

# Mastering Bitcoin

## Prefață

### Scrierea Cărții Bitcoin

Am dat prima dată peste bitcoin la mijlocul anului 2011. Reacția mea imediată a fost mai mult sau mai puțin "Pfft! Bani pentru tocilari!" și am ignorat bitcoin încă șase luni, nereușind să-i înțeleg importanța. Aceasta este o reacție pe care am văzut-o repetată printre mulți dintre cei mai deștepti oameni pe care îi cunosc, ceea ce îmi oferă oarecare consolare. A doua oară când am dat peste bitcoin, într-o discuție pe email, am decis să citesc raportul scris de Satoshi Nakamoto pentru a studia sursa autoritară și a vedea despre ce este vorba. Îmi amintesc încă momentul în care am terminat de citit acele nouă pagini, când mi-am dat seama că bitcoin nu era pur și simplu o monedă digitală, ci o rețea de încredere care ar putea oferi baza pentru mult mai mult decât doar monede. Conștientizarea faptului că "aceasta nu este bani, este o rețea de încredere descentralizată", m-a pornit într-o călătorie de patru luni pentru a devora fiecare bucătă de informații despre bitcoin pe care am putut-o găsi. Am devenit obsedat și încântat, petrecând 12 sau mai multe ore în fiecare zi lipit de un ecran, citind, scriind, codând și învățând cât am putut. Am ieșit din această stare de fugă, cu peste 20 de kilograme mai ușor din lipsa unor mese consistente, hotărât să mă dedic lucrului cu bitcoin.

Doi ani mai târziu, după ce am creat o serie de startup-uri mici pentru a explora diverse servicii și produse legate de bitcoin, am decis că este timpul să scriu prima mea carte. Bitcoin a fost subiectul care mă condusese într-o frenzie a creativității și mi-a consumat gândurile; a fost cea mai interesantă tehnologie pe care am întâlnit-o de la apariția internetului. Acum era timpul să împărtășesc pasiunea mea pentru această tehnologie uimitoare cu un public mai larg.

## Public Țintă

Această carte este destinată mai ales programatorilor. Dacă puteți utiliza un limbaj de programare, această carte vă va învăța cum funcționează monedele criptografice, cum să le utilizați și cum să dezvoltați software care funcționează cu ele. Primele câteva capitole sunt, de asemenea, o introducere în profunzime a bitcoin pentru non-programatori - cei care încearcă să înțeleagă funcționarea bitcoin și a criptomonedelor.

## De Ce Sunt Gândaci pe Copertă?

Furnica tăietoare de frunze este o specie care prezintă un comportament extrem de complex într-un superorganism de colonie, dar fiecare furnică individuală operează conform unui set de reguli simple determinate de interacțiunea socială și schimb de arome chimice (feromoni). Conform Wikipedia: "Alături de oameni, furnicile tăietoare de frunze formează cele mai mari și mai complexe societăți de animale de pe Pământ". Furnicile tăietoare de frunze nu mănâncă de fapt frunze, ci le folosesc pentru a dezvolta o ciupercă, care este sursa principală de hrănă pentru colonie. What? Aceste furnici practică agricultură!

Deși furnicile formează o societate bazată pe caste și au o regină pentru producerea puilor, nu există nici o autoritate centrală sau lider într-o colonie de furnici. Comportamentul extrem de intelligent și sofisticat arătat de o colonie de milioane de membri este o proprietate rezultată din interacțiunea indivizilor într-o rețea socială.

Natura demonstrează că sistemele descentralizate pot fi reziliente și pot produce o complexitate și sofisticare incredibilă fără a fi nevoie de o autoritate centrală, ierarhie, sau piese complexe.

Bitcoin este o rețea de încredere descentralizată extrem de sofisticată care poate susține numeroase procese financiare. Cu toate acestea, fiecare nod din rețeaua bitcoin respectă câteva reguli matematice simple. Interacțiunea dintre mai multe noduri este ceea ce duce la apariția unui comportament sofisticat, și nu o complexitate inherentă sau încredere într-un singur nod. Ca o colonie de furnici, rețeaua bitcoin este o rețea rezilientă formată din noduri simple, care urmează reguli simple, care împreună pot face lucruri uimitoare fără nicio coordonare centrală.

## Convenții Utilizate în Această Carte

În această carte sunt utilizate următoarele convenții tipografice:

### ***Italic***

Indică termeni noi, URL-uri, adrese de email, nume de fișiere și extensii de fișiere.

### **Lățime constantă**

folosit pentru listări de cod, precum și în cadrul paragrafelor pentru a face referiri la elemente de cod, cum ar fi variabile sau nume de funcții, baze de date, tipuri de date, variabile de mediu, declarații, și cuvintele cheie.

### **Lățime constantă bold**

Indică comenzi sau alt text care ar trebui să fie introdus așa cum este de către utilizator.

### ***Lățime constantă italic***

Arată un text care ar trebui să fie înlocuite cu valorile furnizate de utilizator sau prin valori determinate de context.

**TIP** Această pictogramă semnifică un sfat sau o sugestie.

**NOTE** Această pictogramă semnifică o notă generală.

**WARNING** Această pictogramă indică un avertisment sau precauție.

## Exemple de Cod

Exemplile sunt ilustrate în Python, C++ și folosind linia de comandă a unui sistem de operare asemănător Unix, cum ar fi Linux sau macOS. Toate fragmentele de cod sunt disponibile în repo-ul GitHub (<https://github.com/bitcoinbook/bitcoinbook>) în subdirectorul *code*. Bifurcați codul cărții, încercați exemplele de cod sau trimiteți corecții prin GitHub.

Toate fragmentele de cod pot fi reproduse pe majoritatea sistemelor de operare cu o instalare minimă de compilatoare și interprete pentru limbajele corespunzătoare. Acolo unde este necesar, oferim instrucțiuni de instalare și exemple pas cu pas ale rezultatului acestor instrucțiuni.

Unele dintre fragmentele de cod au fost reformatate pentru tipărire. În toate aceste cazuri, liniile au fost împărțite cu un caracter backslash (\), urmat de un caracter line-nouă. Când transcrieți exemplele, eliminați cele două caractere și alăturați-le din nou și ar trebui să vedeți rezultate identice, aşa cum se arată în exemplu.

Toate fragmentele de cod folosesc valori reale și calcule acolo unde este posibil, astfel încât să puteți construi de la un exemplu la altul și să vedeți aceleași rezultate pentru orice cod scrieți pentru a calcula aceleași valori. De exemplu, cheile private și cheile publice corespunzătoare și adresele sunt reale. Tranzacțiile, blocurile, și refrințele la lanțul-de-blocuri toate au fost introduse în lanțul-de-blocuri real bitcoin și fac parte din registrul public, pentru a le putea examina pe orice sistem bitcoin.

## Utilizarea Exemplelor de Cod

Această carte este aici pentru a vă ajuta să vă duceți proiectul la bun sfârșit. În general, exemplele de cod oferite cu această carte se pot folosi în programele și documentația dumneavoastră. Nu este nevoie să ne contactați pentru permisiune dacă nu reproduceți o parte semnificativă a codului. De exemplu, scriind un program care utilizează câteva bucăți de cod din această carte nu este nevoie de permisiunea noastră. Vanzarea sau distribuirea CD-ROM-ului cu exemple de la O'Reilly are nevoie de permisiune. Răspunzând unei întrebări, citând această carte și citând exemplul de cod nu are nevoie de permisiune. Încorporarea unei cantăți semnificative din exemplele de cod din această carte în documentația produsului dumneavoastră are nevoie în schimb de nevoie permisiune.

Apreciem, dar nu necesităm necesită, atribuire. De obicei, o atribuție include titlul, autorul, editorul și ISBN-ul. De exemplu: *“Mastering Bitcoin* de Andreas M. Antonopoulos (O'Reilly). Copyright 2017 Andreas M. Antonopoulos, 978-1-491-95438-6.”

Unele ediții ale acestei cărți sunt oferite sub licență open source, cum ar fi [CC-BY-NC](#), caz în care se aplică condițiile acestei licențe.

Dacă considerați că utilizarea dumneavoastră a exemplelor de cod nu se încadrează în utilizarea corectă sau permisiunea dată mai sus, nu ezitați să ne contactați la [permissions@oreilly.com](mailto:permissions@oreilly.com).

## Adresele și Tranzacțiile Bitcoin din Această Carte

Adresele bitcoin, tranzacțiile, cheile, codurile QR și datele lanțului-de-blocuri utilizate în această carte sunt, în mare parte, reale. Asta înseamnă că puteți răsfoi lanțul-de-blocuri, să vedeți tranzacțiile oferite ca exemple, să le preluăți cu propriile scripturi sau programe etc.

Cu toate acestea, rețineți că cheile private utilizate pentru construirea adreselor sunt fie tipărite în această carte, fie au fost ”arse”. Asta înseamnă că dacă trimiteți bani la oricare dintre aceste adrese, banii vor fi pierduți pentru totdeauna, sau în unele cazuri, toți cei care pot citi cartea îi pot lua folosind cheile private tipărite aici.

**WARNING**

NU TRIMITI BANI LA NICIUNA DINTRE ADRESELE DIN ACEASTĂ CARTE. Banii dumneavoastră vor fi luați de un alt cititor sau îi veți pierde pentru totdeauna.

## O'Reilly Safari

**NOTE**

*Safari* (fostă Safari Books Online) este o platformă de instruire și de referință pentru întreprinderi, guverne, educatori și persoane fizice.

Membrii au acces la mii de cărți, videoclipuri de instruire, căi de învățare, tutoriale interactive și playlists-uri întreținute de la peste 250 de editori, printre care O'Reilly Media, Harvard Business Review, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Adobe, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett și Curs Tehnologie, printre altele.

Pentru mai multe informații, vă rugăm să vizitați <http://oreilly.com/safari>.

## Cum să Ne Contactați

Vă rugăm să adresați editorului comentarii și întrebări referitoare la această carte:

O'Reilly Media, Inc.  
1005 Gravenstein Highway North  
Sebastopol, CA 95472  
800-998-9938 (in the United States or Canada)  
707-829-0515 (international or local)  
707-829-0104 (fax)

Pentru a comenta sau a pune întrebări tehnice despre această carte, trimiteți un e-mail la [bookquestions@oreilly.com](mailto:bookquestions@oreilly.com).

Pentru mai multe informații despre cărțile noastre, cursuri, conferințe, și știri, vizitați site-ul nostru la <http://www.oreilly.com>.

Ne puteți găsi pe Facebook: <http://facebook.com/oreilly>

Urmăriți-ne pe Twitter: <http://twitter.com/oreillymedia>

Urmăriți-ne pe YouTube: <http://www.youtube.com/oreillymedia>

## Contactarea Autorului

Mă puteți contacta, Andreas M. Antonopoulos, pe site-ul meu personal: <https://antonopoulos.com/>

Informații despre *Mastering Bitcoin*, precum și Ediția Open și traduceri sunt disponibile pe: <https://bitcoinbook.info/>

Urmăriți-mă pe Facebook: <https://facebook.com/AndreasMAntonopoulos>

Urmăriți-mă pe Twitter: <https://twitter.com/aantonop>

Urmăriți-mă pe Linkedin: <https://linkedin.com/company/aantonop>

Mulțumesc tuturor patronilor mei care îmi susțin munca prin donații lunare. Puteți să urmăriți pagina Patreon aici: <https://patreon.com/aantonop>

## Mulțumiri

Această carte reprezintă eforturile și contribuțiile multor persoane. Sunt recunoscător pentru tot ajutorul pe care l-am primit de la prietenii, colegi și chiar persoane complet necunoscute, care mi s-au alăturat în acest efort de a scrie această carte tehnică despre criptomonede și bitcoin.

Este imposibil să se facă o distincție între tehnologia bitcoin și comunitatea bitcoin, iar această carte este la fel de mult un produs al comunității cât și o carte despre tehnologie. Munca mea la această carte a fost încurajată, aclamată, susținută și răsplătită de către întreaga comunitate bitcoin încă de la început și până la sfârșit. Mai mult decât orice, această carte mi-a permis să fac parte dintr-o comunitate minunată timp de doi ani și nu pot mulțumi îndeajuns pentru acceptarea mea în această comunitate. Sunt prea multe persoane pentru a menționa numele acestora individual, oameni pe care i-am întâlnit la conferințe, evenimente, seminarii, meetups, întâlniri cu pizza, și adunări private mici, precum și mai mulți sunt cei care au comunicat cu mine pe Twitter, pe Reddit, pe bitcointalk.org, și pe GitHub și care au avut un impact asupra acestei cărți. Fiecare idee, analogie, întrebare, răspuns, și explicație pe care le veți găsi în această carte au fost la un moment dat inspirate, testate, sau îmbunătățite prin interacțiunile mele cu comunitatea. Vă mulțumesc tuturor pentru sprijinul acordat; fără voi această carte nu s-ar fi întâmplat. Sunt pentru totdeauna recunoscător.

Călătoria spre a deveni autor începe cu mult înainte de prima carte, desigur. Prima mea limbă (și școala) a fost limba greacă, așa că a trebuit să urmez un curs de redactare în limba engleză în primul meu an de universitate. Îi datorez mulțumiri Dianei Kordas, profesoara mea de scriere engleză, care m-a ajutat să îmi construiesc încredere și abilități în acel an. Mai târziu, ca profesionist, mi-am dezvoltat abilitățile de scriere tehnică pe tema centrelor de date, scriind pentru revista *Network World*. Îi datorez mulțumiri lui John Dix și John Gallant, care mi-au oferit primul meu post de scriitor în calitate de columnist la *Network World* și editorului meu Michael Cooney și colegii mele Johna Till Johnson, care au editat coloanele mele și le-au făcut potrivite pentru publicare. Scrierea a 500 de cuvinte pe săptămână timp de patru ani mi-a oferit suficientă experiență pentru a avea în vedere în cele din urmă să devin autor.

Mulțumesc și celor care m-au susținut când am trimis propunerea mea de carte lui O'Reilly, oferind referințe și revizuind propunerea. Mai exact, mulțumiri lui John Gallant, Gregory Ness, Richard Stiennon, Joel Snyder, Adam B. Levine, Sandra Gittlen, John Dix, Johna Till Johnson, Roger Ver și Jon Matonis. Mulțumiri speciale lui Richard Kagan și Tymon Mattoszko, care au examinat versiunile anterioare ale propunerii și lui Matthew Taylor, care a copi-editat propunerea.

Mulțumesc lui Cricket Liu, autorul titlului O'Reilly, *DNS și BIND*, care m-a prezentat la O'Reilly. Mulțumesc și lui Michael Loukides și Allyson MacDonald la O'Reilly, care au lucrat luni întregi pentru a ajuta ca această carte să existe. Allyson a avut mai multă răbdare când au lipsit termenele

și livrările au întârziat pe măsură ce viața a intervenit în programul nostru planificat. Pentru a doua ediție, îi mulțumesc lui Timothy McGovern pentru ghidarea procesului, lui Kim Cofer pentru editarea cu răbdare și Rebecca Panzer pentru ilustrarea multor diagrame noi.

Primele câteva proiecte ale primelor capitole au fost cele mai grele, deoarece bitcoin este un subiect dificil de dezvăluit. De fiecare dată când am tras un fir al tehnologiei bitcoin, a trebuit să mă ocup de toate. M-am blocat în repetate rânduri și un pic deznădăjduit, în timp ce m-am străduit să fac subiectul ușor de înțeles și să creez o narățiune în jurul unui subiect tehnic atât de dens. În cele din urmă, am decis să spun povestea bitcoin prin poveștile oamenilor care folosesc bitcoin, iar întreaga carte a devenit mult mai ușor de scris. Datorez prietenului și mentorului meu, Richard Kagan, care m-a ajutat să deslușesc povestea și să trec peste momentele de blocare a scriitorului. Îi mulțumesc Pamelei Morgan, care a examinat proiectele anterioare ale fiecărui capitol din prima și a doua ediție a cărții și a pus întrebările grele pentru a le îmbunătăți. De asemenea, mulțumesc dezvoltatorilor grupului Meetup din San Francisco Bitcoin Developers Meetup, precum și Taariq Lewis și Denise Terry pentru a ajutorul la testarea materialului timpuriu. Mulțumesc și lui Andrew Naugler pentru design infografic.

