin konszenzus-mechanizmusának kompromittálódása révén támadható. Ez azt jelenti, hogy a bizalom alapját a bitcoin biztonsági architektúra legerősebb része jelenti.

A meghackelt bitcoin váltók ez előbbieket számos példával támasztották alá, mert a biztonsági architektúrájuk még a legfelületesebb vizsgálaton sem megy át. A váltók centralizált megvalósítása során a blokkláncon kívül számos más építőelembe helyeztek explicit bizalmat, például a pénztárcákba, a centralizált főkönyvi adatbázisokba, a támadható titkosító kulcsokba stb.

A felhasználók számára követendő példák

Az emberek évezredek óta használnak fizikai biztonsági eszközöket. Ezzel szemben a digitális biztonságra vonatkozó tapasztalatok kevesebb mint 50 évesek. A modern általános célú operációs rendszerek nem nagyon biztonságosak, és nem nagyon alkalmásak digitális pénz tárolására. Számítógépeinket állandóan külső támadásoknak tesszük ki az állandó

Internet összeköttetések révén. Szoftver komponensek ezreit futtatjuk, melyek több száz különböző szerzőtől származnak, és gyakran korlátozás nélkül elérik az adatállományokat. Elég egyetlen egy rossz szándékú szoftver a több ezerből ahhoz, hogy kompromittálja a billentyűzetet és az adatállományokat, és ellopja a pénztárcákból az ott tárolt bitcoinokat. Az a feladat, hogy a számítógép vírusmentes és trójai-mentes állapotban legyen, egy elenyésző töredék kivételével a legtöbb felhasználó üzemeltetési tudását meghaladja.

Az információ-biztonság évtizedes kutatási eredményei és fejlődése ellenére a digitális javak még mindig nagyon sérülékenyek egy céltudatos ellenfellel szemben. Még a pénzügyi szolgáltatók, hírszerző szervezetek és hadiipari szállítók legvédettebb és korlátozottan használható rendszereit is gyakran feltörök. A bitcoin olyan digitális javakat teremtett, amelyeknek belső értéke van, ellophatóak, azonnal és visszavonhatatlanul az új felhasználókhöz irányíthatók át. Ez nagyon nagy motiváció a hackerek számára. Eddig a hackereknek a személyazonosságot vagy a számla adatokat (pl. hitelkártya számokat, bankszámla adatokat) más értékre kellett konvertálniuk, ha sikerült kompromittálniuk őket. A pénzügyi adatok védelme és a pénzmosás elleni törvények ellenére egyre nagyobb lopásokat láthattunk. A bitcoin elmélyíti ezt a problémát, mivel itt nincs szükség a pénzmosásra - a bitcoin egy digitális jószág, melynek belső értéke van.

Szerencsére a bitcoin a számítógépes biztonság növekedését is elősegíti. A bitcoint megelőzően a számítógép kompromittálódásának a veszélye közvetett és bizonytalan volt, a bitcoin viszont ezt a veszélyt nyilvánvalóvá és világossá tette. A bitcoin számítógépen történő tárolása ráirányította a figyelmet arra, hogy fokozott biztonságra van szükség. A bitcoin és más digitális pénzek elterjedésének közvetlen következményeként mind a hackelési módszerek, mind a biztonsági megoldások terén eszkalálódott a helyzet. Egyszerű szavakkal, a hackerek számára a bitcoin nagyon csábító cél jelent, a felhasználók pedig minden módon szeretnék megvédeni magukat.

Az utóbbi három évben a bitcoin elterjedésének közvetlen eredményeképpen hatalmas innováció zajlott le az információbiztonság területén, a hardveres titkosítás, a hardveres kulcsárolás és hardveres pénztárcák, a többszörös aláírás valamint a digitális letét terén. A következő részekben a gyakorlati biztonság legjobb, legkövetendőbb példáit vizsgáljuk meg.

Fizikai bitcoin tárolás

Mivel a felhasználók többsége számára a fizikai biztonság sokkal kézzelfoghatóbb az információbiztonságnál, a bitcoinok védelmének egy nagyon hatékony módja, ha fizikai alakra konvertáljuk őket. A bitcoin kulcsok nem mások, mint hosszú számok. Ez azt jelenti, hogy fizikai alakban tárolhatók, például kinyomtathatók vagy ráváhetők egy érmére. A kulcsok védelme ekkor egyszerűen a kinyomtatott bitcoin kulcsok fizikai védelmét jelenti. A papírra kinyomtatott bitcoin kulcsokat „papír pénztárcának” hívják. Sok szabadon használható eszköz van, mellyel papír tárcák hozhatók létre. Személy szerint én a bitcoinjaim túlnyomó többségét (99%-ánál is többet) papír pénztárcákban tárolom, BIP0038 szerinti titkosítással, több példányban kinyomtatva, és páncélszekrényekbe elzárva. A bitcoinok offline módon történő *hideg tárolása* az egyik leghatékonyabb védelmi módszer. Hideg tároláskor a kulcsok egy offline rendszeren lettek előállítva (vagyis egy olyan rendszeren, amely soha sem volt az Internetre kapcsolva), és offline módon, papíron, vagy más digitális tároló eszközön, pl. USB kulcson tárolják őket.

Hardverrel megvalósított pénztárcák

Hosszabb távon a bitcoinok biztonságát egyre inkább a külső módosításokkal szemben védett hardver pénztárcák fogják megteremteni. Az okostelefonokkal vagy az asztali számítógépekkel ellentétben az ilyen célra épített hardver pénztárcáknak csak egy célja és feladata van - a bitcoinok biztonságos tárolása. Mivel nincs bennük általános célú szoftver, amely kompromittálható, és az interfészeik a célnak megfelelően korlátozottak, ezért a hardver pénztárcákkal a laikus felhasználók szinte korlátlan biztonsághoz juthatnak. Azt várom, hogy a jövőben túlnyomó részben hardver pénztárcákat fognak használni a bitcoinok tárolására. Egy ilyen hardver pénztárca például a [Trezor](#).

Kockázat kezelés

Míg a legtöbb felhasználó jogosan a lopás miatt aggódik, van egy még nagyobb veszély: a kulcsok elvesztése. Adatállományok bármikor elveszhetnek, de ha bitcoin tárolására használták őket, akkor a veszteség még fájdalmasabb. Miközben szeretnénk biztonságban tudni a bitcoin pénztárcáinkat, vigyázunk kell arra, hogy ne menjünk túl messzire, mert emiatt is elveszíthetjük a bitcoinjainkat. 2011 júliusában egy jól ismert bitcoin szervezet majdnem 7000 bitcoint vesztett el ily módon. A lopás ellen úgy védekeztek, hogy titkosított biztonsági másolatok bonyolult rendszerét használták. Végül véletlenül elveszítették a titkosító kulcsokat, emiatt a biztonsági másolatok nem értek semmit, és egy vagyont veszítettek. Olyan ez, mint ha pénzt ásnánk el a sivatagban: ha túl jól sikerül a dolog, akkor nem biztos, hogy megtaláljuk az elásott kincset.

Kockázat megosztás

Vajon öröök az egész vagyonukat készpénzben tartják a pénztárcájukban? A legtöbben ezt meggondolatlanságnak tartanák, ugyanakkor a bitcoin felhasználók gyakran az összes bitcoinjukat egyetlen pénztárcában tartják. Érdemes inkább több különböző bitcoin pénztárca között szétosztani a kockázatot. Egy óvatos felhasználó csak a bitcoinjainak egy kis részét, mondjuk kevesebb mint 5%-át tartja „zsebpénzként” az online vagy mobil pénztárcájában. A többöt különféle egyéb tárolási módszerekkel pl. az asztali számítógépen lévő pénztárca és az offline pénztárcák (hideg tárolók) között érdemes szétosztania.

A többszörös aláírás és a vállalatirányítás

Ha egy cég vagy egy személy nagy mennyiséggű bitcoint tárol, akkor érdemes megfontolnia a több aláírást megkövetelő (multi-signature) bitcoin címek használatát. A több aláírást megkövetelő címek úgy biztosítják a pénzösszegek biztonságát, hogy a kifizetéshez egynél több aláírára van szükség. Az aláíró kulcsokat különböző helyeken kell tárolni, és különböző emberek fennhatósága alá kell helyezni. Vállalati környezetben például a kulcsokat egymástól függetlenül kell előállítani, és az igazgatósági tagok között kell szétosztani ily módon, hogy önmagában egyetlen személy se tudjon hozzájutni a pénzösszegekhez. A többszörös aláírást megkövetelő címekkel redundancia is megvalósítható. Ebben az esetben egyetlen személynek van több kulcsa, és a kulcsok különböző helyeken vannak tárolva.

Túlélési képesség

Az egyik gyakran elhanyagolt biztonsági szempont, hogy hogyan lehet hozzájutni a bitcoinokhoz, különösen a kulcs tulajdonosának cselekvésképtelensége vagy halála esetén. A

bitcoin felhasználóknak adott egyik tanács az, hogy használjanak bonyolult jelszavakat, és tartsák a kulcsait biztonságban és titokban, valamint hogy a kulcsokra vonatkozó információkat senkivel se osszák meg. Sajnos, az ilyen gyakorlat szinte teljesen kizárja, hogy a felhasználó családjához hozzájusson a pénzhez, ha az eredeti tulajdonos nem képes megszüntetni a pénz zárolását. Mi több, a legtöbb esetben a bitcoin felhasználók családjához nem is tud a bitcoinban tartott tőkéről.

Ha önnek sok bitcoinja van, akkor érdemes a hozzáférésre vonatkozó adatokat egy bizalmas rokonnal vagy egy jogással megosztania. Bonyolultabb túlélési módszert lehet létrehozni többszörös aláírással vagy egy „digitális örökség” kezelésére szakosodott jogász segítségét igénybe véve.

Befejezés

A bitcoin egy teljesen új, eddig példa nélkül álló és bonyolult technológia. Idővel jobb biztonsági eszközök és megoldások kifejlesztése várható, melyek a laikusok számára is könnyebben használhatók lesznek. Jelen pillanatban a bitcoin felhasználók a fenti tanácsok alkalmazásával tudják a biztonságos és gondtalan bitcoin használatot biztosítani maguknak.

A. függelék: A Script tranzakciós nyelv műveletei, konstansai és szimbólumai

Az A-1. táblázat azokat a műveletek mutatja, amelyekkel érték helyezhető a veremtárra.

A-1. táblázat: Érték elhelyezése a veremre

Szimbólum	Érték (hex)	Leírás
OP_0 vagy OP_FALSE	0x00	Egy üres tömb kerül a veremre
1-75	0x01- 0x4b	A következő N bájt tárolása a veremben, ahol N 1 és 75 bájt között van
OP_PUSHDATA1	0x4c	A következő script bájt tartalmazza N-et, N bájt tárolása a veremben
OP_PUSHDATA2	0x4d	A következő két script bájt tartalmazza N-et, N byte tárolása a veremben
OP_PUSHDATA4	0x4e	A következő négy script bájt tartalmazza N-et, N byte tárolása a veremben
OP_1NEGATE	0x4f	A „-1” érték tárolása a veremben
OP_RESERVED	0x50	Leállás – Érvénytelen tranzakció, kivéve, ha egy nem végrehajtott OP_IF ágban van
OP_1 or OP_TRUE	0x51	Az „1” érték tárolása a veremben
OP_2 to OP_16	0x52- 0x60	Az OP_N esetén az „N” tárolása a veremben, Pl. az OP_2 a „2”-t tárolja

Az A-2. táblázat a feltételes vezérlésátadó műveleteket mutatja.

A-2. táblázat: Feltételes vezérlésátadás

Szimbólum	Érték (hex)	Leírás
OP_NOP	0x61	Semmit sem csinál
OP_VER	0x62	Leállás – Érvénytelen tranzakció, kivéve, ha egy nem végrehajtott OP_IF ágban van
OP_IF	0x63	A következő utasítások végrehajtása, ha a verem teteje nem 0
OP_NOTIF	0x64	A következő utasítások végrehajtása, ha a verem teteje 0
OP_VERIF	0x65	Leállás – Érvénytelen tranzakció
OP_VERNOTIF	0x66	Leállás – Érvénytelen tranzakció
OP_ELSE	0x67	Csak akkor hajtandó végre, ha az előző utasítások nem voltak végrehajtva
OP_ENDIF	0x68	Az OP_IF, OP_NOTIF, OP_ELSE blokk vége

Szimbólum	Érték (hex)	Leírás
OP_VERIFY	0x69	A verem tetejének ellenőrzése. Leállás és a tranzakció érvénytelenítése, ha nem TRUE
OP_RETURN	0x6a	Leállás és a tranzakció érvénytelenítése

Az A-3. táblázat a verem kezelő műveleteket mutatja.

A-3. táblázat: Verem műveletek

Szimbólum	Érték (hex)	Leírás
OP_TOALTSTACK	0x6b	A legfölső téTEL kivétele a veremből és elhelyezése egy alternatív veremben
OP_FROMALTSTACK	0x6c	A legfölső téTEL kivétele az alternatív veremből és elhelyezése a veremben
OP_2DROP	0x6d	Két téTEL eltávolítása a veremből
OP_2DUP	0x6e	A verem legfölső két téTELének megduplázása
OP_3DUP	0x6f	A verem legfölső három téTELének megduplázása
OP_2OVER	0x70	A verem harmadik és negyedik téTELének átmásolása a verem tetejére
OP_2ROT	0x71	A verem ötödik és hatodik téTELének átmozgatása a verem tetejére
OP_2SWAP	0x72	A verem tetején lévő két téTEL cseréje
OP_IFDUP	0x73	A verem legfölső elemének megduplázása, ha az elem nem 0
OP_DEPTH	0x74	A veremben lévő téTElek számának elhelyezése a verem tetején
OP_DROP	0x75	A legfölső téTEL eltávolítása a veremből
OP_DUP	0x76	A legfölső téTEL megduplázása
OP_NIP	0x77	A veremben lévő második téTEL eltávolítása a veremből
OP_OVER	0x78	A veremben lévő második téTEL lemásolása és elhelyezése a verem tetején
OP_PICK	0x79	A verem tetején lévő szám (N) eltávolítása, majd a számnak megfelelően az N-ik téTEL átmásolása a verem tetejére
OP_ROLL	0x7a	A verem tetején lévő szám (N) eltávolítása, majd a számnak megfelelően az N-ik téTEL átmozgatása a verem tetejére
OP_ROT	0x7b	A verem legfölső három elemének a forgatása
OP_SWAP	0x7c	A verem legfölső három elemének a cseréje
OP_TUCK	0x7d	A verem legfölső elemének másolása, majd beillesztése a legfölső és a második elem közé

Az A-4. tábláza a string műveleteket mutatja.

A-4. táblázat: String műveletek

Szimbólum	Érték (hex)	Leírás
<i>OP_CAT</i>	0x7e	Letiltva (A két felső téTEL összefűzése)
<i>OP_SUBSTR</i>	0x7f	Letiltva (Egy rész-stringet ad vissza)
<i>OP_LEFT</i>	0x80	Letiltva (A string bal oldali rész stringjét adja vissza)
<i>OP_RIGHT</i>	0x81	Letiltva (A string jobb oldali rész stringjét adja vissza)
<i>OP_SIZE</i>	0x82	Kiszámítja a verem tetején lévő string hosszát, és az eredményt a verem tetejére helyezi

Az A-5. táblázat a bináris aritmetikai és logikai műveleteket mutatja.