În timpul scrierii cărții, am pus la dispoziție proiecte timpurii pe GitHub și am invitat comentarii publice. Mai mult de o sută de comentarii, sugestii, corecții și contribuții au fost transmise ca răspuns. Aceste contribuții sunt recunoscute în mod explicit, cu mulțumirile mele, în [\[github\\_contrib\]](#). Mai ales, mulțumirile mele sincere redactorilor voluntari GitHub Ming T. Nguyen (ediția I) și Will Binns (ediția a II-a), care au lucrat neobosit pentru a curata, gestiona și rezolva Pull Request-urile, a emite rapoarte și a efectua corecții de erori pe GitHub.

Odată ce cartea a fost redactată, acesta a trecut prin mai multe runde de revizuiri tehnice. Mulțumesc lui Cricket Liu și Lorne Lantz pentru revizuirea lor aprofundată, comentarii și pentru sprijin.

Mai mulți dezvoltatori bitcoin au contribuit cu mostre de cod, recenzii, comentarii și încurajari. Mulțumesc lui Amir Taaki și Eric Voskuil, de exemplu, fragmente de cod și multe comentarii grozave; Chris Kleeschulte pentru contribuția la apendixul Bitcore; Vitalik Buterin și Richard Kiss pentru ajutor la codul pentru calul eliptic și pentru cod; Gavin Andresen pentru corecturi, comentarii și încurajare; Michalis Kargakis pentru comentarii, contribuții și notările btcd; și Robin Inge pentru trimiterea de errata îmbunătățind cea de-a doua ediție de tipar. În cea de-a două ediție, am primit din nou mult ajutor de la mulți dezvoltatori Bitcoin Core, inclusiv Eric Lombrozo care a demistificat Martorul Segregat, Luke Dashjr care a ajutat la îmbunătățirea capitolului privind tranzacțiile, Johnson Lau care a revizuit Martorul Segregat și alte capitole și multe altele. Îi datorez mulțumiri lui Joseph Poon, Tadge Dryja și Olaoluwa Osuntokun, care au explicat Lightning Network, mi-au revizuit scrisul și mi-au răspuns întrebărilor când am rămas blocat.

Datorez dragostea mea pentru cuvinte și cărți mamei mele, Theresa, care m-a crescut într-o casă cu cărți captușind fiecare perete. Mama mea mi-a cumpărat de asemenea, primul meu calculator în 1982, în ciuda faptului că mă consider technofob. Tatăl meu, Menelaos, un inginer în construcții civile care tocmai a publicat prima sa carte la 80 de ani, a fost cel care m-a învățat gândirea logică și analitică și o iubire pentru știință și inginerie.

Vă mulțumesc tuturor că m-ați susținut pe parcursul acestei călătorii.

Contributie traducere în limba română: <http://bitcoincore.tech/>

# Glosar Rapid

Aceasta secțiune conține mulți dintre termenii utilizați în raport cu Bitcoin. Acești termeni sunt utilizați în întreaga carte, deci puneți un semn aici pentru o referință rapidă.

## adresă (address)

O adresă bitcoin este de forma 1DSrfJdB2AnWaFNgSbv3MZC2m74996JafV. Este compusă dintr-un sir de litere și numere. Este o versiune base58check a unui rezumat de cheie publică pe 160 de biți. La fel cum cereți cuiva să trimite un e-mail la adresa dumneavoastră de e-mail, la fel puteți solicita cuiva să vă trimite bitcoin la una dintre adresele dumneavoastră bitcoin.

## bip

Propuneri de Îmbunătățire Bitcoin (Bitcoin Improvement Proposals). Un set de propuneri pe care membrii comunității bitcoin le-au depus pentru îmbunătățirea bitcoin. De exemplu, BIP-21 este o propunere de îmbunătățire a identificatorului uniform al resurselor (uniform resource identifier - URI) bitcoin.

## bitcoin

Numele unității valutare (moneda), al rețelei și al software-ului.

## bloc (block)

O grupare de tranzacții, marcată cu o marcă de timp și o amprentă a blocului anterior. Antetul blocului este rezumat pentru a produce o doavadă de lucru, validând astfel tranzacțiile. Blocurile valide sunt adăugate la lanțul-de-blocuri principal prin consensul rețelei.

## lanț-de-blocuri (blockchain)

O listă de blocuri validate, fiecare legat de predecesorul său până la blocul geneză.

## Problema Generalilor Bizantini

Un sistem informatic fiabil trebuie să poată face față defecțiunii uneia sau mai multor componente ale sale. O componentă defectă poate prezenta un tip de comportament care este adesea ignorat - și anume, trimitera de informații conflictuale la diferite părți ale sistemului. Problema de a face față acestui tip de defect este exprimată abstract ca Problema Generalilor Bizantini.

## coinbase

Un câmp special folosit ca unică intrare pentru tranzacțiile coinbase. Coinbase permite revendicare recompensei pentru bloc și oferă până la 100 de octeți pentru date arbitrate. Nu trebuie confundat cu tranzacția Coinbase.

## tranzacția coinbase (coinbase transaction)

Prima tranzacție dintr-un bloc. Creată întotdeauna de un miner, include o singură coinbase. Nu trebuie confundată cu Coinbase.

## stocare la rece (cold storage)

Se referă la păstrarea unei rezerve de bitcoin offline. Stocarea la rece se realizează atunci când cheile private bitcoin sunt create și stocate într-un mediu offline sigur. Stocarea la rece este

importantă pentru oricine deține bitcoin. Calculatoarele online sunt vulnerabile la atacul hackerilor și nu ar trebui utilizate pentru a stoca o cantitate semnificativă de bitcoin.

### **confirmări (confirmations)**

Odată ce tranzacția este inclusă într-un bloc, aceasta are o confirmare. De îndată ce *un alt* bloc este minat pe același lanț-de-blocuri, tranzacția are două confirmări, etc. Șase sau mai multe confirmări sunt considerate suficiente pentru ca o tranzacție nu poate fi inversată.

### **consens (consensus)**

Când mai multe noduri, de obicei majoritatea nodurilor din rețea, au aceleași blocuri în lanțul lor de blocuri validat local. Nu trebuie confundat cu regulile de consens.

### **reguli de consens (consensus rules)**

Regulile de validare a blocurilor pe care le urmează nodurile complete pentru a rămâne în consens cu alte noduri. Nu trebuie confundat cu consensul.

### **dificultate (difficulty)**

O setare la nivel de rețea care controlează cât de multă putere de calcul este necesară pentru a produce o dovedă-de-lucru.

### **reajustarea dificultății (difficulty retargeting)**

O recalculare la nivel de rețea a dificultății care apare o dată la 2.016 blocuri și ia în considerare puterea de rezumare (hashing power) a celor 2.016 blocuri anterioare.

### **țintă de dificultate (difficulty target)**

O dificultate la care toate calculele din rețea vor găsi blocuri la aproximativ fiecare 10 minute.

### **cheltuire-dublă (double-spending)**

Cheltuirea dublă este rezultatul cheltuirii cu succes a unor bani de mai multe ori. Bitcoin protejează împotriva cheltuirii duble prin verificarea fiecărei tranzacții adăugate în lanțul-de-blocuri pentru a se asigura că intrările pentru tranzacție nu au fost deja cheltuite anterior.

### **ECDSA**

Algoritmul de Semnătură Digitală Curbă Eliptică (Elliptic Curve Digital Signature Algorithm) sau ECDSA este un algoritm criptografic utilizat de bitcoin pentru a se asigura că fondurile pot fi cheltuite doar de către proprietarii lor.

### **nonce extra (extra nonce)**

Pe măsură ce dificultatea a crescut, minerii au parcurs adesea toate cele 4 miliarde de valori ale nonce-ului fără a găsi un bloc. Deoarece scriptul coinbase poate stoca între 2 și 100 de octeți de date, minerii au început să folosească acel spațiu ca spațiu nonce extra, permitându-le să exploreze o gamă mult mai mare de valori ale antetului blocului pentru a găsi blocuri valide.

### **comisioane (fees)**

Expeditorul unei tranzacții include adesea un comision pentru rețea pentru procesarea tranzacției solicitate. Majoritatea tranzacțiilor necesită un comision minim de 0,5 mBTC.

## **bifurcare (fork)**

Bifurcare, cunoscută și sub numele de bifurcare accidentală, apare atunci când două sau mai multe blocuri au aceeași înălțime, bifurcând lanțul-de-blocuri. De obicei apare atunci când doi sau mai mulți mineri găsesc blocuri aproape în același timp. Se poate întâmpla și ca parte a unui atac.

## **block geneză (genesis block)**

Primul bloc din lanțul-de-blocuri, folosit pentru inițializarea criptomonedei.

## **bifurcare hard (hard fork)**

Bifurcare hard, cunoscută și sub numele de Schimbare folosind Bifurcare Hard (Hard-Forking Change), este o divergență permanentă în lanțul-de-blocuri, apare frecvent când nodurile neactualizate nu pot valida blocurile create de nodurile actualizate care respectă reguli de consens mai noi. Nu trebuie confundată cu bifurcare, bifurcare soft, bifurcare software sau bifurcare Git.

## **portofel hardware (hardware wallet)**

Un portofel hardware este un tip special de portofel bitcoin care stochează cheile private ale utilizatorului într-un dispozitiv hardware securizat.

## **rezumat (hash)**

O amprentă digitală a unor intrări binare.

## **blocări-pe-rezumat (hashlocks)**

O blocare-pe-rezumat (hashlock) este un tip de restricție care limitează cheltuirea unei ieșiri până când o informație specificată este dezvăluită public. Blocările-pe-rezumat au proprietatea utilă că, odată ce orice blocare-pe-rezumat este deschisă public, orice altă blocare-pe-rezumat securizată folosind aceeași cheie poate fi, de asemenea, deschisă. Acest lucru face posibilă crearea de ieșiri multiple, care sunt toate restricționate de aceeași blocare-pe-rezumat și care devin cheltuibile în același timp.

## **Protocol HD (Determinist Ierarhic)**

Protocolul Determinist Ierarhic (Hierarchical Deterministic - HD) de creare și transfer de chei (BIP32), care permite crearea de chei copil din cheile părinte într-o ierarhie.

## **Portofel HD (Determinist Ierarhic)**

Portofelele care utilizează protocolul Determinist Ierarhic (Protocol HD) de creare și transfer de chei (BIP32).

## **Semințe de portofel HD (HD wallet seed)**

Sămânța de portofel HD sau sămânța rădăcină este o valoare potențial scurtă folosită ca sămânță pentru a genera cheia privată principală și codul de lanț principal pentru un portofel HD.

## **HTLC**

Un contract cu timp de blocare (Hashed TimeLock Contract - HTLC) este o clasă de plăți care utilizează blocări-pe-rezumat și timpi-de-blocare pentru a solicita ca destinatarul unei plăți să confirme primirea plății înainte de o dată limită, prin generarea de dovezi criptografice de plată

sau să piardă capacitatea de a solicita plata, returnând-o către plătitor.

## KYC

Cunoașteți-vă clientul (Know Your Customer - KYC) este procesul unei afaceri de identificare și verificare a identității clientilor săi. Termenul este folosit și pentru a face referire la regulamentul bancar care reglementează aceste activități.

## LevelDB

LevelDB este soft de stocare pe disc a unor perechi chei-valoare. LevelDB este o bibliotecă suplă, care are ca singur scop persistența pe mai multe platforme.

## Rețele Lightning (Lightning Networks)

Lightning Network este o implementare a contractelor cu timpi de blocare (HTLC) cu canale bidirectionale de plată, care permite direcționarea în siguranță a plăților pe mai multe canale de plată de-la-egal-la-egal. Acest lucru permite formarea unei rețele în care orice seamăn (peer) din rețea poate plăti orice alt seamăn (peer), chiar dacă nu au un canal direct deschis între ei.

## Timp-de-blocare (Locktime)

Timp-de-blocare, sau mai tehnic nLockTime, este partea unei tranzacții care indică cea mai devreme dată sau cel mai devreme bloc când această tranzacție poate fi adăugată la lanțul-de-blocuri.

## mempool

Mempool-ul bitcoin (bazin de memorie) este o colecție a tuturor datelor de tranzacții dintr-un bloc care au fost verificate de nodurile bitcoin, dar încă nu sunt confirmate.

## rădăcină merkle (merkle root)

Nodul rădăcină al unui arbore merkle, descendent al tuturor perechilor de rezumate din arbore. Anteturile blocului trebuie să includă o rădăcină merkle validă descendentă din toate tranzacțiile din acel bloc.

## arbore merkle (merkle tree)

Un arbore construit prin rezumarea datelor (frunzelor) în perechi, apoi prin crearea de perechi din rezumatele obținute, și tot așa până când rămâne un singur rezumat, rădăcina merkle. În bitcoin, frunzele sunt aproape întotdeauna tranzacții dintr-un singur bloc.

## miner

Un nod de rețea care găsește o dovedă-de-lucru validă pentru blocuri noi, prin rezumări repetitive.

## multisemnătură (multisignature)

Multisemnătură (multisig) se referă la solicitarea a mai mult de o cheie pentru autorizarea unei tranzacții bitcoin.

## rețea (network)

O rețea de-la-egal-la-egal care propagă tranzacțiile și blocurile la fiecare nod bitcoin din rețea.

## nonce

”Nonce-ul” dintr-un bloc bitcoin este un câmp pe 32 de biți (4 octeți) a cărui valoare este setată astfel încât rezumatul blocului să conțină o serie de zerouri la început. Restul câmpurilor nu pot fi modificate, deoarece au un scop anume.

## tranzacții în-afara-lanțului (off-chain transactions)

O tranzacție în-afara-lanțului este circulația valorii în afara lanțului-de-blocuri. În timp ce o tranzacție pe-lanț denumită de obicei simplu *o tranzacție* modifică lanțul-de-blocuri și depinde de lanțul-de-blocuri pentru a determina validitatea acesteia, o tranzacție în-afara-lanțului se bazează pe alte metode de înregistrare și validare a tranzacției.

## operator (opcode)

Operatori (coduri de operare) din limbajul Bitcoin Script care împing date sau îndeplinesc funcții într-un script pubkey sau într-un script semnatură.

## Protocolul Open Assets

Protocolul Open Assets este un protocol simplu și puternic, construit peste lanțul-de-blocuri bitcoin. Permite emiterea și transferul de active create de utilizator.

## OP\_RETURN

Un operator utilizat într-una dintre ieșirile dintr-o tranzacție OP\_RETURN. Nu trebuie confundat cu tranzacția OP\_RETURN.

## Tranzacție OP\_RETURN

Un tip de tranzacție care adaugă date arbitrar la un script pubkey (necheltuibil) pe care nodurile complete nu trebuie să le stocheze în baza lor de date UTXO. Nu trebuie confundat cu operatorul OP\_RETURN.

## bloc orfan (orphan block)

Blocurile al căror bloc părinte nu a fost procesat de nodul local, deci nu pot fi validate complet. Nu trebuie confundat cu bloc învechit.

## tranzacții orfane (orphan transactions)

Tranzacții care nu pot intra în bazin din cauza uneia sau a mai multor tranzacții de intrare lipsă.

## ieșire (output)

Ieșire, ieșire de tranzacție sau TxOut este o ieșire într-o tranzacție care conține două câmpuri: un câmp valoric pentru transferul a zero sau mai mulți satoshi și un script pubkey pentru a indica ce condiții trebuie îndeplinite pentru ca acei satoshi să poată fi cheltuiți în continuare.