A-5. táblázat: Bináris aritmetikai és logikai műveletek

Szimbólum	Érték (hex)	Leírás
<i>OP_INVERT</i>	0x83	Letiltva (Negálja a verem tetején lévő téTEL bitjeit)
<i>OP_AND</i>	0x84	Letiltva (A két legföLső téTEL logikai ÉS kapcsolata)
<i>OP_OR</i>	0x85	Letiltva (A két legföLső téTEL logikai VAGY kapcsolata)
<i>OP_XOR</i>	0x86	Letiltva (A két legföLső téTEL logikai XOR kapcsolata)
<i>OP_EQUAL</i>	0x87	TRUE (1) értéket helyez a verebe, ha a két legföLső téTEL pontosan azonos, egyébként FALSE (0) értéket
<i>OP_EQUALVERIFY</i>	0x88	Ua., mint az <i>OP_EQUAL</i> , de ez után egy <i>OP_VERIFY</i> futtatása, amely leállítja a további futást, ha nem TRUE volt az eredmény
<i>OP_RESERVED1</i>	0x89	Leállás – Érvénytelen tranzakció, kivéve, ha egy nem végrehajtott <i>OP_IF</i> ágban fordul elő
<i>OP_RESERVED2</i>	0x8a	Leállás – Érvénytelen tranzakció, kivéve, ha egy nem végrehajtott <i>OP_IF</i> ágban fordul elő

Az A-6. táblázat a numerikus (aritmetikai) műveleteket mutatja.

A-6. táblázat: Numerikus műveletek

Szimbólum	Érték (hex)	Leírás
<i>OP_1ADD</i>	0x8b	A legföLső téTELhez hozzáad 1-ET
<i>OP_1SUB</i>	0x8c	A legföLső téTELből levon 1-ET
<i>OP_2MUL</i>	0x8d	Letiltva (A legföLső téTELt megszorozza 2-vel)

Szimbólum	Érték (hex)	Leírás
OP_2DIV	0x8e	Letiltva (A legfölső tételel elosztja 2-vel)
OP_NEGATE	0x8f	Megváltoztatja a legfölső tételel előjelét
OP_ABS	0x90	A legfölső tételel előjelét pozitívra változtatja
OP_NOT	0x91	Ha a legfölső tételel 0 vagy 1, akkor átváltoztatja 1-re vagy 0-ra, egyébként 0-t ad vissza
OP_0NOTEQUAL	0x92	Ha a legfölső tételel 0, akkor 0-t ad vissza, egyébként 1-et
OP_ADD	0x93	Eltávolítja a két legfölső tételel, összeadja őket, és az eredményt a verem tetejére helyezi
OP_SUB	0x94	Eltávolítja a két legfölső tételel, az elsőt kivonja a másodikból, és az eredményt a verem tetejére helyezi
OP_MUL	0x95	Letiltva (A két legfölső tételel összeszorzása)
OP_DIV	0x96	Letiltva (A második tételel osztása az elsővel)
OP_MOD	0x97	Letiltva (Maradék, ha a második tételel elosztjuk az elsővel)
OP_LSHIFT	0x98	Letiltva (A második tételel balra tolása annyi bittel, amennyi az első tételel)
OP_RSHIFT	0x99	Letiltva (A második tételel jobbra tolása annyi bittel, amennyi az első tételel)
OP_BOOLAND	0x9a	A két legfölső tételel logikai AND-je
OP_BOOLOR	0x9b	A két legfölső tételel logikai OR-ja
OP_NUMEQUAL	0x9c	TRUE-t ad vissza, ha a két legfölső tételel mint szám egyenlő
OP_NUMEQUALVERIFY	0x9d	U.a mint a NUMEQUAL, de egy OP_VERIFY leállítja a futást, ha az eredmény nem TRUE
OP_NUMNOTEQUAL	0x9e	TRUE-t ad vissza, ha a két legfölső tételel számként értelmezve nem egyenlő
OP_LESSTHAN	0x9f	TRUE-t ad vissza, ha a második tételel kisebb, mint a legfölső tételel
OP_GREATERTHAN	0xa0	TRUE-t ad vissza, ha a második tételel nagyobb, mint a legfölső tételel
OP_LESSTHANOREQUAL	0xa1	TRUE-t ad vissza, ha a második tételel a legfölső tételel nélkül kisebb vagy egyenlő
OP_GREATERTHANOREQUAL	0xa2	TRUE-t ad vissza, ha a második tételel a legfölső tételel nélkül nagyobb vagy egyenlő
OP_MIN	0xa3	A két legfölső tételel közül a kisebbet adja vissza
OP_MAX	0xa4	A két legfölső tételel közül a nagyobbat adja vissza
OP_WITHIN	0xa5	TRUE-t ad vissza, ha a harmadik tételel a második tételel és az első tételel között van (vagy egyenlő a második tételel)

Az A-7. táblázat a kriptográfiai műveleteket mutatja.

A-7. táblázat: Kriptográfiai és hash műveletek

Szimbólum	Érték (hex)	Leírás
OP_RIPEMD160	0xa6	legfölső tétel RIPEMD160 hashét adja vissza
OP_SHA1	0xa7	A legfölső tétel SHA1 hashét adja vissza
OP_SHA256	0xa8	A legfölső tétel SHA256 hashét adja vissza
OP_HASH160	0xa9	A legfölső tétel RIPEMD160(SHA256(x)) hashét adja vissza
OP_HASH256	0xaa	A legfölső tétel SHA256(SHA256(x)) hashét adja vissza
OP_CODESEPARATOR	0xab	Az aláírással ellenőrzött adat kezdetét jelöli
OP_CHECKSIG	0xac	Eltávolítja a veremről a nyilvános kulcsot és az aláírást, és ellenőrzi, hogy az aláírás megfelel-e a tranzakció hashelt adatának, TRUE-t ad vissza, ha igen
OP_CHECKSIGVERIFY	0xad	U.a. mint a CHECKSIG, de egy OP_VERIFY megállítja a végrehajtást, ha az eredmény nem TRUE
OP_CHECKMULTISIG	0xae	A CHECKSIG futtatása minden egyes megadott nyilvános kulcs és aláírás párra. Az összesnek egyeznie kell. Egy implementációs hiba miatt eggyel több értéket távolít el a veremről. Az OP_NOP előtaggal megkerülhető a hiba.
OP_CHECKMULTISIGVERIFY	0xaf	U.a. mint a CHECKMULTISIG, de egy OP_VERIFY megállítja a végrehajtást, ha az eredmény nem TRUE

Az A-8. táblázat az üres műveletet mutatja.

A-8. táblázat: Üres műveletek

Szimbólum	Érték (hex)	Leírás
OP_NOP1-OP_NOP10	0xb0-0xb9	Nem csinál semmit sem, figyelmen kívül marad

Az A-9. táblázat az elemző belső működésére fenntartott műveleti kódokat mutatja.

A-9. táblázat: Az elemző belső működésére fenntartott műveleti kódok

Szimbólum	Érték (hex)	Leírás
OP_SMALLDATA	0xf9	Kis adatmezőt képvisel
OP_SMALLINTEGER	0xfa	Kis egész adatmezőt képvisel
OP_PUBKEYS	0xfb	Nyilvános kulcs mezőket képvisel
OP_PUBKEYHASH	0xfd	Nyilvános kulcs hash mezőt képvisel
OP_PUBKEY	0xfe	Nyilvános kulcs mezőt képvisel
OP_INVALIDOPCODE	0xff	Bármilyen, jelenleg nem kiosztott műveleti kódot képvisel

B. függelék: Bitcoin továbbfejlesztési javaslatok (Bitcoin Improvement Proposals)

A bitcoin továbbfejlesztési javaslatok olyan tervezési dokumentumok, melyek a bitcoin közösség számára adnak információkat, vagy egy új bitcoin jellemzőt, folyamatot vagy környezetet írnak le.

A *BIP Purpose and Guidelines* (*_A BIP-ek célja*) című BIP0001 szerint a BIP-eknek három fajtája van:

Szabvány BIP

Azokat a változtatásokat írja le, melyek a bitcoin implementációk nagy részét vagy egészét érintik. Ilyen pl. a hálózati protokoll megváltoztatása, a blokk vagy tranzakció ellenőrzési szabályok megváltoztatása, vagy bármilyen egyéb változás vagy bővítés, amely érinti a bitcoint használó alkalmazások együttműködését.

Tájékoztatásra szolgáló BIP

Valamilyen tervezési kérdés leírása, vagy általános útmutató, ill. tájékoztatás a bitcoin közösség számára. Nem tartalmaz új jellemzőt. A tájékoztatásra szolgáló BIP-ek nem feltétlenül képviselik a közösségi közmegegyezést vagy nem feltétlenül jelentenek ajánlást, amiatt a felhasználók és fejlesztők szabadon eldönthetik, hogy figyelmen kívül hagyják a BIP-et, vagy követik a tanácsát.

Folyamatot leíró BIP

Egy bitcoin folyamatot ír le, vagy valamilyen változtatást javasol egy folyamatban vagy egy esemény beillesztését javasolja egy folyamatba. A folyamatot leíró BIP-ek olyanok, mint a szabvány BIP-ek, de a bitcoin protokollen túlmenően más területekre is vonatkozhatnak. Javasolhatnak megvalósításokat, de ez a bitcoin kódmezőt nem érintheti. Gyakran közösségi konsenzust igényelnek. A tájékoztatásra szolgáló BIP-ektől eltérően a folyamat leíró BIP-ek nem csupán ajánlások, a felhasználók pedig általában nem hagyhatják őket figyelmen kívül. Például ilyenek az eljárásokra, útmutatókra és a döntési folyamat megváltoztatására vonatkozó BIP-ek, valamint a bitcoin fejlesztési folyamatban használt eszközök vagy fejlesztési környezet megváltoztatása. A meta-BIP-eket szintén folyamat leíró BIP-eknek tekintik.

A bitcoin továbbfejlesztési javaslatokat egy verziótól [GitHub](#) gyűjteményben tárolják. A B-1. táblázat a BIP-ek 2014 őszi pillanatfelvételét mutatja. A létező BIP-ekről és azok aktuális tartalmáról a mérvadó gyűjteményből tájékozódhat.

B-1. táblázat: Pillanatfelvétel a BIP-ekről

BIP #	Hivatkozás	Cím	Tulajdonos	Típus	Státusz
1	https://github.com/bitcoin/bips/blob/master/bip-0001.mediawiki	A BIP-ek célja. Útmutató	Amir Taaki	Szabvány	Aktív
10	https://github.com/bitcoin/bips/blob/master/bip-0010.mediawiki	Multi-Sig tranzakciók megoszlása	Alan Reiner	Tájékoztató	Tervezet
11	https://github.com/bitcoin/bips/blob/master/bip-0011.mediawiki	M-of-N szabványos tranzakciók	Gavin Andresen	Szabvány	Elfogadva
12	https://github.com/bitcoin/bips/blob/master/bip-0012.mediawiki	OP_EVAL	Gavin Andresen	Szabvány	Visszavonva
13	https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki	A pay-to-script-hash címformátuma	Gavin Andresen	Szabvány	Végső
14	https://github.com/bitcoin/bips/blob/master/bip-0014.mediawiki	Protocol Version and User Agent	Amir Taaki, Patrick Strateman	Szabvány	Elfogadva
15	https://github.com/bitcoin/bips/blob/master/bip-0015.mediawiki	Aliases	Amir Taaki	Szabvány	Visszavonva
16	https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki	Pay To Script Hash	Gavin Andresen	Szabvány	Elfogadva
17	https://github.com/bitcoin/bips/blob/master/bip-0017.mediawiki	OP_CHECKHASHVERIFY (CHV)	Luke Dashjr	Visszavonva	Tervezet
18	https://github.com/bitcoin/bips/blob/master/bip-0018.mediawikilink;	hashScriptCheck	Luke Dashjr	Szabvány	Tervezet
19	https://github.com/bitcoin/bips/blob/master/bip-0019.mediawiki	M-of-N szabványos tranzakciók (Low SigOp)	Luke Dashjr	Szabvány	Tervezet
20	https://github.com/bitcoin/bips/blob/master/bip-0020.mediawiki	URI Scheme	Luke Dashjr	Szabvány	Mással lett felváltva
21	https://github.com/bitcoin/bips/blob/master/bip-0021.mediawiki	URI Scheme	Nils Schneider, Matt Corallo	Szabvány	Elfogadva
22	https://github.com/bitcoin/bips/blob/master/bip-0022.mediawiki	getblocktemplate - alapok	Luke Dashjr	Szabvány	Elfogadva
23	https://github.com/bitcoin/bips/blob/master/bip-0023.mediawiki	getblocktemplate - társult bányászat	Luke Dashjr	Szabvány	Elfogadva

BIP #	Hivatkozás	Cím	Tulajdonos	Típus	Státusz
30	https://github.com/bitcoin/bips/blob/master/bip-0030.mediawiki	Dupla tranzakciók	Pieter Wulle	Szabvány	Elfogadva
31	https://github.com/bitcoin/bips/blob/master/bip-0031.mediawiki	Pong üzenet	Mike Hearn	Szabvány	Elfogadva
32	https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki	Hierarchikus determinisztikus pénztárcák	Pieter Wulle	Tájékoztató	Elfogadva
33	https://github.com/bitcoin/bips/blob/master/bip-0033.mediawiki	Stratum csomópontok	Amir Taaki	Szabvány	Tervezet
34	https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki	Block v2, magasság a coinbase-ben	Gavin Andresen	Szabvány	Elfogadva
35	https://github.com/bitcoin/bips/blob/master/bip-0035.mediawiki	mempool üzenet	Jeff Garzik	Szabvány	Elfogadva
36	https://github.com/bitcoin/bips/blob/master/bip-0036.mediawiki	Egyedi szolgáltatások	Stefan Thomas	Szabvány	Tervezet
37	https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki	Bloom szűrők	Mike Hearn and Matt Corallo	Szabvány	Elfogadva
38	https://github.com/bitcoin/bips/blob/master/bip-0038.mediawiki	Jelmondattal védett titkos kulcs	Mike Caldwell	Szabvány	Tervezet
39	https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki	Mnemonikok a determinisztikus kulcsok előállításához	Slush	Szabvány	Tervezet
40		Stratum wire protocol	Slush	Szabvány	BIP szám hozzárendelv
41		Stratum mining protocol	Slush	Szabvány	BIP szám hozzárendelv
42	https://github.com/bitcoin/bips/blob/master/bip-0042.mediawiki	A bitcoin véges pénzkészlete	Pieter Wulle	Szabvány	Tervezet
43	https://github.com/bitcoin/bips/blob/master/bip-0043.mediawiki	Cél mező determinisztikus pénztárcákhoz	Slush	Szabvány	Tervezet
44	https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki	Számlák hierarchiája determinisztikus pénztárcákban	Slush	Szabvány	Tervezet

BIP #	Hivatkozás	Cím	Tulajdonos	Típus	Státusz
50	https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki	A 2013. márciusi fork utólagos vizsgálata	Gavin Andresen	Tájékoztató	Tervezet
60	https://github.com/bitcoin/bips/blob/master/bip-0060.mediawiki	Fix hosszúságú „version” üzenet (Relay-Transactions mező)	Amir Taaki	Szabvány	Tervezet
61	https://github.com/bitcoin/bips/blob/master/bip-0061.mediawiki	„reject” P2P üzenet	Gavin Andresen	Szabvány	Tervezet
62	https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki	A tranzakciók változékonyságának kezelése	Pieter Wille	Szabvány	Tervezet
63		Lopakodó címek	Peter Todd	Szabvány	BIP szám hozzárendelv
64	https://github.com/bitcoin/bips/blob/master/bip-0064.mediawiki	getutxos üzenet	Mike Hearn	Szabvány	Tervezet
70	https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki	Payment protokoll	Gavin Andresen	Szabvány	Tervezet
71	https://github.com/bitcoin/bips/blob/master/bip-0071.mediawiki	Payment protokoll MIME típusok	Gavin Andresen	Szabvány	Tervezet
72	https://github.com/bitcoin/bips/blob/master/bip-0072.mediawiki	Payment protokoll URI-k	Gavin Andresen	Szabvány	Tervezet
73	https://github.com/bitcoin/bips/blob/master/bip-0073.mediawiki	Használjunk „Accept” fejet a Payment kérések URL-jeiben	Stephen Pair		

C. függelék: pycoin, ku és tx

A [pycoin](#) Python könyvtárat eredetileg Richard Kiss írta és tartotta karban. A pycoin egy Python alapú könyvtár, amely bitcoin kulcsok és tranzakciók kezelését támogatja. sőt, még a script nyelvet is támogatja annyira, hogy kezelni lehet benne a nem szabványos tranzakciókat.