## P2PKH

Tranzacțiile care plătesc o adresă bitcoin conțin scripturi P2PKH sau Plată-către-Rezumat-Cheie-Publică (Pay To PubKey Hash). O ieșire blocată de un script P2PKH poate fi deblocată (cheltuită) prin prezentarea unei chei publice și a unei semnături digitale create de cheia privată corespunzătoare.

## P2SH

P2SH sau Plată-către-Rezumat-Script (Pay-to-Script-Hash) este un nou tip puternic de tranzacție care simplifică mult utilizarea scripturilor complexe de tranzacții. Cu P2SH scriptul complex care detaliază condițiile de cheltuire a ieșirii (script de răscumpărare) nu este prezentat în scriptul de blocare. În schimb, doar un rezumat al acestuia este în scriptul de blocare.

## Adresă P2SH

Adresele P2SH sunt codificări Base58Check ale rezumatului de 20 de octeți ale unui script, adresele P2SH folosesc prefixul versiunii "5", ceea ce are ca rezultat adrese codificate Base58Check care încep cu un "3". Adresele P2SH ascund toată complexitatea, astfel încât persoana care efectuează o plată să nu vadă scriptul.

## P2WPKH

Semnătura unui P2WPKH Plată-către-Martor-Rezumat-Cheie-Publică (Pay-to-Witness-Public-Key-Hash) conține aceleași informații ca o cheltuire P2PKH, dar este localizată în câmpul martor și nu în câmpul scriptSig. scriptPubKey este de asemenea modificat.

## P2WSH

Diferența dintre P2SH și P2WSH (Pay-to-Witness-Script-Hash) se referă la schimbarea locației dovezii criptografice din câmpul scriptSig în câmpul martor și scriptPubKey care este, de asemenea, modificat.

## portofel de hârtie (paper wallet)

În sensul cel mai specific, un portofel de hârtie este un document care conține toate datele necesare pentru a genera orice număr de chei private bitcoin, formând un portofel de chei. Cu toate acestea, oamenii folosesc adesea termenul pentru a însemna orice mod de stocare offline a bitcoin ca document fizic. Această a doua definiție include de asemenea chei de hârtie și coduri de răscumpărare.

## canale de plată (payment channels)

Un canal de microplată sau un canal de plată este o clasă de tehnici concepute pentru a permite utilizatorilor să efectueze mai multe tranzacții bitcoin fără a transmite toate tranzacțiile către lanțul-de-blocuri bitcoin. Într-un canal de plată tipic, doar două tranzacții sunt adăugate în lanțul-de-blocuri, dar un număr nelimitat sau aproape nelimitat de plăți poate fi făcut între participanți.

## minerit în bazin (pooled mining)

Mineritul în bazin este o abordare minieră în care mai mulți clienți contribuie la generarea unui bloc, apoi împart recompensa blocului în funcție de puterea de procesare cu care au contribuit.

## Dovadă-a-Mizei (Proof-of-Stake)

Dovadă-a-Mizei (PoS) este o metodă prin care o rețea lanț-de-blocuri de criptomonedă își propune să obțină un consens distribuit. Dovadă-Mizei solicită utilizatorilor să demonstreze deținerea unei anumite sume de monedă ("miza" lor în monedă).

## Dovadă-de-Lucru (Proof-of-Work)

Un fragment de date care necesită calcul semnificativ pentru a fi găsi. În bitcoin, minerii trebuie

să găsească o soluție numerică la algoritmul SHA256 care îndeplinește o țintă la nivelul întregii rețele, ținta de dificultate.

### **recompensă (reward)**

Suma inclusă în fiecare nou bloc ca recompensă de către rețea pentru minerul care a găsit soluția Dovadă-de-Lucru. În prezent este 6,25 BTC pe bloc.

### **RIPEMD-160**

RIPEMD-160 este o funcție de rezumare criptografică pe 160 de biți. RIPEMD-160 este o versiune consolidată a RIPEMD cu un rezumat pe 160 de biți și este de așteptat să fie sigură pentru următorii zece ani sau mai mult.

### **satoshi**

Un satoshi este cea mai mică denominare în bitcoin care poate fi înregistrată pe lanțul-de-blocuri. Este echivalentul a 0.00000001 bitcoin și poartă numele creatorului Bitcoin, Satoshi Nakamoto.

### **Satoshi Nakamoto**

Satoshi Nakamoto este numele folosit de persoana sau persoanele care au conceput Bitcoin și au creat implementarea sa de referință inițială, Bitcoin Core. Ca parte a implementării, au conceput și prima bază de date lanț-de-blocuri. În acest proces, ei au fost primii care au rezolvat problema dublei cheltuiiri pentru monede digitale. Identitatea lor reală rămâne necunoscută.

### **Script**

Bitcoin utilizează un sistem de scriptare pentru tranzacții. Asemănător cu Forth, Script este simplu, bazat pe stivă și procesat de la stânga la dreapta. În mod intenționat nu este Turing-complet, fără bucle.

### **ScriptPubKey (sau pubkey script)**

ScriptPubKey sau script pubkey, este un script inclus în ieșiri care stabilește condițiile care trebuie îndeplinite pentru ca satoshii să poată să fie cheltuiți. Datele pentru îndeplinirea condițiilor pot fi furnizate într-un script semnătură.

### **ScriptSig (sau script semnătură)**

ScriptSig sau script semnătură, sunt datele generate de către cel care cheltuiește care sunt aproape întotdeauna folosite ca variabile pentru a satisface un script pubkey.

### **cheie secretă (sau cheie privată)**

Numărul secret care deblochează bitcoin trimis la adresa corespunzătoare. O cheie secretă arată astfel:

```
5J76sF8L5jTtzE96r66Sf8cka9y44wdpJjMwCxR3tzLh3ibVPxh
```

### **Martor Segregat (Segregated Witness)**

Martorul Segregat este o actualizarea protocolului Bitcoin în care datele de semnătură ("martor") sunt separate de datele expeditorului/receptorului pentru a optimiza și mai mult structura tranzacțiilor. Martorul segregat a fost implementat ca o bifurcare soft; o schimbare

care, din punct de vedere tehnic, face ca regulile protocolului bitcoin să fie mai restrictive.

## SHA

Algoritmul de Rezumare Sigur (Secure Hash Algorithm ) sau SHA este o familie de funcții de rezumare criptografice publicate de Institutul Național de Standarde și Tehnologie (NIST) din SUA.

## Verificare Simplificată a Plății (Simplified Payment Verification - SPV)

SPV (Simplified Payment Verification) sau Verificarea simplificată a Plății este o metodă pentru a verifica dacă anumite tranzacții au fost incluse într-un bloc, fără a descărca întregul bloc. Această metodă de verificare este adesea folosită de clienții supli bitcoin.

## bifurcare soft (soft fork)

Bifurcarea Soft sau Schimbare prin Bifurcare Soft este o bifurcare temporară din lanțul-de-blocuri, care apare în mod obișnuit când minerii folosesc noduri neactualizate și nu respectă o nouă regulă de consens despre care nodurile lor nu știe. Nu trebuie confundat cu bifurcare, bifurcare hard, bifurcare software sau bifurcare Git

## bloc învechit (stale block)

Blocul care a fost minat cu succes, dar care nu este inclus în cel mai bun lanț-de-blocuri actual, probabil pentru că un alt bloc la aceeași înălțime și-a extins primul lanț. Nu trebuie confundat cu blocul orfan.

## timpi-de-blocare (timelocks)

Un timp-de-blocare este un tip de limitare care restricționează cheltuirea bitcoin până la o dată viitoare sau o anumită înălțime a blocului. Timpii-de-blocare au o caracteristică importantă în multe contracte bitcoin, inclusiv canale de plată și contracte cu timp-de-blocare.

## tranzacție

În termeni simpli, un transfer de bitcoin de la o adresă la alta. Mai precis, o tranzacție este o structură de date semnată care exprimă un transfer de valoare. Tranzacțiile sunt transmise prin rețeaua bitcoin, colectate de către mineri și incluse în blocuri, devenind permanente pe lanțul-de-blocuri.

## bazin de tranzacții (transaction pool)

O colecție neordonată de tranzacții care nu se află în blocurile din lanțul principal, dar pentru care avem tranzacții de intrare.

## Turing complet

Un limbaj de programare se numește "Turing complet" dacă poate rula orice program pe care îl poate rula o mașină Turing, având suficient timp și memorie.

## ieșire de tranzacție necheltuită (unspent transaction output - UTXO)

UTXO este o ieșire a unei tranzacții care poate fi utilizată ca o intrare într-o nouă tranzacție.

## portofel (wallet)

Software care conține toate adresele dumneavoastră bitcoin și cheile secrete. Folosiți-l pentru a trimite, primi și stoca bitcoinul dumneavoastră.

## Format de Import Portofel (Wallet Import Format - WIF)

Formatul WIF sau Wallet Import Format este un format de schimb de date conceput pentru a permite exportul și importul unei singure chei private cu un indicator care indică dacă folosește sau nu o cheie publică comprimată.

Unele definiții contribuite au fost obținute sub licență CC-BY de la [bitcoin Wiki](#) sau din alte surse de documentare open source.

# Introducere

## Ce este Bitcoin?

Bitcoin este o colecție de concepte și tehnologii care formează bazele unui ecosistem monetar digital. Unități monetare numite bitcoin sunt folosite pentru a stoca și transmite valoare între participanții rețelei bitcoin. Utilizatorii bitcoin comunică unii cu alții folosind protocolul bitcoin în principal via internet, deși alte rețele de transport pot fi deasemenea folosite. Stiva de protocol bitcoin, disponibilă ca software open source, poate fi rulată pe o gamă largă de dispozitive de calcul, inclusiv laptop-uri și smartphone-uri, făcând tehnologia ușor de accesat.

Utilizatorii pot transfera bitcoin în rețea pentru a face aproape tot ce se poate face cu monede convenționale, inclusiv să cumpere și să vândă bunuri, să trimită bani către persoane sau organizații, sau să acorde credite. Bitcoin se poate achiziționa, vinde, precum și schimba pentru alte valute la case de schimb valutar de specialitate. Bitcoin este într-un sens, forma perfectă de bani pentru internet, pentru că este rapid, sigur și nu are granițe.

Spre deosebire de monedele tradiționale, bitcoin este în întregime virtual. Nu există monede fizice și nici chiar monede digitale în sine. Monedele sunt implicate în tranzacții care transferă valoare de la expeditor la destinatar. Utilizatorii de bitcoin dețin chei care le permit să dovedească deținerea de bitcoin în rețeaua bitcoin. Cu aceste chei utilizatorii pot semna tranzacții pentru a debloca valoarea și a o cheltui prin transferul la un nou proprietar. Cheile sunt adesea stocate într-un portofel digital pe calculatorul sau smartphone-ului fiecărui utilizator. Posesia cheii care poate semna o tranzacție este singura precondiție pentru a cheltui bitcoin, controlul fiind în întregime în mâinile fiecărui utilizator.

Bitcoin este un sistem de-la-egal-la-egal (peer-to-peer) distribuit. Prin urmare, nu există un server "central" sau un punct de control. Bitcoin sunt creați printr-un proces numit "minerit", care implică o competiție pentru a găsi soluții la o problemă matematică în timp ce se procesează tranzacții bitcoin. Orice participant în rețeaua bitcoin poate opera ca miner (i.e. oricine folosește un dispozitiv care rulează în întregime stiva de protocol bitcoin), folosind puterea de procesare a calculatorului lui pentru a verifica și înregistra tranzacții. La fiecare 10 minute, în medie, un miner bitcoin reușește să valideze tranzacțiile ultimelor 10 minute și este recompensat cu bitcoin noi creați. Practic, mineritul de bitcoin descentralizează emiterea de monedă și funcțiile de compensare ale unei bănci centrale și înlocuiește nevoia existenței unei bănci centrale.

Protocolul bitcoin include algoritmi care reglementează funcțiile de minerit în rețea. Dificultatea sarcinii de procesare pe care minerii trebuie să o efectueze este ajustată dinamic astfel încât, în medie, cineva reușește la fiecare 10 minute indiferent de căți mineri (și cât de multă

putere de procesare) concureaza în orice moment. Protocolul înjumătățește la fiecare 4 ani rata la care bitcoin noi sunt creați, și limitează numărul total de bitcoin care vor fi creati la un total puțin sub 21 de milioane de monede. Rezultatul este că numărul total de bitcoin în circulație urmează indeaproape o linie curbă ușor predictibilă care se apropie de 21 de milioane in anul 2140. Datorită diminuării ratei de emitere de bitcoin, pe termen lung, moneda bitcoin este deflaționistă. În plus, bitcoin nu poate fi umflat prin "tiparirea" de bani noi peste nivelul anticipat de emitere.

În culise, bitcoin este de asemenea numele protocolului, o rețea de-la-egal-la-egal, și o inovație în calcul distribuit. Moneda bitcoin este în realitate doar prima aplicație a acestei invenții. Bitcoin reprezintă punctul culminant al deceniilor de cercetare în criptografie și sisteme distribuite și include patru inovații cheie reunite într-o combinație unică și puternică. Bitcoin este format din:

- O rețea de-la-egal-la-egal (peer-to-peer) descentralizată (protocolul bitcoin)
- Un registru de tranzacții public (lanțul-de-blocuri sau blockchain)
- Un set de reguli pentru validarea independentă a tranzacțiilor și emitere de monedă (reguli de consens)
- Un mecanism pentru atingerea unui consens global descentralizat asupra lanțului-de-blocuri (blockchain) valid (algoritmul Dovadă-de-Lucru)

Ca programator, văd bitcoin asemănător unui internet al banilor, o rețea care propagă valoare și securizează deținerea de active digitale folosind calculul distribuit. Bitcoin este mai mult decât pare la prima vedere.

În acest capitol vom începe să explicăm unele dintre principalele concepte și termeni, obținerea software-ului necesar și folosirea bitcoin pentru tranzacții simple. În următoarele capitole vom începe să analizăm straturile tehnologiei care fac ca bitcoin să fie posibil și vom examina mecanismele interne ale rețelei și ale protocolului bitcoin.

## Monede Digitale Înainte De Bitcoin

Apariția de monede digitale viabile este strâns legată de evoluțiile în domeniul criptografiei. Acest lucru nu este surprinzător dacă se iau în considerare provocările fundamentale implicate în utilizarea biților pentru a reprezenta valoare care poate fi schimbată pentru bunuri și servicii. Trei întrebări de bază pentru oricine acceptă monede digitale sunt:

1. Pot să am încredere că banii sunt autentici și nu contrafăcuți?
2. Pot să am încredere că banii digitali pot fi cheltuiți doar o singură dată (cunoscută ca problema "cheltuirii duble")?
3. Pot să fiu sigur că nimeni altcineva nu poate pretinde că banii le aparțin lor și nu mie?

Emitenții de monedă de hârtie se luptă constant cu problema contrafacerii utilizând hârtii și tehnologii de tipărire din ce în ce mai sofisticate. Banii fizici abordează chestiunea cheltuirii duble ușor pentru că aceeași bancnotă de hârtie nu poate fi în două locuri simultan. Desigur, banii convenționali sunt, de asemenea, de multe ori stocați și transmiși digital. În aceste cazuri, problemele de contrafacere și cheltuire dublă sunt gestionate prin compensarea tuturor tranzacțiilor electronice prin intermediul autorităților centrale care au o imagine de ansamblu asupra monedei în circulație. Pentru banii digitali, care nu pot profita de cerneluri ezoterice sau benzi holografice, criptografia oferă baza pentru încrederea în legitimitatea revendicării valorii de către un utilizator. Mai exact, semnăturile digitale criptografice permit unui utilizator să semneze un activ digital sau o tranzacție care atestă dreptul de proprietate asupra activului. Cu arhitectura potrivită, semnăturile digitale pot fi folosite de asemenea pentru a rezolva problema cheltuirii duble.

Când criptografia a început să devină mai larg accesibilă și înțeleasă la sfârșitul anilor 1980, mulți cercetători au început să încerce utilizarea criptografiei pentru a construi valute digitale. Aceste proiecte timpurii de valute digitale au emis bani digitali, de obicei, susținuți de o monedă națională sau de metale prețioase, cum ar fi aurul.

Deși aceste valute digitale timpurii funcționau, ele erau centralizate și, prin urmare, ușor de atacat de către guverne și hackeri. Monedele digitale timpurii foloseau un centru de compensare pentru decontarea tuturor tranzacțiilor la intervale regulate, la fel ca un sistem bancar tradițional. Din păcate, în cele mai multe cazuri aceste monede în curs de formare au fost vizate de guverne îngrijorate și în cele din urmă contestate până la dispariție. Unele au eșuat în prăbușiri spectaculoase atunci când societatea-mamă a fost lichidată brusc. Pentru a fi robust împotriva intervențiilor antagonice, fie ele guverne legitime sau elemente criminale, o monedă digitală *descentralizată* a fost necesară pentru a evita existența unui singur punct de atac. Bitcoin este un astfel de sistem, proiectat să fie *descentralizat* și liber de orice autoritate centrală sau punct de control care poate fi atacat sau corupt.

## Istoria Bitcoin

Bitcoin a fost inventat în 2008 prin publicarea unei lucrări intitulată "Bitcoin: A Peer-to-Peer Electronic Cash System,"<sup>[1]</sup> scrisă sub pseudonimul de Satoshi Nakamoto (vezi [Referatul Bitcoin de Satoshi Nakamoto](#)). Nakamoto a combinat câteva invenții anterioare, cum ar fi b-money și

HashCash pentru a creea un sistem electronic de numerar complet descentralizat care nu se bazează pe nicio autoritate centrală pentru emiterea de monedă sau pentru decontarea și validarea tranzacțiilor. Inovația cheie a fost să folosească un sistem de calcul distribuit (denumit "Dovadă-de-Lucru") care să realizeze o "alegere" globală la fiecare 10 minute, permitând rețelei descentralizate să ajungă la un *consens* privind starea tranzacțiilor. Aceasta rezolvă elegant problema dublei cheltuiiri unde o singura unitate monetară poate fi cheltuită de două ori. Anterior, problema dublei-cheltuiiri era o slăbiciune a monedelor digitale și fusese abordată prin compensarea tuturor tranzacțiilor prin intermediul unui centru de compensare.

Rețeaua bitcoin a inceput în 2009, bazată pe o implementare de referință publicată de Nakamoto și revizuită de atunci de mulți alți programatori. Implementarea algoritmului Dovadă-de-Lucru (minerit) care oferă securitate și rezistență pentru bitcoin a crescut exponențial în putere, și acum depășește puterea de calcul combinată a celor mai puternice super-calculatoare din lume. Valoarea totală de piață a bitcoin a depășit în unele momente 135 de miliarde de dolari, depinzând de cursul de schimb bitcoin-dolar. Cea mai mare tranzacție procesată până acum a fost de 400 de milioane de dolari, transmisă instant și procesată pentru un comision de sub 1 dolar.

Satoshi Nakamoto s-a retras din spațiul public în aprilie 2011, lăsând în seama unui grup înfloritor de voluntari responsabilitatea dezvoltării codului și a rețelei. Identitatea persoanei sau persoanelor aflate în spatele bitcoin este încă necunoscută. Cu toate acestea, nici Satoshi Nakamoto nici altcineva nu exercită control individual asupra sistemului bitcoin, care operează în baza unor principii matematice complet transparente, cod open source, și consens între participanți. Invenția în sine este revoluționară și deja a dat naștere unei noi discipline în domeniile calculului distribuit, al economiei și al econometriei.

### O Soluție la o Problemă de Calcul Distribuit

Invenția lui Satoshi Nakamoto este de asemenea o soluție nouă și practică la o problemă de calcul distribuit, cunoscută sub denumirea de "Problema Generalilor Bizantini". Pe scurt, problema constă în încercarea de a conveni asupra unui plan de acțiune sau asupra stării unui sistem prin schimbul de informații într-o rețea nesigură și potențial compromisă. Soluția lui Satoshi Nakamoto, care folosește conceptul de Dovadă-de-Lucru pentru a ajunge la consens *fără o autoritate centrală de încredere*, reprezintă o descoperire în domeniul calculului distribuit și are o aplicare largă dincolo de monedă. Poate fi folosită pentru a ajunge la consens în rețele descentralizate pentru a dovedi corectitudinea scrutinelor (alegerilor), loteriilor, registrelor de active, notariatelor digitale, și multe altele.

## Utilizările Bitcoin, Utilizatorii, și Poveștile Lor

Bitcoin este o inovație în problema antică a banilor. La fundația lor, banii pur și simplu facilitează schimbul de valoare între oameni. Prin urmare, pentru a înțelege în întregime bitcoin și utilizările sale, îl vom examina din perspectiva oamenilor care îl folosesc. Fiecare dintre oameni și poveștile lor, cum sunt expuse aici, ilustrează una sau mai multe utilizări. Le vom vedea pe parcursul cărții:

### Comerț cu amănuntul de valoare redusă în America de Nord

Alice locuiește în nordul Californiei în zona Bay Area. A auzit de bitcoin de la prietenii ei mai tehnici și vrea să înceapă să îl folosească. Îl vom urmări povestea pe parcurs ce învață despre

bitcoin, dobândește câțiva, și apoi cheltuieste o parte din ei ca să cumpere o ceașcă de cafea la cafeneaua lui Bob din Palo Alto. Această poveste ne va face cunoștință cu software-ul, casele de schimb, și tranzacțiile de bază din perspectiva unui consumator obișnuit.

### **Comerț cu amănuntul de mare valoare în America de Nord**

Carol este proprietara unei galerii de artă din San Francisco. Ea vinde tablouri scumpe pentru bitcoin. Această poveste ne va familiariza cu riscurile unui atac de tip consens "51%" în cazul comercianților cu amănuntul de produse de mare valoare.

### **Servicii contractuale offshore**

Bob, proprietarul cafenelei din Palo Alto, își construiește un site web nou. El a contractat un dezvoltator web indian, Gopesh, care locuiește în Bangalore, India. Gopesh a fost de acord să fie plătit în bitcoin. Această poveste va examina utilizarea bitcoin pentru externalizare, servicii de contractare, și transferuri internaționale.

### **Magazin Online**

Gabriel este un adolescent întreprinzător din Rio de Janeiro. El deține un magazin online care vinde tricouri, șorțuri și stickere brand-uite cu bitcoin. Gabriel este prea Tânăr pentru a avea un cont bancar, dar părinții îi încurajează spiritul antreprenorial.

### **Donații caritabile**

Eugenia este directorul unei organizații caritabile pentru copii în Filipine. Recent, ea a descoperit bitcoin și vrea să-l folosească pentru a accesa un întreg nou grup de donatorii străini și interni în scopul de a atrage fonduri pentru activitatile ei de caritate. Ea investighează, de asemenea, moduri de a folosi bitcoin pentru a distribui rapid fonduri în zonele în care este nevoie. Această poveste va exemplifica utilizarea bitcoin pentru strângerea de fonduri la nivel global dincolo de valute și frontiere, precum și utilizarea unui registru deschis pentru transparență în organizațiile caritabile.

### **Import/export**

Mohammed este un importator de electronice din Dubai. El încearcă să folosească bitcoin pentru a cumpăra electronice din Statele Unite și China pentru importul în Emiratele Arabe Unite pentru a accelera procesul de plăti pentru importuri. Această poveste va arăta cum bitcoin poate fi folosit pentru plăti internaționale mari business-to-business legate de bunuri fizice.

### **Mineritul de bitcoin**

Jing este un student la școală de calculatoare din Shanghai. El a construit o instalație de "minerit" bitcoin folosind abilitățile sale științifice pentru a-și suplimenta veniturile. Această poveste va examina baza "industrială" a bitcoin: echipamentele specializate utilizate pentru a securiza rețeaua bitcoin și pentru emisie de monedă nouă.

Fiecare dintre aceste povesti se bazează pe oameni reali și industrii reale care în prezent folosesc bitcoin pentru a crea noi piețe, noi industrii, și soluții inovatoare la problemele economice globale.

## **Noțiuni de bază**

Bitcoin este un protocol care poate fi accesat folosind o aplicație client care înțelege protocolul. Un "portofel bitcoin" este cea mai comună interfață a utilizatorilor cu sistemul bitcoin, la fel cum un

browser web este cea mai comună interfață a utilizatorilor pentru protocolul HTTP. Există multe implementări și branduri de portofele bitcoin, la fel cum există multe branduri de browsere web (e.g., Chrome, Safari, Firefox, și Internet Explorer). Și la fel cum cu toții avem browser-ul nostru preferat (Mozilla Firefox, Yay!) și detestat (Internet Explorer, Yuck!), portofelele bitcoin variază în calitate, performanță, securitate, protecția identității și robustețe. Există și o implementare de referință a protocolului bitcoin care include un portofel, cunoscut drept "Clientul Satoshi" sau "Bitcoin Core", care este derivată din implementarea originală scrisă de Satoshi Nakamoto.

## Alegerea unui Portofel Bitcoin

Portofelele bitcoin sunt unele dintre cele mai dezvoltate aplicații din ecosistemul bitcoin. Există o competiție intensă: în timp ce un nou portofel probabil e dezvoltat în acest moment, câteva portofele de anul trecut nu mai sunt menținute activ. Multe portofele sunt centrate pe anumite platforme sau pe anumite utilizări. Unele sunt mai potrivite pentru începători, în timp ce altele sunt dotate cu funcționalități pentru utilizatori avansați. Alegerea unui portofel este foarte subiectivă și depinde de modul de utilizare și de experiența utilizatorului. Prin urmare este imposibil să recomandăm un brand sau un portofel anume. În orice caz, putem să categorizăm portofelele bitcoin după platformă și funcționalitate. Astfel putem oferi o anumită claritate asupra tuturor tipurilor de portofele existente. Chiar mai mult, mutarea cheilor sau a semințelor (seeds) între portofele se realizează relativ ușor, deci merită să încercați câteva portofele până îl găsiți pe cel care se potrivește nevoilor dumneavoastră.

Portofelele bitcoin pot fi categorisite după cum urmează, conform platformei:

### Portofel desktop

Portofelul desktop a fost primul tip de portofel bitcoin creat ca implementare de referință și mulți utilizatori rulează portofele desktop pentru funcționalitățile, autonomia, și controlul pe care îl oferă. Rularea pe sisteme de operare pentru publicul larg, cum ar fi Windows sau Mac OS are însă cu siguranță unele dezavantaje, deoarece aceste platforme sunt adesea nesigure și slab configurate.

### Portofel pentru mobil

Portofelul pentru mobil este cel mai întâlnit tip de portofel bitcoin. Rulând pe sisteme de operare pentru smart-phone-uri precum Apple iOS și Android, aceste portofele sunt adesea o alegere bună pentru utilizatorii noi. Multe sunt gândite pentru simplitate și ușurință în utilizare, dar există și portofele pentru mobil complet echipate pentru utilizatorii avansați.

### Portofel Web

Portofelele web sunt accesate folosind un browser web și stochează portofelul utilizatorului pe un server deținut de un terț. Este similar cu un webmail în sensul că se bazează în întregime pe un server terț. Unele din aceste servicii operează folosind cod client care rulează în browser-ul utilizatorului, ceea ce păstrează controlul cheilor în mâinile utilizatorului. Multe, totuși, oferă un compromis luând controlul cheilor de la utilizator în schimbul unei utilizări mai ușoare. Nu este recomandat să fie stocate cantități mari de bitcoin pe sisteme terțe.

### Portofel hardware

Portofelele hardware sunt dispozitive care rulează un portofel bitcoin independent și securizat pe un hardware dedicat. Ele sunt folosite prin USB cu un browser web pentru desktop sau prin

near-field-communication (NFC) pentru un dispozitiv mobil. Controlând toate operațiunile bitcoin pe hardware-ul specializat, aceste portofele sunt considerate foarte sigure și potrivite pentru a stoca cantități mari de bitcoin.

## Portofel de hârtie

Cheile care controlează bitcoin pot de asemenea să fie printate pentru stocare pe termen lung. Acestea sunt cunoscute ca portofele de hârtie, deși alte materiale (lemn, metal, etc.) pot fi folosite. Portofelele de hârtie oferă o metodă low-tech, dar foarte sigură pentru a stoca bitcoin pe termen lung. Stocarea offline este adesea numită și *stocare la rece*.

Un alt mod de a categoriza portofelele bitcoin este după gradul de autonomie și după cum interacționează cu rețeaua bitcoin:

## Client nod-complet

Un client complet, sau "nod complet," este un client care stochează întreaga istorie a tranzacțiilor bitcoin ( fiecare tranzacție a fiecarui utilizator, din totdeauna), administrează portofelele utilizatorului, și poate să inițieze tranzacții direct în rețeaua bitcoin. Un nod complet tratează toate aspectele protocolului și poate să valideze independent întregul lanț-de-blocuri (blockchain) și orice tranzacție. Un client nod-complet consumă resurse substanțiale (e.g., mai mult de 125 GB de disk, 2 GB de RAM) dar oferă autonomie completă și verificare independentă a tranzacțiilor.

## Client suplu

Un client suplu (lightweight), cunoscut sub denumirea de client pentru verificarea-simplă-a-plății (simple-payment-verification - SPV), se conectează la noduri-complete bitcoin (menționate anterior) pentru a accesa informații privind tranzacțiile bitcoin, dar stochează portofelul utilizatorului local și creează, validează și transmite tranzacții independent. Clienții supli interacționează direct cu rețeaua bitcoin, fără intermediari.

## Client pentru API terț

Un client pentru API (application programming interfaces) terț e un client care interacționează cu bitcoin prin intermediul unui sistem terț de interfețe de programare a aplicațiilor, în loc să se conecteze direct la rețeaua bitcoin. Portofelul poate fi stocat de către utilizator sau de un server terț, dar toate tranzacțiile trec prin un terț.