A pycoin könyvtár a Python 2 (2.7.x) és a Python 3 (3.3 utáni verzió) alatt egyaránt használható. Jól kezelhető parancssori segédprogramjai a ku és a tx.

Key Utility (KU)

A ku („key utility”) parancssori segédprogram a kulcs kezelés „svájci bicskája”. Támogatja a BIP32 kulcsokat, a WIF-et, és a bitcoin vagy alt-coin címeket. Az alábbiakban néhány példa szerepel.

Egy BIP32 kulcs létrehozása, a GPG és a */dev/random* alapértelmezett entrópiaforrásainak a használatával:

```
$ ku create

input          : create
network        : Bitcoin
wallet key    :
xprv9s21ZrQH143K3LU5ctPZTBnb9kJA5Su9DcWHvXJemiJBsY7VqXUG7hipgdWaU
                  m2nhnzdvxJf5KJ09vjP2nABX65c5sFsWsV8oXcbpehtJi
public version :
xpub661MyMwAqRbcFpYYiuvZpKjKhnJDZYAkWSY76JvvD7FH4fsG3Nqiov2CfxzxY8
                  DGcpfT56AMFeo8M8KPkFMfLUTvwjwb6WPv8rY65L2q8Hz
tree depth     : 0
fingerprint   : 9d9c6092
parent f'print: 00000000
child index   : 0
chain code    :
80574fb260edaa4905bc86c9a47d30c697c50047ed466c0d4a5167f6821e8f3c
private key    : yes
secret exponent:
112471538590155650688604752840386134637231974546906847202389294096567806844
862
hex           :
f8a8a28b28a916e1043cc0aca52033a18a13cab1638d544006469bc171fddfbe
wif            : L5Z54xi6qJusQT42JHA44mfPVZGjyb4XBRWfxAzUWwRiGx1kV4sP
uncompressed   : 5KhoEavGNNH4GHKoy2Ptu4KfdNp4r56L5B5un8FP6RZnbsz5NmB
public pair x :
764606382405464783648433974782784681018771177678734621270215603682901140160
34
public pair y :
598078796574697741020401202982722077309212917366332477370774067536768257777
01
x as hex      :
a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
y as hex      :
843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcfd625
y parity      : odd
key pair as sec:
03a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
```

```

uncompressed      :
04a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322

843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcfd625
hash160          : 9d9c609247174ae323acfc96c852753fe3c8819d
uncompressed     : 8870d869800c9b91ce1eb460f4c60540f87c15d7
Bitcoin address  : 1FNNRQ5fSv1wBi5gyfVBs2rkNheMGT86sp
uncompressed     : 1DSS5isnH4FsVaLVjeVXewVSpfqktdiQAM

```

Egy BIP32 kulcs létrehozása egy jelmondatból:



A példában szereplő jelmondat túlságosan egyszerű.

```

$ ku p:foo

input          : P:foo
network        : Bitcoin
wallet key    :
xprv9s21ZrQH143K31AgNK5pyVvW23gHnkBq2wh5aEk6g1s496M8ZMjxncCKZKgb5j
               ZoY5eSJMJ2Vbyvi2hbmQnCuHBujZ2WXGTux1X2k9Krdtq
public version :
xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvts
               VYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy
tree depth     : 0
fingerprint   : 5d353a2e
parent f'print : 00000000
child index   : 0
chain code    :
5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc
private key    : yes
secret exponent :
658257305470973057160571604379707902201238642997619089487468358860077939982
75
hex           :
91880b0e3017ba586b735fe7d04f1790f3c46b818a2151fb2def5f14dd2fd9c3
wif           : L26c3H6jEPVsqArlusXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
uncompressed   : 5JvNzA5vXDoKYJdw8SwwLhxUxaWvn9mDea6k1vRPCX7KLUVWa7W
public pair x :
818219827193811040617773492691304190244936166509935893945534043477743931681
91
public pair y :
589942180696054242783207032506897801547850995092776917231263250512004590382
90
x as hex      :
b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
y as hex      :
826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52
y parity      : even
key pair as sec :
02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
uncompressed   :
04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f

826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52
hash160        : 5d353a2ecdb262477172852d57a3f11de0c19286
uncompressed   : e5bd3a7e6cb62b4c820e51200fb1c148d79e67da
Bitcoin address : 19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii

```

```
uncompressed : 1MwkRkogzBRMehBntgcq2aJhXCStJTXHT
```

Információk JSON formában történő lekérdezése:

```
$ ku P:foo -P -j
{
  "yparity": "even",
  "publicpair_y_hex":
"826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
  "privatekey": "no",
  "parentfingerprint": "00000000",
  "treedepth": "0",
  "network": "Bitcoin",
  "btcaddress_uncompressed": "1MwkRkogzBRMehBntgcq2aJhXCStJTXHT",
  "keypair_as_sec_uncompressed":
"04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f826d8b4d
3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
  "publicpair_x_hex":
"b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f",
  "walletkey":
"xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtSVYFvXz2v
PPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy",
  "chaincode":
"5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc",
  "childindex": "0",
  "hash160_uncompressed": "e5bd3a7e6cb62b4c820e51200fb1c148d79e67da",
  "btcaddress": "19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii",
  "fingerprint": "5d353a2e",
  "hash160": "5d353a2ecdb262477172852d57a3f11de0c19286",
  "input": "P:foo",
  "publicpair_x":
"81821982719381104061777349269130419024493616650993589394553404347774393168
191",
  "publicpair_y":
"58994218069605424278320703250689780154785099509277691723126325051200459038
290",
  "keypair_as_sec":
"02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f"
}
```

Publikus BIP32 kulcs:

```
$ ku -w -P P:foo
xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtSVYFvXz2vP
PPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy
```

Egy alkulcs előállítása:

```
$ ku -w -s3/2 P:foo
xprv9wTERTSkjVyJa1v4cUTFMFkWMe5eu8ErbQcs9xajnsUzCBT7ykHAwdxvG3g3f6BFk7ms5h
HBvmbdutNmyg6iogWKxx6mefEw4M8EroLgKj
```

Megerősített alkulcs:

```
$ ku -w -s3/2H P:foo
xprv9wTERTSu5AWGkDeUPmqBcbZWX1xq85ZNX9iQRQW9DXwygFp7iRGJo79dsVctcsCHsnZ3XU3
DhsuaGZbDh8iDkBN45k67UKsJUXM1JfRCdn1
```

WIF:

```
$ ku -W P:foo
L26c3H6jEPVSqArlusXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
```

Cím:

```
$ ku -a P:foo
19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii
```

Több alkulcs előállítása:

```
$ ku P:foo -s 0/0-5 -w
xprv9xWkBDFyBXmZjBG9EiXBpy67KK72fphUp9utJokEBFtjsjiuKUUDF5V3TU8U8cDzytqYnSe
kc8bYuJS8G3bhXxKWB9Ggn2dzLcoJsuEdRK
xprv9xWkBDFyBXmZnZKf3bAGifK593gT7WJZPnYAmvc77gUQVej5QHckc5Adtwxa28ACmANi9Xh
CrRvtFqQcUxt8rUgfz3souMiDdWxDzNQzx
xprv9xWkBDFyBXmZqdXA8y4SWqfBdy71gSW9sjx9JpCiJEiBwSMQyRxan6srXUPBtj3PTxQFkZJ
AiwoUpmvtrxKZu4zfsnr3pqyy2vthpkwuoVq
xprv9xWkBDFyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9c
nGj5P7e8EskpzKL1Y8Gk9aX6QbryA5raK73p
xprv9xWkBDFyBXmZv2q3N6hhZ8DAcEnQDnXML1J62krJAcf7Xb1HJwuW2VMJQrCofY2jtFXdiE
Y8UsRNJfqK6DAdyZXoMvtaLHyWQx3FS4A9zw
xprv9xWkBDFyBXmZw4jEYXUHYc9ft25k9irP87n2RqfJ5bqbjKdT84Mm7Wtc2xmzFuKg7iYf7XF
HKkSsaYKWKJbR54bnyAD9GzjUYbAYTtN4ruo
```