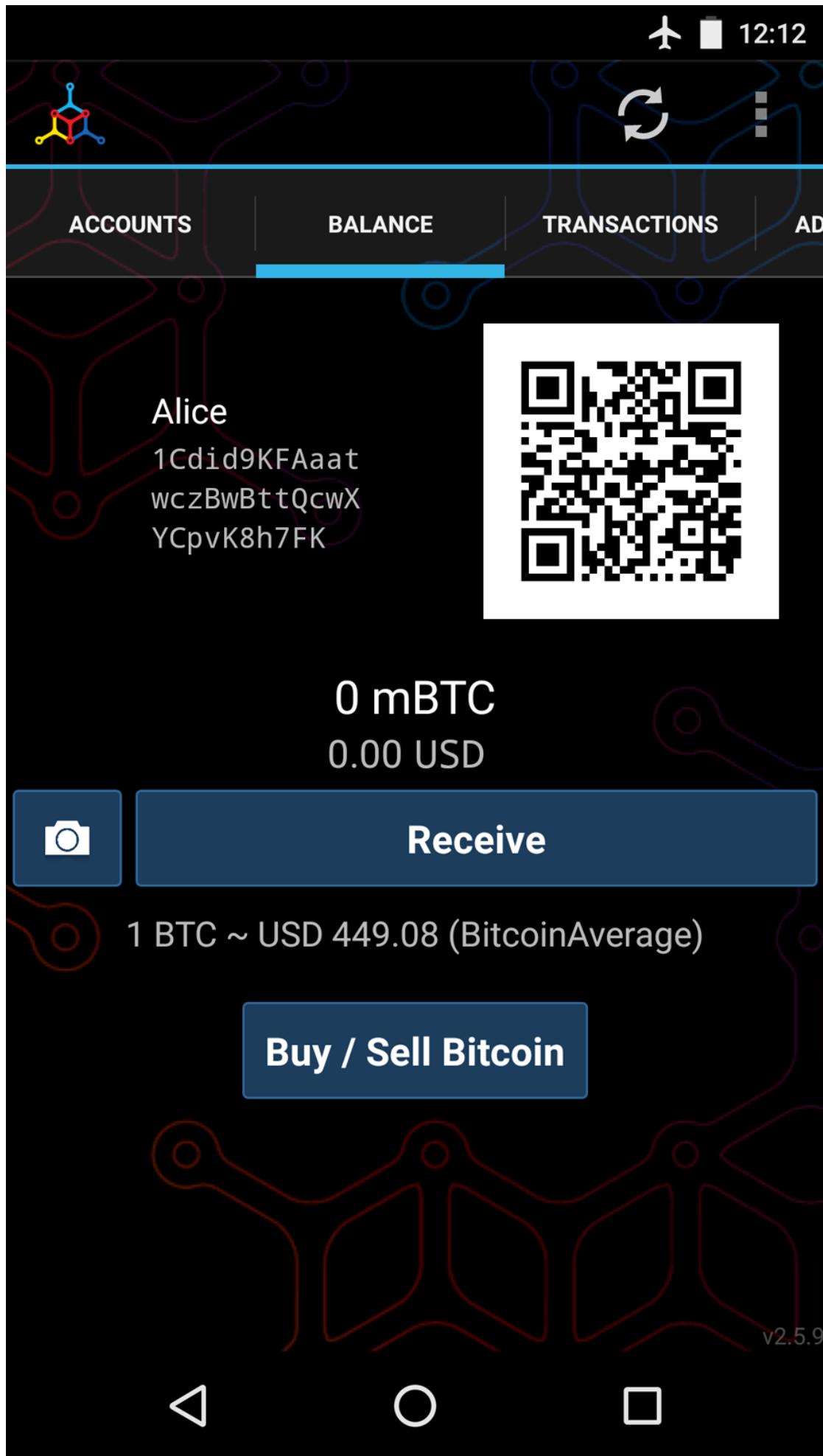
Prin combinarea acestor categorisiri, mai multe portofele bitcoin se încadrează în câteva grupuri, cele mai comune trei fiind client-complet desktop, portofel suplu pentru mobil, și portofel web la terț. Demarcarea între diferite categorii e adesea ambiguă, deoarece multe portofele rulează pe platforme multiple și pot interacționa cu rețeaua în moduri diferite.

Pentru scopul acestei cărți, vom demonstra folosirea unei varietăți de clienți bitcoin ce pot fi downloadați, de la implementarea referință (Bitcoin Core) la portofele pentru mobil și web. Unele dintre exemple vor necesita folosirea Bitcoin Core, care pe lângă faptul că este un client-complet, expune de asemenea API-uri pentru portofel, rețea și serviciile pentru tranzacții. Dacă aveți de gând să explorați interfețele programatice pentru rețeaua bitcoin, va trebui să rulați Bitcoin Core, sau unul dintre clienții alternativi (vezi [Clienți, Biblioteci și Instrumente Alternative](#)).

## Pornire Rapidă

Alice, pe care am introdus-o în [Utilizările Bitcoin, Utilizatorii, și Poveștile Lor](#), nu este un utilizator tehnic și a auzit doar recent de bitcoin de la prietenul ei Joe. Când erau la o petrecere, Joe explica din nou entuziast despre bitcoin tuturor celor din jurul lui și a oferit o demonstrație. Intrigată, Alice întreabă cum poate să înceapă să folosească bitcoin. Joe spune că un portofel mobil este cel mai potrivit pentru utilizatorii noi și îi recomandă câteva dintre portofelele lui preferate. Alice descarcă "Mycelium" pentru Android și îl instalează pe mobilul ei.

Când Alice rulează Mycelium pentru prima dată, cum este cazul cu multe portofele bitcoin, aplicația creează automat un portofel nou pentru ea. Alice vede portofelul pe ecran, după cum se vede în [Portofelul Mycelium pentru Mobil](#) (atenție: *nu* trimiteți bitcoin la această adresă exemplu, vor fi pierduți definitiv).



## Figure 1. Portofelul Mycelium pentru Mobil

Cea mai importantă parte a acestui ecran este *adresa bitcoin* a lui Alice. Pe ecran apare ca un sir lun de litere și numere: *1Cd1d9KFAaatwczBwBttQcwXYCpvK8h7FK*. Lângă adresa bitcoin a portofelului este un cod QR, o formă de cod de bare care conține aceeași informație într-un format care poate fi scanat de camera unui telefon intelligent. Codul QR este pătratul cu un model de puncte albe și negre. Alice poate să copieze adresa bitcoin sau codul QR atingând codul QR sau butonul *Receive*. În cele mai multe portofele, atingerea codului QR îl va mări, astfel încât să poată fi mai ușor scanat de camera telefonului intelligent.

**TIP**

Adretele bitcoin încep cu un 1, 3 sau cu bc1. La fel ca adretele de email, pot fi prezentate altor utilizatori bitcoin care le pot folosi ca să trimită bitcoin direct în portofelul dumneavoastră. Nu este nimic sensibil la o adresă bitcoin din perspectiva securității. Ea poate fi postată oriunde fără a pune în pericol securitatea contului. Spre deosebire de adretele de email, puteți crea noi adrese bitcoin cât de des doriti, toate acestea vor directa fondurile către portofelul dumneavoastră. De fapt, multe portofele moderne crează automat adrese noi la fiecare tranzacție pentru a maximiza anonimitatea. Un portofel este pur și simplu o colecție de adrese și de chei care deblochează fondurile conținute.

Alice este acum pregătită să primească fonduri. Portofelul ei a generat aleator o cheie privată (descrișă mai în detaliu în [Chei Private](#)) împreună cu adresa bitcoin corespunzătoare. În acest punct, adresa ei bitcoin nu este cunoscută în rețeaua bitcoin și nici nu este "înregistrată" în sistemul bitcoin. Adresa ei bitcoin e pur și simplu un număr care corespunde unei chei pe care ea o poate folosi să controleze accesul la fonduri. Adresa a fost generată independent de către portofelul ei fără vreo referință sau înregistrare la vreun serviciu. De fapt, la majoritatea portofelelor, nu există nici o asociere între adresa bitcoin și orice informație identificabilă extern, inclusiv identitatea utilizatorului. Până în momentul când la această adresă se face referință ca fiind destinatarul unei valori într-o tranzacție publicată în registrul bitcoin, adresa bitcoin este doar parte din numărul imens de adrese posibile care sunt valide în rețeaua bitcoin. Ea devine parte a adreselor cunoscute din rețea doar după ce a fost asociată cu o tranzacție.

Alice este acum pregătită să folosească noul ei portofel bitcoin.

## Obținerea Primului Dumneavoastră Bitcoin

Prima și adesea cea mai dificilă sarcină pentru un utilizator nou este să obțină niște bitcoin. Spre deosebire de alte valute, încă nu puteți cumpăra bitcoin de la bancă sau de la o casă de schimb valutar.

Tranzacțiile bitcoin sunt ireversibile. În majoritatea rețelelor de plăți electronice, precum carduri de credit, carduri de debit, PayPal, și conturile bancare tranzacțiile sunt reversibile. Pentru cineva care vinde bitcoin, această diferență introduce un risc foarte ridicat în cazul în care cumpărătorul va anula plata electronică după ce a primit bitcoin, practic înșelând vânzătorul. Pentru atenuarea acestui risc, companiile care acceptă plăți electronice tradiționale în schimbul bitcoin de obicei solicită cumpărătorilor să se supună verificării identității și verificărilor de valabilitate ale creditului, ceea ce poate dura câteva zile sau săptămâni. Ca utilizator nou, asta înseamnă că nu puteți cumpăra bitcoin instant cu un card de credit. Totuși, cu puțină răbdare și gândire creativă

nici nu va trebui.

Iată câteva metode pentru a obține bitcoin ca utilizator nou:

- Găsiți un prieten care are bitcoin și cumpărați de la el sau ea direct. Mulți utilizatori bitcoin încep astfel. Această metodă este cea mai puțin complicată. O modalitate de a cunoaște persoane cu bitcoin este să participați la o întâlnire locală bitcoin listată la [Meetup.com](https://www.meetup.com).
- Utilizați un serviciu clasificat, cum ar fi [localbitcoins.com](https://www.localbitcoins.com) pentru a găsi un vânzător din zona dumneavoastră pentru a cumpăra bitcoin în numerar într-o tranzacție personală.
- Câștigați bitcoin vânzând un produs sau serviciu pentru bitcoin. Dacă sunteți programator, vindeți-vă abilitățile de programare. Dacă sunteți frizer, tundeți pentru bitcoin.
- Folosiți bancomatele bitcoin din orașul dumneavoastră. Un bancomat bitcoin este un aparat care acceptă numerar și trimite bitcoin către portofelul bitcoin al smartphone-ului dumneavoastră. Găsiți un bancomat bitcoin aproape de dumneavoastră folosind o hartă online de la [Coin ATM Radar](https://www.coinaltradar.com).
- Folosiți o casă de schimb bitcoin legată la contul dumneavoastră bancar. Multe țări au acum case de schimb care oferă o piață unde cumpărătorii și vânzătorii pot schimba bitcoin în moneda locală. Serviciile de listare a cursului de schimb, cum ar fi [BitcoinAverage](https://www.bitcoinaverage.com), prezintă adesea o listă de case de schimb bitcoin pentru fiecare țară.

Unul dintre avantajele bitcoin față de alte sisteme de plată este că oferă utilizatorilor mult mai multă confidențialitate atunci când este folosit corect. Achiziționarea, deținerea și cheltuirea de bitcoin nu necesită divulgarea către terți de informații sensibile sau care vă pot identifica personal. Totuși, în punctele în care bitcoin intră în contact cu sistemele tradiționale, cum ar fi casele de schimb, se aplică adesea reglementările naționale și internaționale. Pentru a putea schimba bitcoin pentru moneda dumneavoastră națională, vi se va solicita adesea să furnizați dovada identității și a informațiilor bancare. Utilizatorii ar trebui să știe că, odată ce o adresă bitcoin este atașată la o identitate, toate tranzacțiile bitcoin asociate sunt ușor de identificat și de urmărit. Aceasta este unul dintre motivele pentru care mulți utilizatori aleg să mențină conturile de la casele de schimb online neconectate cu portofelele lor.

**TIP**

Alice a intrat în contact cu bitcoin prin intermediul unui prieten, astfel că are o modalitate ușoară de a achiziționa primul ei bitcoin. În continuare, vom analiza cum cumpără bitcoin de la prietenul său Joe și cum Joe trimitе bitcoin în portofelul ei.

## Găsirea Prețului Curent al Bitcoin

Înainte ca Alice să poată cumpăra bitcoin de la Joe, trebuie să se pună de acord asupra *cursului de schimb* între bitcoin și dolarul american. Astfel apare o întrebare comună pentru cei noi cu bitcoin: "Cine stabilește prețul bitcoin?" Răspunsul scurt este că prețul este stabilit de piețe.

Bitcoin, la fel ca majoritatea altor monede, are un *curs de schimb fluctuant*. Asta înseamnă că valoarea bitcoin față de orice altă monedă fluctuează în funcție de cerere și ofertă pe diferitele piețe unde este tranzacționată. De exemplu "prețul" bitcoin în dolari americani este calculat pe fiecare piață în baza celui mai recent schimb între bitcoin și dolari americani. Ca atare, prețul tinde să fluctueze de mai multe ori pe secundă. Un serviciu de stabilire a prețurilor va agraga pre-

țurile de pe mai multe piețe și va calcula o medie ponderată în volum care reprezintă în mare cursul de schimb pe piață a unei perechi de valute (de exemplu, BTC/USD).

Există sute de aplicații și site-uri web care pot oferi cursul curent de schimb. Iată câteva dintre cele mai populare:

### **Bitcoin Average**

Un site care oferă o vizualizare simplă a mediei ponderate în volum pentru fiecare monedă.

### **CoinCap**

Un serviciu care listează capitalizarea de piață și cursul de schimb a sute de cripto-monede, inclusiv bitcoin.

### **Chicago Mercantile Exchange Bitcoin Reference Rate**

Un curs de referință care poate fi folosit ca referință instituțională și contractuală, furnizat ca parte a fluxurilor de date privind investițiile de către CME.

Pe lângă aceste diverse site-uri și aplicații, majoritatea portofelelor bitcoin vor converti automat sumele între bitcoin și alte monede. Joe își va folosi portofelul pentru a converti automat prețul înainte de a trimite bitcoin către Alice.

## **Trimiterea și Primirea Bitcoin**

Alice a decis să schimbe 10 dolari americani pentru bitcoin, pentru a nu risca prea mulți bani cu această nouă tehnologie. Ea îi dă 10 dolari în numerar lui Joe, deschide aplicația portofel Mycelium, și selectează Receive (Primește). Aplicația afișează un cod QR cu prima adresă bitcoin a lui Alice.

Apoi Joe selectează Send (Trimite) pe portofelul smartphone-ului său și îi este afișat un ecran care conține două intrări:

- O adresă bitcoin de destinație
- Suma de trimis, în bitcoin (BTC) sau în moneda sa locală (USD)

În câmpul pentru adresa bitcoin, există o pictogramă mică care arată ca un cod QR. Acest lucru îi permite lui Joe să scaneze codul de bare cu camera smartphone-ului său, astfel încât să nu fie necesar să introducă adresa bitcoin a lui Alice, care este destul de lungă și dificil de tastat. Joe atinge pictograma codului QR și activează camera smartphone-ului, apoi scană codul QR afișat pe smartphone-ul lui Alice.

Joe are acum setată adresa bitcoin a lui Alice ca destinatar. Joe introduce suma în valoare de 10 dolari americani, iar portofelul său o convertește accesând cel mai recent curs de schimb de la un serviciu online. Cursul de schimb la momentul respectiv este de 100 USD per bitcoin, deci 10 USD valorează 0.10 bitcoin (BTC), sau 100 milibitcoin (mBTC) așa cum se vede în captura de ecran a portofelului lui Joe (vezi [Ecranul de trimitere al portofelului mobil bitcoin Airbitz](#)).

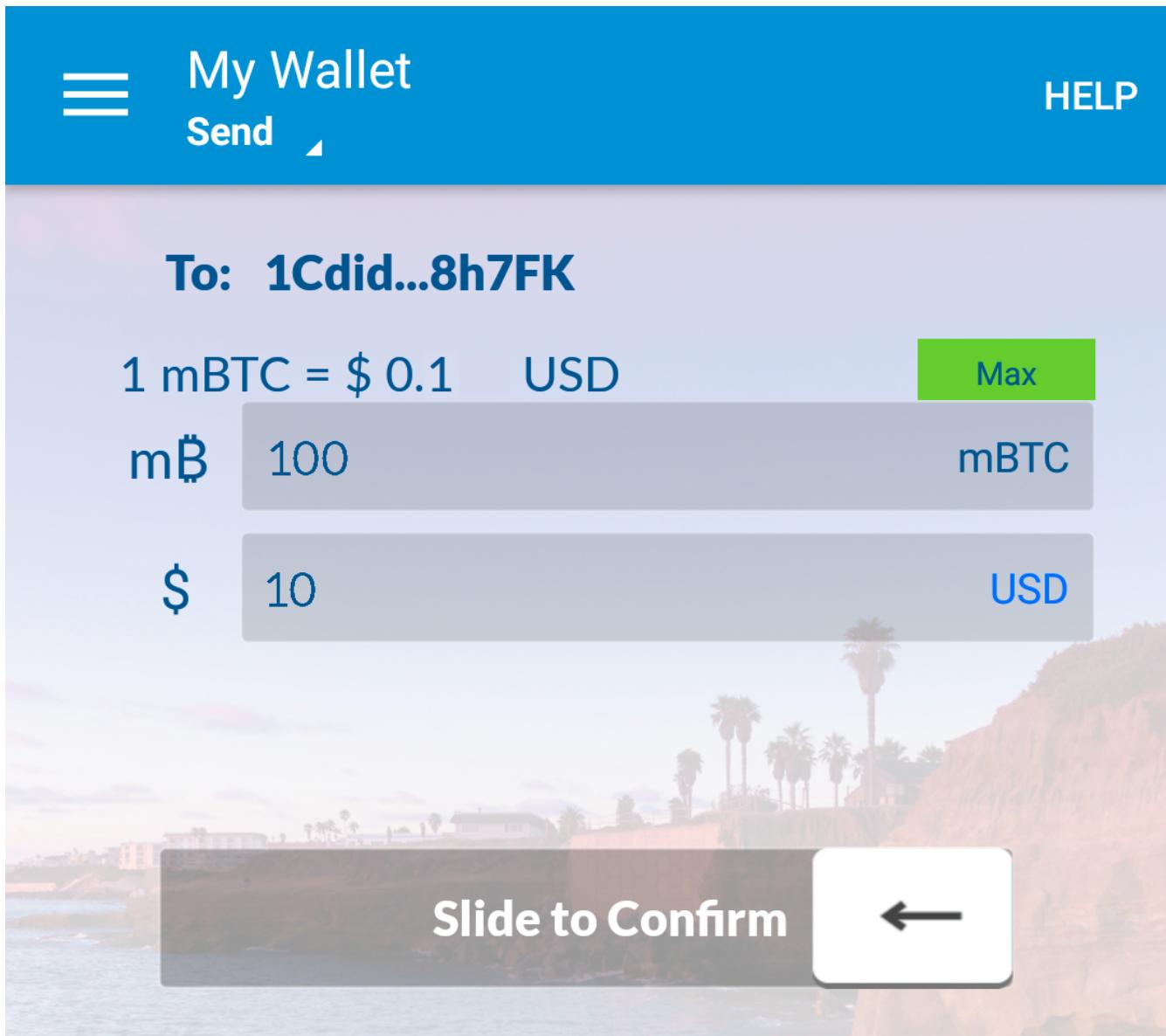


Figure 2. Ecranul de trimitere al portofelului mobil bitcoin Airbitz

Joe verifică apoi cu atenție pentru a se asigura că a introdus suma corectă, pentru că urmează să transmită bani, iar greșelile sunt ireversibile. După ce verifică de două ori adresa și suma, el apasă Send (Trimite) pentru a transmite tranzacția. Portofelul bitcoin mobil al lui Joe construiește o tranzacție care atribuie 0,10 BTC adresei furnizate de Alice, achiziționând fondurile din portofelul lui Joe și semnând tranzacția cu cheile private ale lui Joe. Acest lucru spune rețelei bitcoin că Joe a autorizat un transfer de valoare la noua adresă a lui Alice. Deoarece tranzacția este transmisă prin protocolul de-la-egal-la-egal (peer-to-peer), aceasta se propagă rapid în toată rețeaua bitcoin. În mai puțin de o secundă, majoritatea nodurilor bine conectate din rețea primesc tranzacția și văd adresa lui Alice pentru prima dată.

Între timp, portofelul lui Alice ”ascultă” în mod constant tranzacțiile publicate în rețeaua bitcoin, căutând orice tranzacție care se potrivește cu adresele din portofelul ei. La câteva secunde după ce portofelul lui Joe transmite tranzacția, portofelul lui Alice va indica că a primit 0.10 BTC.

## Confirmări

La început, adresa lui Alice va arăta tranzacția de la Joe drept "Neconfirmată". Aceasta înseamnă că tranzacția a fost propagată în rețea, dar încă nu a fost înregistrată în registrul de tranzacții bitcoin, cunoscut sub numele de blockchain (lanț-de-blocuri). Pentru a fi confirmată, o tranzacție trebuie inclusă într-un bloc și adăugată la lanțul-de-blocuri (blockchain), ceea ce se întâmplă în medie la fiecare 10 minute. În termeni financiari tradiționali, acest lucru este cunoscut sub numele de *compensare*. Pentru mai multe detalii despre propagarea, validarea și compensarea (confirmarea) tranzacțiilor bitcoin, consultați [Minerit și Consens](#).

Alice este acum deținătoarea a 0.10 BTC pe care îi poate cheltui. În capitolul următor, vom analiza prima ei achiziție cu bitcoin și vom examina mai în detaliu tehnologiile care stau la baza tranzacțiilor și propagării.

# Cum Funcționează Bitcoin

## Tranzacții, Blocuri, Minerit și Blockchain

Sistemul bitcoin, spre deosebire de sistemele tradiționale bancare și de plăți, se bazează pe o încredere descentralizată. În locul unei autorități centrale de încredere, în bitcoin, încrederea este obținută ca o proprietate emergentă din interacțiunile diferenților participanți în sistemul bitcoin. În acest capitol, vom examina bitcoin de la un nivel ridicat urmărind o singură tranzacție prin sistemul bitcoin și vom urmări cum devine "de încredere" și acceptată de către mecanismul bitcoin de consens distribuit și în final înregistrată pe lanțul-de-blocuri (blockchain), registrul distribuit al tuturor tranzacțiilor. Capitolele ulterioare vor aprofunda tehnologia din spatele tranzacțiilor, rețelei și mineritului.

## Prezentare Generală a Bitcoin

În diagrama de ansamblu prezentată în [Prezentare generală bitcoin](#), vedem că sistemul bitcoin este format din utilizatori cu portofele care conțin chei, tranzacții care sunt propagate în rețea și mineri care produc (prin calcul competitiv) lanțul-de-blocuri (blockchain-ul) consensual, care este registrul autoritar a tuturor tranzacțiilor.

Fiecare exemplu din acest capitol se bazează pe o tranzacție reală realizată în rețeaua bitcoin, simulând interacțiunile dintre utilizatori (Joe, Alice, Bob și Gopesh) prin trimitera de fonduri dintr-un portofel în altul. În timp ce urmărim o tranzacție prin rețeaua bitcoin către lanțul-de-blocuri, vom folosi un site *blockchain explorer* (explorator pentru lanțul-de-blocuri) pentru a vizualiza fiecare pas. Un explorator pentru lanțul-de-blocuri este o aplicație web care funcționează ca un motor de căutare bitcoin, prin faptul că vă permite să căutați adrese, tranzacții și blocuri și să vedeați relațiile și fluxurile dintre ele.

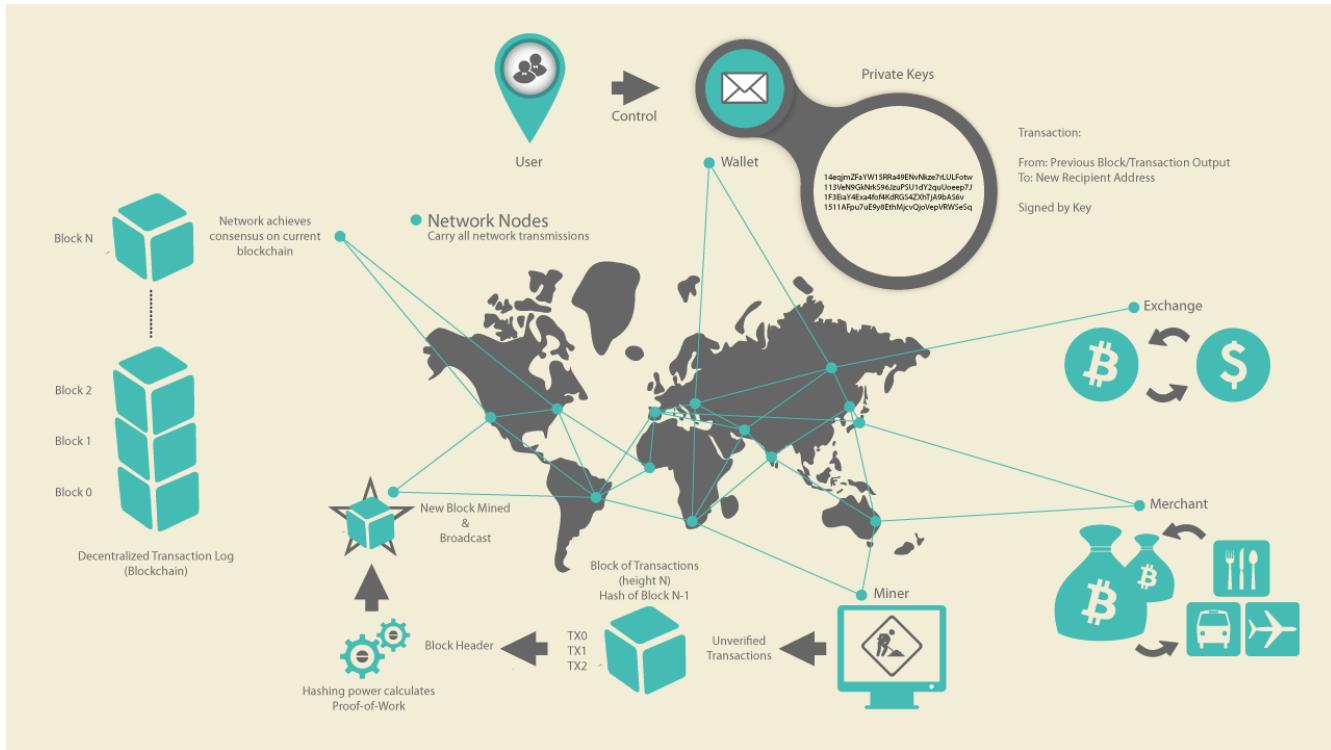


Figure 3. Prezentare generală bitcoin

Exploratorii populari pentru lanțul-de-blocuri (blockchain) includ:

- [BlockCypher Explorer](#)
- [blockchain.info](#)
- [BitPay Insight](#)
- [Blockstream Explorer](#)

Fiecare dintre aceștia are o funcție de căutare care acceptă ca intrare o adresă bitcoin, un rezumat de tranzacție, un număr de bloc sau un rezumat de bloc și returnează ca ieșire informațiile corespunzătoare din rețeaua bitcoin. Cu fiecare exemplu de tranzacție sau bloc, vom oferi o adresă URL, astfel încât să le puteți căuta și să le analizați în detaliu.

## Cumpărarea unei Cești de Cafea

Alice, prezentată în capitolul precedent, este un utilizator nou care tocmai a achiziționat primul ei bitcoin. În [Obținerea Primului Dumneavoastră Bitcoin](#), Alice s-a întâlnit cu prietenul ei Joe pentru a schimba bani în bitcoin. Tranzacția creată de Joe a alimentat portofelul lui Alice cu 0,10 BTC. Acum Alice va face prima ei tranzacție, cumpărând o ceașcă de cafea de la cafeneaua lui Bob din Palo Alto, California.

Cafeneaua lui Bob a început recent să accepte plăți bitcoin prin adăugarea unei opțiuni bitcoin în sistemul său de punct de vânzare. Prețurile la cafeneaua lui Bob sunt listate în moneda locală (dolari americanii), dar la teajhea, clienții au opțiunea de a plăti fie în dolari, fie în bitcoin. Alice plasează comanda pentru o ceașcă de cafea iar Bob o introduce în registru, aşa cum face pentru toate tranzacțiile. Sistemul de vânzare convertește automat prețul total din dolari americanii la cursul de schimb existent și afișează prețul în ambele monede:

Total:  
\$1.50 USD  
0.015 BTC

Bob spune: "Costă un dolar-cincizeci sau cincisprezece milibităi".

De asemenea, sistemul de vânzări al lui Bob va crea automat un cod QR special care conține o *solicitare de plată* (vezi [Cod QR pentru solicitare de plată](#)).

Spre deosebire de un codul QR care conține doar o adresă bitcoin de destinație, o solicitare de plată este o adresă URL codificată în codul QR care conține o adresă de destinație, o sumă de plată și o descriere generică, cum ar fi "Cafeneaua lui Bob". Acest lucru permite unei aplicații portofel bitcoin să completeze automat informațiile utilizate pentru a trimite plata și în același timp să arate o descriere care poate fi citită cu ușurință de către utilizator. Puteți scana codul QR cu o aplicație portofel bitcoin pentru a observa ce va vedea Alice.



Figure 4. Cod QR pentru solicitare de plată

**TIP**

Încercați să scănați acest cod cu portofelul pentru a vedea adresa și suma, dar NU TRIMITEȚI BANI.

*Codul QR al solicitării de plată codifică următoarea adresă URL, definită în BIP-21:*

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

Componentele URL-ului

O adresă bitcoin: "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"  
Suma de plată: "0.015"

O etichetă pentru adresa destinatarului: "Bob's Cafe"  
O descriere a plății: "Purchase at Bob's Cafe"

Alice folosește smartphone-ul pentru a scana codul de bare afișat. Smartphone-ul ei arată o plată de 0.0150 BTC la Bob's Cafe și ea selectează Send (Trimite) pentru a autoriza plata. În câteva secunde (aproximativ aceeași perioadă de timp ca și o autorizație cu cardul), Bob vede tranzacția în

registru și o finalizează.

În secțiunile următoare, vom examina această tranzacție mai detaliat. Vom vedea cum portofelul lui Alice a construit-o, cum a fost propagată în rețea, cum a fost verificată și, în final, cum Bob poate cheltui suma respectivă în tranzacții ulterioare.

#### NOTE

Rețea bitcoin poate tranzacționa în valori fractionale, de exemplu, de la milibitcoin (1/1000 dintr-un bitcoin) până la 1/100.000.000 de bitcoin, care este cunoscut sub numele de satoshi. De-a lungul acestei cărți, vom folosi termenul "bitcoin" pentru a ne referi la orice cantitate de monedă bitcoin, de la cea mai mică unitate (1 satoshi) la numărul total (21.000.000) de bitcoin care poate fi minerit vreodată.

Puteți examina tranzacția lui Alice cu Cafeneaua lui Bob în lanțul-de-blocuri (blockchain) folosind un site de explorator blocuri ([Vizualizați tranzacția lui Alice la: blockchain.info](#)):

*Example 1. Vizualizați tranzacția lui Alice la: [blockchain.info](#)*

```
https://blockchain.info/tx/0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a5728  
6c345c2f2
```

## Tranzacții Bitcoin

În termeni simpli, o tranzacție spune rețelei că deținătorul unei anumite valori bitcoin a autorizat transferul acelei valori către un alt proprietar. Noul proprietar poate acum cheltui bitcoin prin crearea unei alte tranzacții care autorizează transferul către un alt proprietar și așa mai departe, într-un lanț de proprietate.

### Intrările și ieșirile unei Tranzacții

Tranzacțiile sunt ca niște linii într-un registru de evidență cu două intrări. Fiecare tranzacție conține una sau mai multe "intrări", care sunt precum debiturile pentru un cont bitcoin. De cealaltă parte a tranzacției, există una sau mai multe "ieșiri", care sunt precum creditele adăugate într-un cont bitcoin. Intrările și ieșirile (debite și credite) nu însumează neapărat aceeași sumă. În schimb, ieșirile însumează mai puțin decât intrările, iar diferența reprezintă un *comision de tranzacție* implicit, care este o mică plată colectată de către minerul care include tranzacția în registru. O tranzacție bitcoin este prezentată ca o înregistrare în evidența contabilă în [Tranzacția ca evidență contabilă cu intrare dublă](#).

Tranzacția conține, de asemenea, dovada dreptului de proprietate pentru fiecare sumă de bitcoin (intrări) a cărei valoare este cheltuită, sub forma unei semnături digitale de la proprietar, care poate fi validată independent de către oricine. În termeni bitcoin "a cheltui" înseamnă a semna o tranzacție care transferă valoare de la o tranzacție anterioară către un nou proprietar identificat printr-o adresă bitcoin.