Az alkulcsokhoz tartozó címek előállítása:

```
$ ku P:foo -s 0/0-5 -a
1MrjE78H1R1rqdFrmkjdhnPUDLCJALbv3x
1AnYyVEcuqeovzH96zj1eYKwoWfwte2pxu
1GXR1kZfxE1FcK6ZRD5sqqqs5YfvuzA1Lb
116AXZc4bDVQrqmcinz4aaPdrYqvuiBEK
1Cz2rTLjRM6pMnxPNrRKp9ZSvRtj5dDUML
1WstdwPnU6HEUPme1DQayN9nm6j7nDDEM
```

Az alkulcsokhoz tartozó WIF-ek előállítása:

```
$ ku P:foo -s 0/0-5 -w
L5a4iE5k9gcJKGqX3FWmxzBYQc29PvZ6pgBaePLVqT5YByEnBomx
Kyjgne6GZwPGB6G6kJEhoPbmyjMP7D5d3zRbHVjwcq4iQXD9QqKQ
L4B3ygQxK6zH2NQGxLDee2H9v4Lvwg14cLJW7QwWPzCtKhDWMaQz
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQnmTDMrqemY8UF
L2oD6vA4TUyqPF8QG4vhUFsgwCyuvVFZ3v8SKHYFDwkbM765Nrfd
KzChTbc3kZFxUSJ3Kt54cxsogeFAD9CCM4zGB22si8nfKcThQn8C
```

Annak az ellenőrzése, hogy minden működik, ha egy BIP32 stringet választunk (ami a 0/3 alkulcsnak felel meg):

```
$ ku -w
xprv9xWkBDFyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9c
nGj5P7e8EskpzKL1Y8Gk9aX6QbryA5raK73p
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQnmTDMrqemY8UF
$ ku -a
xprv9xWkBDFyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9c
nGj5P7e8EskpzKL1Y8Gk9aX6QbryA5raK73p
116AXZc4bDVQrqmcinz4aaPdrYqvuiBEK
```

Igen ez ismerősnek tűnik.

A titkos kitevőből:

```
$ ku 1

input          : 1
network        : Bitcoin
secret exponent : 1
hex            : 1
wif            : KwDiBf89QgGbjEhKnhXJuH7LrciVrZi3qYjgd9M7rFU73sVHnoWn
uncompressed   : 5HpHagT65TzzG1PH3CSu63k8DbpvD8s5ip4nEB3kEsreAnchuDf
public pair x  :
550662630222773436695787188951685343262506034537775941755001873603891167292
40
public pair y  :
326705100207588169780830851305070431844712733806592432759389043357573374824
24
x as hex       :
79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
y as hex       :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity       : even
key pair as sec :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
uncompressed   :
0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798

483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160         : 751e76e8199196d454941c45d1b3a323f1433bd6
uncompressed   : 91b24bf9f5288532960ac687abb035127b1d28a5
Bitcoin address : 1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAMH
uncompressed   : 1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZm
```

Litecoin verzió:

```
$ ku -nL 1

input          : 1
network        : Litecoin
secret exponent : 1
hex            : 1
wif: T33ydQRKp4FCW5LCLLUB7deioUMoveiwekdwUwyfRDeGZm76aUjV
uncompressed   : 6u823ozcyt2rjPH8Z2ErsSXJB5PPQwK7VVTwN4mxLBFrao69XQ
public pair x  :
550662630222773436695787188951685343262506034537775941755001873603891167292
40
public pair y  :
326705100207588169780830851305070431844712733806592432759389043357573374824
24
x as hex       :
79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
y as hex       :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity       : even
key pair as sec :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
uncompressed   :
0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
```

```

483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160           : 751e76e8199196d454941c45d1b3a323f1433bd6
uncompressed       : 91b24bf9f5288532960ac687abb035127b1d28a5
Litecoin address  : LVuDpNCSSj6pQ7t9Pv6d6sUkLkoqDEVUnJ
uncompressed       : LYWKqJhtPeGyBAw7WC8R3F7ovxtzAiubdM

```

Dogecoin WIF:

```

$ ku -nD -W 1
QNcdLVw8fHkixm6NNyN6nVwxKek4u7qrrioRbQmjxac5TVoTtZuot

```

Nyilvános kulcspróból (a Testneten):

```

$ ku -nT
550662630222773436695787188951685343262506034537775941755001873603891167292
40,even

input           :
550662630222773436695787188951685343262506034537775941755001873603
                     89116729240,even
network         : Bitcoin testnet
public pair x   :
550662630222773436695787188951685343262506034537775941755001873603891167292
40
public pair y   :
326705100207588169780830851305070431844712733806592432759389043357573374824
24
x as hex        :
79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
y as hex        :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity        : even
key pair as sec :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
uncompressed    :
0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798

483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160          : 751e76e8199196d454941c45d1b3a323f1433bd6
uncompressed      : 91b24bf9f5288532960ac687abb035127b1d28a5
Bitcoin testnet address : mrCDrCybB6J1vRfbwM5hemdJz73FwDBC8r
uncompressed      : mtoKs9V381UAhUiA3d7Vb9GNak8Qvmcsme

```

hash160-ból:

```

$ ku 751e76e8199196d454941c45d1b3a323f1433bd6

```

```

input           : 751e76e8199196d454941c45d1b3a323f1433bd6
network         : Bitcoin
hash160          : 751e76e8199196d454941c45d1b3a323f1433bd6
Bitcoin address : 1BgGZ9tcN4rm9KBzDn7KprQz87Sz26SAMH

```

Mint Dogecoin cím:

```

$ ku -nD 751e76e8199196d454941c45d1b3a323f1433bd6

```

```

input           : 751e76e8199196d454941c45d1b3a323f1433bd6
network         : Dogecoin

```

```
hash160      : 751e76e8199196d454941c45d1b3a323f1433bd6
Dogecoin address : DFpN6QqFFUm3gKNaxN6tNcab1FArL9cZLE
```

Transaction Utility (TX)

A `tx` parancssori segédprogram olvasható formátumban jeleníti meg a tranzakciókat, letölți őket a pycoin cache-éből vagy web szervizekről (jelenleg a blockchain.info, a blockr.io, és a biteeasy.com van támogatva), összefésüli őket, bemenetek és kimenetek hozzáadását vagy törlését teszi lehetővé, és tranzakciók aláírására képes.