Figure 5. Tranzacția ca evidență contabilă cu intrare dublă

## Lanțuri de Tranzacție

Plata lui Alice către Cafeneaua lui Bob utilizează ca intrare o ieșire a unei tranzacții anterioare. În capitolul precedent, Alice a primit bitcoin de la prietenul ei Joe în schimbul dolarilor. Acea tranzacție a creat o valoare bitcoin blocată de cheia lui Alice. Noua ei tranzacție cu Cafeneaua lui Bob face referire la tranzacția anterioară drept intrare și creează noi ieșiri pentru a plăti ceașca de cafea și pentru a primi restul. Tranzacțiile formează un lanț, în care intrările din ultima tranzacție corespund ieșirilor tranzacțiilor anterioare. Cheia lui Alice furnizează semnătura care deblochează acele ieșiri ale tranzacțiilor anterioare, dovedind astfel rețelei bitcoin că ea deține fondurile. Ea atașează plata pentru cafea la adresa lui Bob, "restricționând" acea ieșire cu cerința ca Bob să producă o semnătură pentru a cheltui acea sumă. Acesta reprezintă un transfer de valoare între Alice și Bob. Acest lanț de tranzacții, de la Joe la Alice până la Bob, este ilustrat în [Un lanț de tranzacții](#), în care ieșirea unei tranzacții este intrarea următoarei tranzacții.

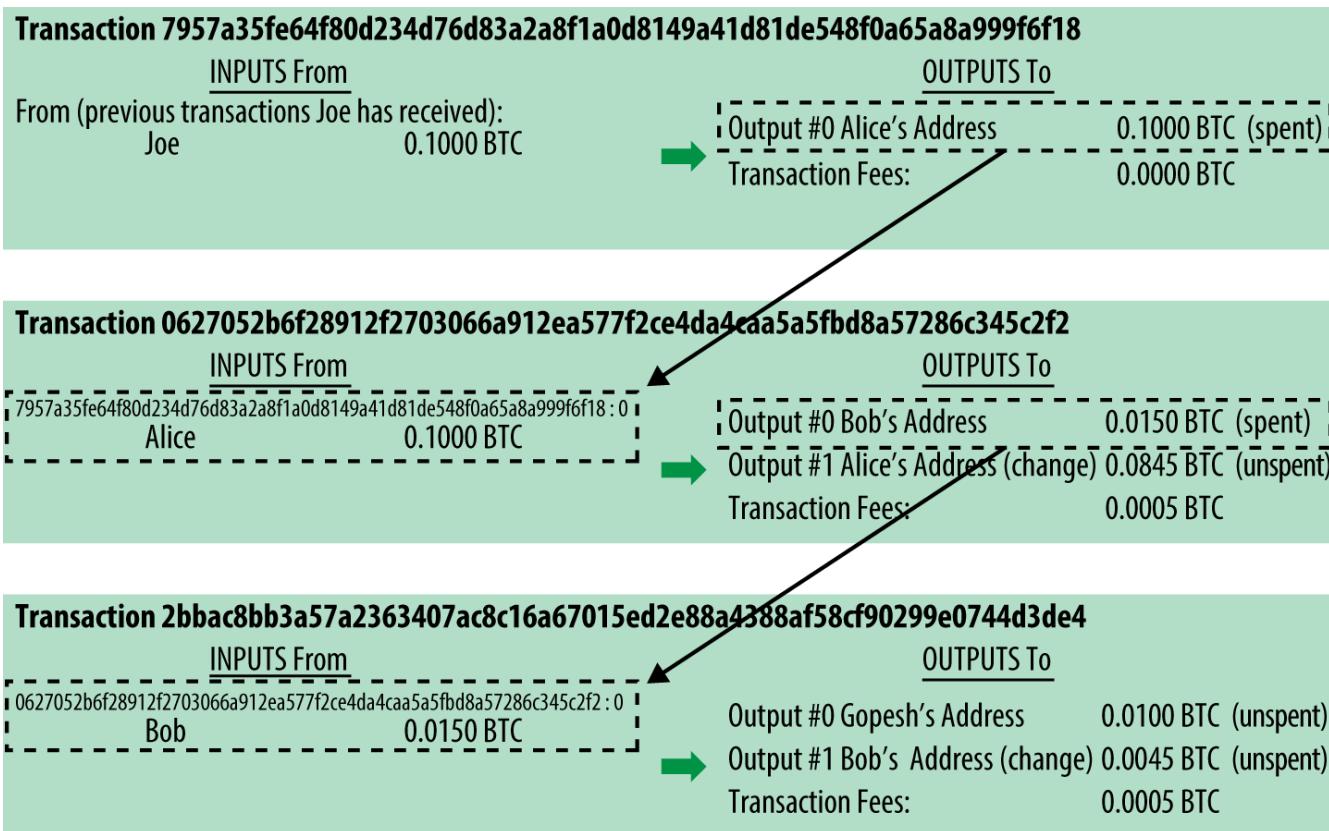


Figure 6. Un lanț de tranzacții, în care ieșirea unei tranzacții este intrarea următoarei tranzacții

## Primirea Restului

Multe tranzacții bitcoin vor include ieșiri care se referă atât la o adresă a noului proprietar, cât și la o adresă a proprietarului curent, numită adresă pentru *rest*. Acest lucru se datorează faptului că intrările tranzacției, la fel ca bancnotele, nu pot fi divizate. Dacă achiziționați un articol de 5 dolari dintr-un magazin, dar utilizați o banctontă în valoare de 20 de dolari pentru a plăti articolul, vă așteptați să primiți rest 15 dolari. Același concept se aplică intrărilor tranzacțiilor bitcoin. Dacă ați achiziționat un articol care costă 5 bitcoin, dar ați avut doar o intrare de 20 de bitcoin, veți trimite o ieșire de 5 bitcoin proprietarului magazinului și o ieșire de 15 bitcoin înapoi ca rest (exceptând cazul când se aplică comisioane pentru tranzacție).

Diferite portofele pot utiliza diferite strategii atunci când agregă intrări pentru a efectua o plată solicitată de utilizator. Acestea ar putea agrupa multe intrări mici sau pot utiliza una egală sau mai mare decât plata dorită. Cu excepția cazului în care portofelul poate agrupa intrări în așa fel încât să corespundă exact plății dorite, plus comisioanele de tranzacție, portofelul va trebui să genereze niște rest. Acest lucru este foarte similar cu modul în care oamenii lucrează cu numerar. Dacă utilizați întotdeauna cea mai mare bancnotă din buzunar, veți avea un buzunar plin de mărunțis. Dacă utilizați doar mărunțis, veți avea întotdeauna doar bancnote mari. Oamenii găsesc în mod inconștient un echilibru între aceste două extreme, iar dezvoltatorii de portofele bitcoin se străduiesc să programeze acest echilibru.

Pe scurt, *tranzacțiile* mută valoare de la *intrările tranzacției* la *ieșirile tranzacției*. O intrare este o referință la ieșirea unei tranzacții anterioare, aratănd de unde vine valoarea. Ieșirea unei tranzacții direcționează o valoare specifică la adresa unui nou proprietar și poate include o ieșire pentru rest înapoi la proprietarul initial. Ieșirile de la o tranzacție pot fi folosite ca intrări într-o nouă tranzacție, creând astfel un lanț de proprietate, valoarea fiind mutată de la un proprietar la altul (vezi [Un](#)

lanț de tranzacții, în care ieșirea unei tranzacții este intrarea următoarei tranzacții).

## Forme Uzuale de Tranzacții

Cea mai uzuală formă de tranzacție este o plată simplă de la o adresă la alta, care include adesea niște "rest" înapoiat proprietarului original. Acest tip de tranzacție are o intrare și două ieșiri și este prezentat în [Cele mai uzuale tranzacții](#).

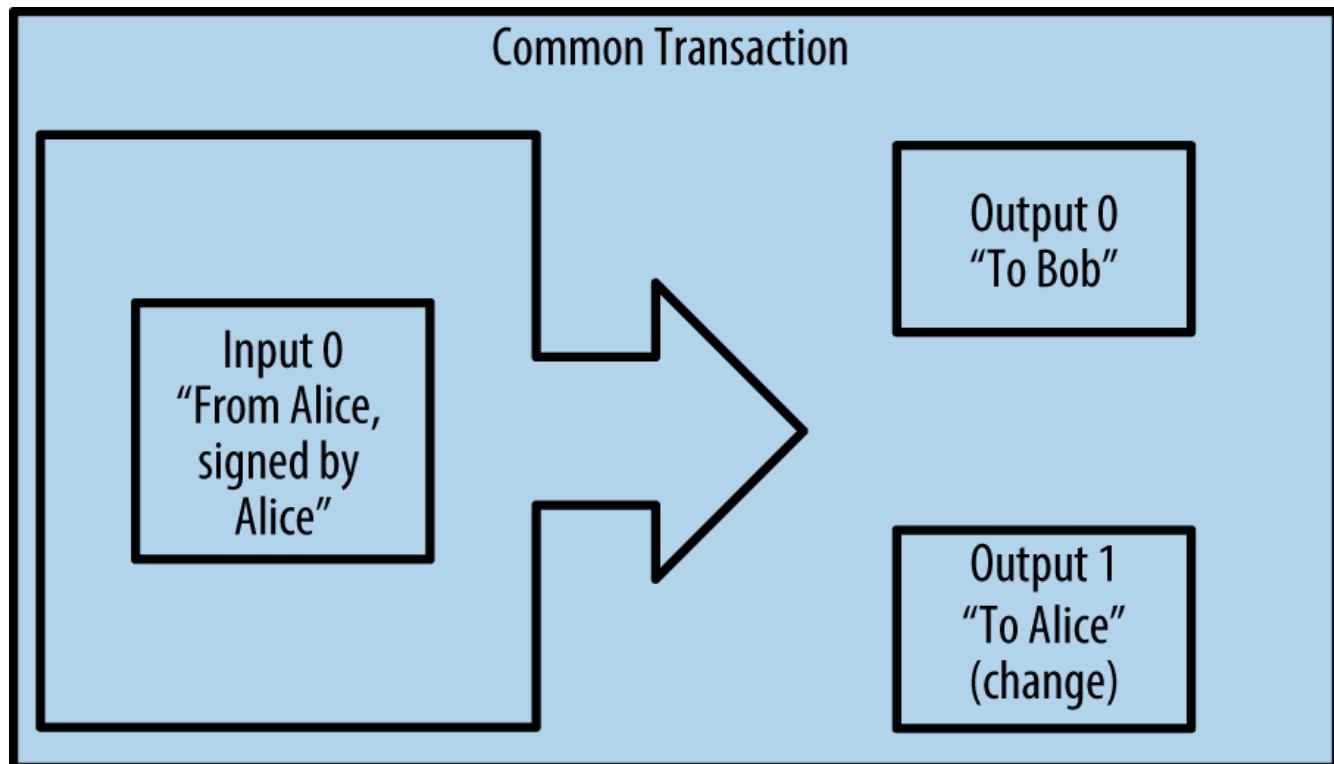


Figure 7. Cele mai uzuale tranzacții

O altă formă uzuală de tranzacție este una care agregă câteva intrări într-o singură ieșire (vezi [Tranzacție care agregă fonduri](#)). Aceasta reprezintă echivalentul în lumea reală a schimbării unei monede și bancnote pentru o singură bancnotă mai mare. Tranzacțiile de felul acesta sunt uneori generate de aplicația portofel pentru a curăța multe sume mai mici care au fost primite ca rest.

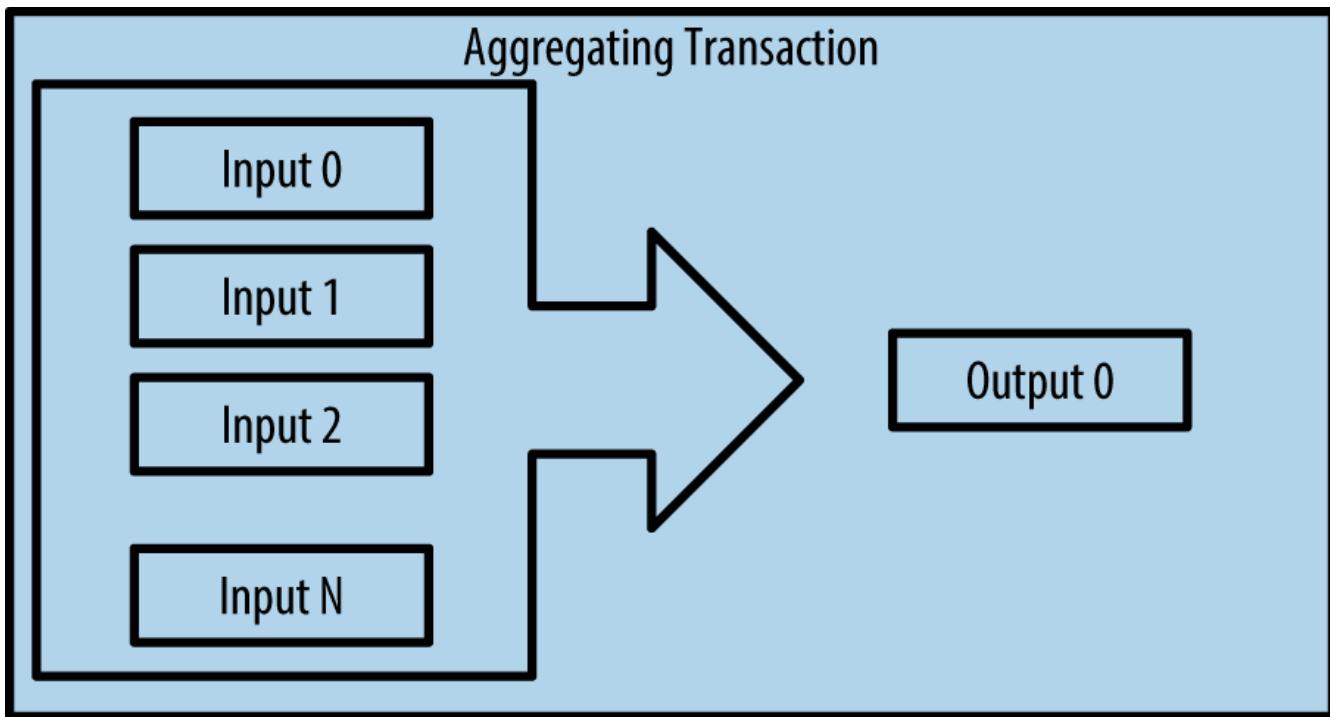


Figure 8. Tranzacție care agregă fonduri

În cele din urmă, o altă formă de tranzacție care e des întâlnită pe registrul bitcoin este o tranzacție care distribuie o intrare la mai multe ieșiri, reprezentând mai mulți destinatari (vezi [Tranzacții care distribuie fonduri](#)). Acest tip de tranzacție este folosit uneori de entități comerciale pentru a distribui fonduri, cum ar fi atunci când se prelucrează plăți salariale către mai mulți angajați.