Néhány példa következik:

Nézzük meg a híres [PIZZA] „pizza” tranzakciót:

```
$ tx 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
figyelmeztetés: állítsák be a PYCOIN_CACHE_DIR környezeti változót a
következőképpen, ha cache-elti szeretnék a web szervizekről letöltött
tranzakciókat: PYCOIN_CACHE_DIR=~/.pycoin_cache
figyelmeztetés: a get_tx nem talált szolgáltatót, állítsa be a következő
környezeti változót:
PYCOIN_SERVICE_PROVIDERS=BLOCKR_IO:BLOCKCHAIN_INFO:BITEASY:BLOCKEXPLORER
használata: tx [-h] [-t TRANSACTION_VERSION] [-l LOCK_TIME] [-n NETWORK] [-a]
           [-i address] [-f path-to-private-keys] [-g GPG_ARGUMENT]
           [--remove-tx-in tx_in_index_to_delete]
           [--remove-tx-out tx_out_index_to_delete] [-F transaction-fee] [-u]
           [-b BITCOIND_URL] [-o path-to-output-file]
           argument [argument ...]
tx: error: can't find Tx with id
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
```

Hoppá! Nem állítottuk be a web szervizeket. Tegyük ezt most meg:

```
$ PYCOIN_CACHE_DIR=~/.pycoin_cache
$ PYCOIN_SERVICE_PROVIDERS=BLOCKR_IO:BLOCKCHAIN_INFO:BITEASY:BLOCKEXPLORER
$ export PYCOIN_CACHE_DIR PYCOIN_SERVICE_PROVIDERS
```

Azért nem történik ez meg automatikusan, nehogy a parancssori eszköz véletlenül információkat szívárogtasson ki arról, hogy mely tranzakciók érdekelnek minket. Ha ez nem szempont a számunkra, akkor ezeket a sorokat betehetjük a *.profile* file-ba:

Próbáljuk meg ismét:

```
$ tx 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
Version: 1 tx hash
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a 159 bytes
TxIn count: 1; TxOut count: 1
Lock time: 0 (valid anytime)
Input:
  0:                               (unknown) from
1e133f7de73ac7d074e2746a3d6717dfc99ecaa8e9f9fade2cb8b0b20a5e0441:0
Output:
  0: 1CZDM6oTttND6WPdt3D6bydo7DYKzd9Qik receives 10000000.00000 mBTC
Total output 10000000.00000 mBTC
including unspents in hex dump since transaction not fully signed
```

```
010000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e0
00000004a493046022100a7f26eda87493199c90f87f01ff1ffc76bcd058fe16137e0e63fd
b6a35c2d78022100a61e919238eb73f07c8f209504c84b80f03e30ed8169edd44f80ed17dd
f451901fffffffff010010a5d4e80000001976a9147ec1003336542cae8bded8909cdd6b5e48
ba0ab688ac00000000
```

```
** can't validate transaction as source transactions missing
```

Az utolsó sor azért jelenik meg, mert a tranzakciók aláírásainak az ellenőrzéséhez szükség van a forrás tranzakciókra is. Egy a hozzáadásával tudjuk a tranzakciót a forrás tranzakciókkal kiegészíteni:

```
$ tx -a 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
warning: transaction fees recommendations casually calculated and estimates
may be incorrect
warning: transaction fee lower than (casually calculated) expected value of
0.1 mBTC, transaction might not propagate
Version: 1 tx hash
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a 159 bytes
TxIn count: 1; TxOut count: 1
Lock time: 0 (valid anytime)
Input:
 0: 17WFx2GQZUmh6Up2NDNCEdK3deYomdNCfk from
1e133f7de73ac7d074e2746a3d6717dfc99ecaa8e9f9fade2cb8b0b20a5e0441:0
10000000.00000 mBTC sig ok
Output:
 0: 1CZDM6oTttND6WPdt3D6bydo7DYKzd9Qik receives 10000000.00000 mBTC
Total input 10000000.00000 mBTC
Total output 10000000.00000 mBTC
Total fees 0.00000 mBTC

010000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e0
00000004a493046022100a7f26eda87493199c90f87f01ff1ffc76bcd058fe16137e0e63fd
b6a35c2d78022100a61e919238eb73f07c8f209504c84b80f03e30ed8169edd44f80ed17dd
f451901fffffffff010010a5d4e80000001976a9147ec1003336542cae8bded8909cdd6b5e48
ba0ab688ac00000000

all incoming transaction values validated
```

Nézzük meg most egy adott cím elköltetlen kimeneteit (UTXO). A #1 blokkban látható a 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX coinbase tranzakció. A fetch_unspent használatával keressük meg, hogy összesen hány érme van ezen a címen:

```
$ fetch_unspent 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX
a3a6f902a51a2cbebede144e48a88c05e608c2cce28024041a5b9874013a1e2a/0/76a91411
9b098e2e980a229e139a9ed01a469e518e6f2688ac/333000
cea36d008badf5c7866894b191d3239de9582d89b6b452b596f1f1b76347f8cb/31/76a9141
19b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
065ef6b1463f552f675622a5d1fd2c08d6324b4402049f68e767a719e2049e8d/86/76a9141
19b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
a66dddd42f9f2491d3c336ce5527d45cc5c2163aaed3158f81dc054447f447a2/0/76a91411
9b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
ffd901679de65d4398de90cefef68d2c3ef073c41f7e8dbec2fb5cd75fe71dfe7/0/76a91411
9b098e2e980a229e139a9ed01a469e518e6f2688ac/100
d658ab87cc053b8dbcfd4aa2717fd23cc3edfe90ec75351fadd6a0f7993b461d/5/76a91411
9b098e2e980a229e139a9ed01a469e518e6f2688ac/911
36ebe0ca3237002acb12e1474a3859bde0ac84b419ec4ae373e63363ebef731c/1/76a91411
9b098e2e980a229e139a9ed01a469e518e6f2688ac/100000
```

fd87f9adecbb17f4ebb1673da76ff48ad29e64b7afa02fda0f2c14e43d220fe24/0/76a91411
9b098e2e980a229e139a9ed01a469e518e6f2688ac/1
dfdf0b375a987f17056e5e919ee6eadd87dad36c09c4016d4a03cea15e5c05e3/1/76a91411
9b098e2e980a229e139a9ed01a469e518e6f2688ac/1337
cb2679bf0a557b2dc0d8a6116822f3fcbe281ca3f3e18d3855aa7ea378fa373/0/76a91411
9b098e2e980a229e139a9ed01a469e518e6f2688ac/1337
d6be34ccf6edddc3cf69842dce99fe503bf632ba2c2adb0f95c63f6706ae0c52/1/76a91411
9b098e2e980a229e139a9ed01a469e518e6f2688ac/2000000

0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098/0/410496b5
38e853519c726a2c91e61ec11600ae1390813a627c66fb8be7947be63c52da7589379515d4e
0a604f8141781e62294721166bf621e73a82cbf2342c858eeac/5000000000

D. függelék: Bitcoin Explorer (bx) parancsok

Használata: bx COMMAND [--help]

Info: A bx parancsok a következők:

```
address-decode
address-embed
address-encode
address-validate
base16-decode
base16-encode
base58-decode
base58-encode
base58check-decode
base58check-encode
base64-decode
base64-encode
bitcoin160
bitcoin256
btc-to-satoshi
ec-add
ec-add-secrets
ec-multiply
ec-multiply-secrets
ec-new
ec-to-address
ec-to-public
ec-to-wif
fetch-balance
fetch-header
fetch-height
fetch-history
fetch-stealth
fetch-tx
fetch-tx-index
hd-new
hd-private
hd-public
hd-to-address
hd-to-ec
hd-to-public
hd-to-wif
help
input-set
input-sign
input-validate
message-sign
message-validate
mnemonic-decode
mnemonic-encode
ripemd160
satoshi-to-btc
script-decode
script-encode
script-to-address
seed
send-tx
send-tx-node
send-tx-p2p
```

```
settings
sha160
sha256
sha512
stealth-decode
stealth-encode
stealth-public
stealth-secret
stealth-shared
tx-decode
tx-encode
uri-decode
uri-encode
validate-tx
watch-address
wif-to-ec
wif-to-public
wrap-decode
wrap-encode
```

További információért menjen a [Bitcoin Explorer honlapjára](#) és a [Bitcoin Explorer felhasználói dokumentációjához](#).

Példa a bx parancsok használatára

Nézzük néhány példát arra, hogyan lehet a Bitcoin Explorer parancsaival kulcsokat és címeket kezelni:

Állítsunk elő egy véletlen „magot” a `seed` parancs segítségével, amely az operációs rendszer véletlenszám generátorára épül. A magot az `ec-new` parancsnak átadva hozzunk létre egy új titkos kulcsot. A szabványos kimenetet a `private_key` file-ba irányítjuk át:

```
$ bx seed | bx ec-new > private_key
$ cat private_key
73096ed11ab9f1db6135857958ece7d73ea7c30862145bcc4bbc7649075de474
```

És most állítsunk elő ebből a titkos kulcsból egy nyilvános kulcsot az `ec-to-public` parancccsal. A `private_key` file-t a szabványos bemenetre irányítjuk át, és a parancs szabványos kimenetét a `public_key` új file-ba mentjük el:

```
$ bx ec-to-public < private_key > public_key
$ cat public_key
02fcfa46a6006a62dfdd2dbb2149359d0d97a04f430f12a7626dd409256c12be500
```

A `public_key` nyilvános kulcsot címként tudjuk megjeleníteni az `ec-to-address` parancccsal. A `public_key` file-t a szabványos bemenetre irányítjuk át:

```
$ bx ec-to-address < public_key
17re1S4Q8ZHCP8Kw7xQad1Lr6XUzWUnkG
```

Az így előállított kulcsok 0-ik típusú, nem determinisztikus kulcsok. Ez azt jelenti, hogy a minden egyes kulcs egy saját, független magból származik. A Bitcoin Explorerrel a BIP0032-nek megfelelő, determinisztikus kulcsok is létrehozhatók. Ebben az esetben a magból egy „mesterkulcs” jön létre, majd ennek a kiterjesztése révén állnak elő a alkulcsok, és jön létre a 2-es típusú, determinisztikus pénztárca.

Először a `seed` és a `hd-new` parancsokkal egy mester kulcsot állítunk elő, melyet a további kulcshierarchia vezetésére fogunk használni.

```
$ bx seed > seed
$ cat seed
eb68ee9f3df6bd4441a9feadec179ff1

$ bx hd-new < seed > master
$ cat master
xprv9s21ZrQH143K2BEhMYpNQoUvAgjArAVaZaCTgsaGe6LsAnwubeiTcDzd23mAoyizm9cAp
e51gNfLMkBqkYoWWMCRwfJk8RwF1SVEpAQ
```

Most pedig a `hd-private` parancssal előállítjuk a megerősített „számla” kulcsot és két, a számlán belüli titkos kulcsot.

```
$ bx hd-private --hard < master > account
$ cat account
xprv9vkDLt81dTkJwHB8fsVB5QK8cGnzveChzSrtCfvu3aMWvQaThp59ueufuyQ8Qi3qpjk4aKs
bmbfxwcfg8PYbgoR2NWHeLyvg4DhoEE68A1n

$ bx hd-private --index 0 < account
xprv9xHfb6w1vX9xgZyPNXVgAhPxSSEkeRcPHEUV5iJcVEsuUEACvR3NRY3fpGhcnBiDbvG4Lgn
dirDsiale9F3DWPKX7Tp1V1u97HKG1FJwUpU

$ bx hd-private --index 1 < account
xprv9xHfb6w1vX9xjc8XbN4GN86jzNAZ6xHEqYxzBLB4fzHFd6VqCLPGRZFsdjsuMVERadbgDbz
iCRJru9n6tzEWrASVpEdrZrFidt1Rdfn4yA3
```

Ezt követően a `hd-public` parancssal előállítjuk a titkos kulcsoknak megfelelő két darab nyilvános kulcsot.

```
$ bx hd-public --index 0 < account
xpub6BH1zcTuktifu43rUZ2gXqLgzu5F3tLEetQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYm
qL2eYPFJ3vezHz5wzaSW4FiGrseNDR4LKqTy

$ bx hd-public --index 1 < account
xpub6BH1zcTuktifx6CzhPgjG3UYQ13WR16CmtbPiagEKpEVtpyjshWyMaMV1cn7nUPUkgQHPV
XJVqsrA8xWbGQDhohEcDFTEYMyYzwRD7Juf8
```

A titkos kulcsokhoz tartozó nyilvános kulcsok a `hd-to-public` parancssal is előállíthatók.

```
$ bx hd-private --index 0 < account | bx hd-to-public
xpub6BH1zcTuktifu43rUZ2gXqLgzu5F3tLEetQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYm
qL2eYPFJ3vezHz5wzaSW4FiGrseNDR4LKqTy

$ bx hd-private --index 1 < account | bx hd-to-public
xpub6BH1zcTuktifx6CzhPgjG3UYQ13WR16CmtbPiagEKpEVtpyjshWyMaMV1cn7nUPUkgQHPV
XJVqsrA8xWbGQDhohEcDFTEYMyYzwRD7Juf8
```

Gyakorlatilag korlátlan számú kulcsot tudunk létrehozni a determinisztikus láncban, és ezek mindegyike ugyanabból a magból származik. Számos pénztárca használja ezt a módszert, mert az így előállított címek mentéséhez és visszaállításához csak egyetlen egy mag értékre van szükség. Ezt könnyebb menteni, mint a véletlenszerű kulcsokat tartalmazó pénztárcát, melyet minden egyes alkalommal menteni kell, amikor új kulcsok jöttek létre benne.

A mag a `mnemonic-encode` parancssal állítható elő.

```
$ bx hd-mnemonic < seed > words
adore repeat vision worst especially veil inch woman cast recall dwell
appreciate
```

Ezt követően a mag a `mnemonic-decode` parancsával dekódolható.

```
$ bx mnemonic-decode < words
eb68ee9f3df6bd4441a9feadec179ff1
```

A mnemonikus kódolás megkönnyíti a mag leírását és támaszt jelent az emlékezet számára is.

Kolofon

A *Bitcoin fejlesztőknek* fedőlapján látható állat a levélvágó hangya (*Atta colombica*). A levélvágó hangyák trópusi, gombatermesztő hangyák, melyek tömegesen Dél- és Közép-Amerikában, Mexikóban és az Egyesült Államok déli részén fordulnak elő. Bolygónkon az emberek mellett a levélvágó hangyák alkotják a legnagyobb és legbonyolultabb társadalmakat. Nevüket annak köszönhetik, hogy leveleket rágunk, melyek tápanyagként szolgának a gombaültetvények számára.

A hím- és nőnemű szárnyas példányok a *revoada*, azaz a párzási repülés során tömegesen hagyják el a bolyt. A nőstények több hímmel is párosodnak, hogy birtokába jussanak az új boly létesítéséhez szükséges 300 millió ivarsejtnek. A nőstények a szájnyílásukban lévő tasakban az új gombatelep létesítésére szolgáló micéliumot tárolják. A földreszállás után a nőstények elveszítik a szárnyukat, és egy föld alatti üregben saját bolyt hoznak létre. Az új királynők közül csupán 2.5 %-nak sikerül hosszabb ideig fennálló bolyt létrehoznia.

A hosszabb ideje fennálló bolyokban a hangyák a méretük alapján kasztokat alkotnak. Az egyes kasztok által végzett feladatak különbözők. Általában négy kasztot különböztethetünk meg. A legkisebb dolgozók feladata a lárvák és a gombaültetvények gondozása. Egy kicsit nagyobb méretű a következő kaszt. Az ő feladatak a boly védelme és az ellenség megtámadása. A közepes méretű hangyák a gyűjtögetők, akik levágják a leveleket és a levéldarabkákat visszaszállítják a bolyba. Végül a legnagyobb méretű munkások alkotják a katonaságot, amely megvédi a bolyt a behatolóktól. Az újabb kutatások szerint az ő feladatak a gyűjtögetéskor használt útvonalak tisztán tartása és a nagyobb méretű darabok bolyba szállítása is.

Az O'Reilly kiadványok fedőlapjain szereplő állatok közül sok a veszélyeztetettek közé tartozik. Fontos, hogy ezek fennmaradjanak. Megtudhatja, hogy hogyan segíthet, ha felkeresi az animals.oreilly.com web helyet.

A fedőlapon lévő kép az *Insects Abroad* kiadványból származik. A fedőlapon az URW Typewriter és a Guardian Sans betűtípus használtuk. A főszöveg betűtípusa Adobe Minion Pro; a címek betűtípusa Adobe Myriad Condensed; a forráskód betűtípusa a Dalton Maag tervezte Ubuntu Mono.