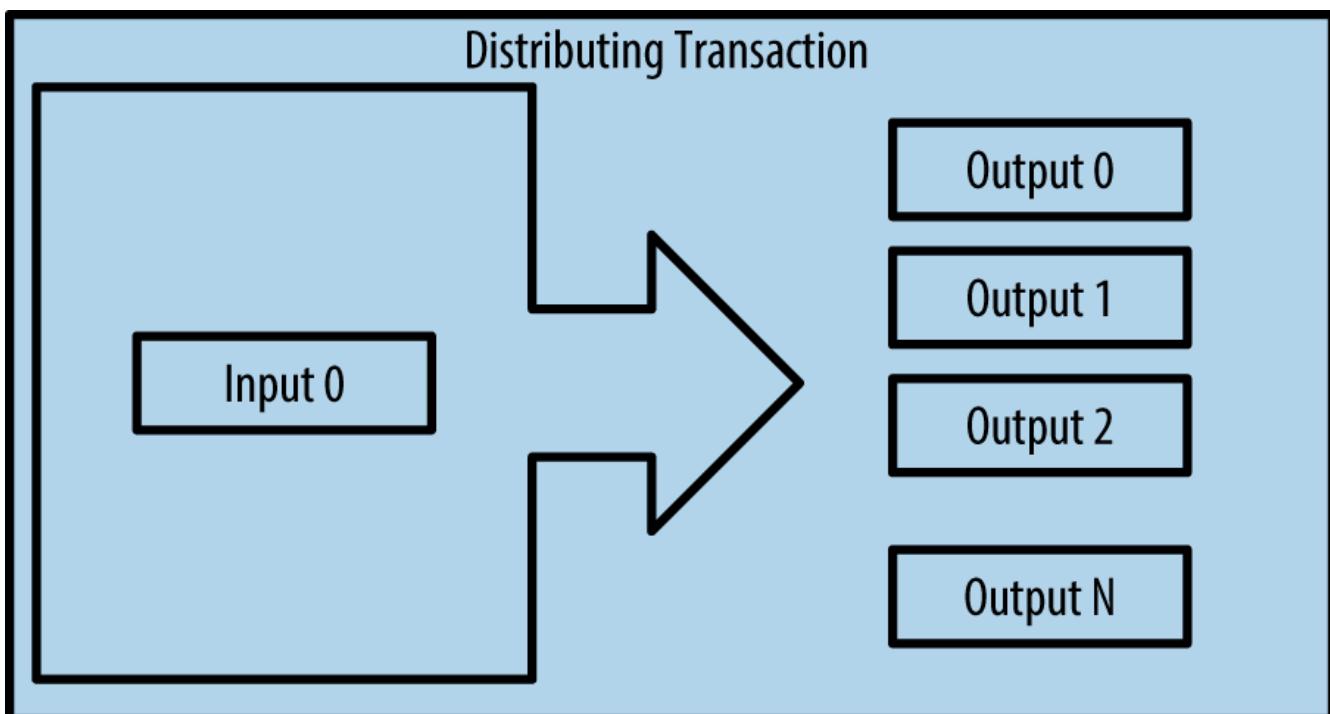


Figure 9. Tranzacții care distribuie fonduri

## Construirea unei tranzacții

Aplicația portofel a lui Alice conține toată logica necesară pentru a selecta intrările și ieșirile corespunzătoare în scopul de a construi o tranzacție conform specificațiilor lui Alice. Ea trebuie să

specifice doar o destinație și o sumă, iar restul se întâmplă în aplicația portofel fără ca Alice să vadă detaliile. E important de ținut minte că o aplicație portofel poate construi tranzacții chiar dacă este complet offline. La fel cum ați scrie un cec acasă iar mai târziu l-ați trimis la bancă într-un plic, nu e nevoie ca tranzacția să fie construită și semnată în timp ce sunteți conectat la rețeaua bitcoin.

## Obținerea Intrărilor Corecte

Aplicația portofel a lui Alice va trebui mai întâi să găsească intrările care pot plăti suma pe care vrea să o trimită lui Bob. Majoritatea portofelelor urmăresc toate ieșirile disponibile aparținând adreselor din portofel. Prin urmare portofelul lui Alice ar conține o copie a ieșirilor de la tranzacția lui Joe, care a fost creată la schimb pentru numerar (vezi [Obținerea Primului Dumneavoastră Bitcoin](#)). O aplicație portofel bitcoin care rulează ca și client nod-complet conține, de fapt, o copie a fiecărei ieșiri necheltuite de la fiecare tranzacție din lanțul-de-blocuri. Aceasta permite unui portofel să construiască intrările tranzacției, precum și să verifice rapid că tranzacțiile primite au intrările corecte. Totuși, pentru că un client nod-complet ocupă mult spațiu pe disc, majoritatea utilizatorilor aleg să ruleze clienti "supli" (lightweight) care pot urmări doar ieșirile necheltuite ale utilizatorului.

Dacă aplicația portofel nu păstrează o copie a ieșirilor necheltuite ale tranzacției, ea poate interoga rapid rețeaua bitcoin pentru a primi această informație folosind o varietate de API-uri oferite de diferiți furnizori sau interogând un nod-complet folosind un apel la interfața de programare a aplicației (API). [Identifică toate ieșirile necheltuite pentru adresa bitcoin a lui Alice](#) arată o solicitare API construită ca o comandă HTTP GET la un URL specific. Această URL va returna pentru o adresă toate ieșirile necheltuite ale tranzacției. Vom folosi clientul HTTP *cURL* din linia de comandă pentru a primi răspunsul.

*Example 2. Identifică toate ieșirile necheltuite pentru adresa bitcoin a lui Alice*

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

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

Răspunsul la [Identifică toate ieșirile necheltuite pentru adresa bitcoin a lui Alice](#) arată o ieșire necheltuită (una care nu a fost încă folosită) în proprietatea adresei lui Alice `1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK`. Răspunsul include referința la tranzacția în care este conținută această ieșire necheltuită (plata de la Joe) și valoarea ei în statoshi, la 10 milioane, echivalentul a 0,10 bitcoin. Cu aceste informații aplicația portofel a lui Alice poate construi o tranzacție pentru a transfera acea valoare la adresa noului proprietar.

**TIP** | [Vedeți tranzacție de la Joe la Alice.](#)

După cum puteți vedea, portofelul lui Alice conține destul bitcoin într-o singură ieșire necheltuită pentru a plăti ceașca de cafea. Dacă nu ar fi fost cazul, aplicația portofel a lui Alice ar fi fost nevoie să "scotocească" printr-o grămadă de ieșiri necheltuite, ca atunci când căutam monede într-o poșetă, până când ar fi putut găsi destule pentru a plăti pentru cafea. În ambele cazuri, s-ar putea să fie necesar să primească rest, ceea ce vom vedea în secțiunea următoare, când aplicația portofel creează ieșirile tranzacției (plăți).

## Creearea ieșirilor

Ieșirea unei tranzacții este creată sub forma unui script care creează o restricție pentru valoare, și poate fi încasată prin prezentarea unei soluții la script. În termeni simpli, ieșirea tranzacției lui Alice va conține un script care spune ceva de genul, "Această ieșire este plătibilă oricui poate prezenta o semnătură de la cheia care corespunde adresei lui Bob". Deoarece doar Bob are portofelul cu cheile care corespund acelei adrese, doar portofelul lui Bob poate să prezinte o asemenea semnătură pentru a încasa această ieșire. Drept urmare, Alice va "restrictiona" valoarea ieșirii cu cerința unei semnături de la Bob.

Această tranzacție va include de asemenea o a doua ieșire, pentru că fondurile lui Alice sunt sub forma unei ieșiri de 0,10 BTC, prea mulți bani pentru cei 0,015 BTC cât costă ceașca de cafea. Alice va avea nevoie de 0,085 BTC rest. Plata pentru restul lui Alice este creată de către portofelul lui Alice sub forma unei ieșiri chiar în aceeași tranzacție cu plata pentru Bob. Practic, portofelul lui Alice împarte fondurile ei în două plăți: una către Bob și una înapoi către ea însăși. Ea poate apoi

folosi (cheltui) ieșirea rest într-o tranzacție ulterioară.

În cele din urmă, pentru ca tranzacția lui Alice să fie procesată de către rețea în timp util, aplicația portofel a lui Alice va adăuga un mic comision. Acesta nu este explicit în tranzacție; este implicit în diferența dintre intrări și ieșiri. În loc să ceară rest 0,085, Alice creează doar 0,0845 pentru a doua ieșire, vor fi 0,0005 BTC (jumătate de milibitcoin) rămași. Cei 0,10 BTC ai intrării nu vor fi cheltuiți în întregime pe cele două ieșiri, pentru că ieșirile vor însuma mai puțin de 0,10. Diferența rezultată este *comisionul de tranzacție* care este colectat de către miner drept comision pentru validarea și includerea tranzacției într-un block ca parte a lanțului-de-blocuri.

Tranzacția rezultată poate fi văzută folosind o aplicație explorator lanț-de-blocuri web (blockchain explorer), așa cum se vede în [Tranzacția lui Alice către Cafeneaua lui Bob](#).

## Transaction View information about a bitcoin transaction

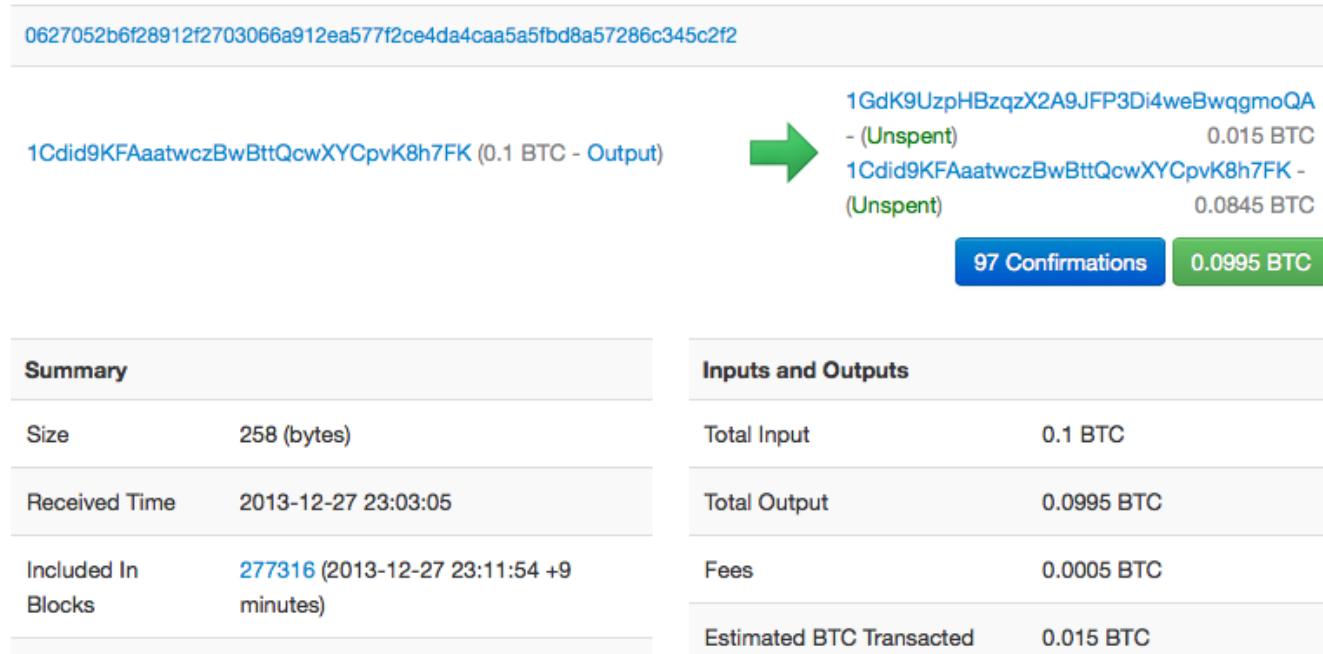


Figure 10. Tranzacția lui Alice către Cafeneaua lui Bob

**TIP** Vedeți [tranzacția lui Alice către Cafeneaua lui Bob](#).

## Adăugarea Tranzacției în Registrul

Tranzacția creată de aplicația portofel a lui Alice are o lungime de 258 octeți și conține tot ceea ce este necesar pentru a confirma deținerea fondurilor și pentru a desemna noii proprietari. Acum, tranzacția trebuie transmisă către rețeaua bitcoin unde va deveni parte a lanțului-de-blocuri. În secțiunea următoare vom vedea cum o tranzacție devine parte a unui nou bloc și cum blocul este "minerit". În cele din urmă, vom vedea cum noul bloc, odată adăugat în lanțul-de-blocuri, devine din ce în ce mai de încredere pentru rețea pe măsură ce noi blocuri sunt adăugate.

### Transmiterea tranzacției

Deoarece tranzacția conține toate informațiile necesare procesării, nu contează cum sau unde este transmisă către rețeaua bitcoin. Rețeaua bitcoin este o rețea de-la-egal-la-egal (peer-to-peer), cu

fiecare client bitcoin participând prin conectarea la alți clienți bitcoin. Scopul rețelei bitcoin este să propage tranzacții și blocuri la toți participanții.

## Cum se propagă

Orice sistem, cum ar fi un server, o aplicație desktop, sau un portofel, care participă în rețeaua bitcoin și care ”vorbește” protocolul bitcoin este numit *nod bitcoin*. Aplicația portofel a lui Alice poate să trimită tranzacții noi către orice nod bitcoin la care este conectată prin orice tip de conexiune: cablu, WiFi, mobilă, etc. Portofelul ei bitcoin nu trebuie să se conecteze direct la portofelul bitcoin a lui Bob și ea nu trebuie să folosească conexiunea la internet oferită de către cafenea, deși ambele opțiuni sunt posibile și ele. Orice nod bitcoin care primește o tranzacție validă pe care nu a mai vazut-o înainte o va transmite imediat tuturor celorlalte noduri cu care este conectat, un mecanism de propagare cunoscut sub numele de *inundare*. Astfel, tranzacția se propagă rapid în toată rețeaua de-la-egal-la-egal, ajungând la un procent ridicat de noduri în câteva secunde.

## Perspectiva lui Bob

Dacă aplicația portofel a lui Bob este conectată direct la aplicația portofel a lui Alice, este posibil ca aplicația portofel a lui Bob să fie primul nod care primește tranzacția. Totuși, chiar dacă portofelul lui Alice trimite tranzacția prin alte noduri, aceasta va ajunge la portofelul lui Bob în câteva secunde. Portofelul lui Bob va identifica imediat tranzacția lui Alice ca o plată de intrare deoarece conține ieșiri care pot fi încasate cu cheile lui Bob. Aplicația portofel a lui Bob poate și să verifice independent că tranzacția este formată corect, că folosește ieșiri anterioare necheltuite, și că ea conține un comision de tranzacție suficient de mare pentru a fi inclusă în următorul bloc. În acest moment, Bob poate presupune, cu un risc scăzut, că tranzacția va fi inclusă în scurt timp într-un bloc și apoi confirmată.

O concepție greșită despre tranzacțiile bitcoin este că ele trebuie să fie ”confirmate” săptămând 10 minute pentru un bloc nou, sau până la 60 de minute pentru șase confirmări complete. Deși confirmările asigură că tranzacția a fost acceptată de către toată rețeaua, o asemenea întârziere nu este necesară pentru articole de valoare mică cum ar fi o ceașcă de cafea. Un comerciant poate accepta o tranzacție de valoare mică fără nici o confirmare, fară a avea un risc mai mare decât o plată cu cardul de credit fără un act de identificare sau semnatură, cum comercianții acceptă în mod curent astăzi.

## Minerit de Bitcoin

Tranzacția lui Alice este acum propagată în rețeaua bitcoin. Nu devine parte a *lanțului-de-blocuri* până când nu este verificată și inclusă într-un bloc printr-un process numit *minerit*. Vezi [Minerit și Consens](#) pentru o explicație detaliată.

Sistemul de încredere al bitcoin se bazează pe calcul. Tranzacțiile sunt grupate în *blocuri*, care necesită o cantitate enormă de calcul pentru a fi construite într-un mod valid, însă doar o cantitate mică de calcul pentru a verifica dacă sunt valide. Procesul de minerit are două scopuri în bitcoin:

- Nodurile de minerit validează toate tranzacțiile făcând referire la *regulile de consens*. Prin

urmare, mineritul asigură securitatea pentru tranzacțiile bitcoin respingând tranzacțiile invalide sau care au anomalii.

- Mineritul creează bitcoin nou la fiecare bloc, aproape ca o bancă centrală care tipărește bani noi. Cantitatea de bitcoin creată per bloc este limitată și se diminuează cu timpul, urmând un program de emitere fix.

Mineritul realizează un echilibru fin între cost și recompensă. Mineritul necesită electricitate pentru a rezolva o problemă matematică. Un miner care are succes va colecta o *recompensă* sub forma de bitcoin nou și comisioane de tranzacție. Totuși, recompensa va fi colectată doar dacă minerul a validat corect tranzacțiile, în conformitate cu regulile de *consens*. Această balanță delicată oferă securitate pentru bitcoin fără o autoritate centrală.

O modalitate bună de a descrie mineritul este asemenea un joc competitiv de sudoku uriaș care se resetează de fiecare dată când cineva găsește o soluție și a cărui dificultate se ajustează automat astfel încât să dureze aproximativ 10 minute pentru a găsi o soluție. Imagineați-vă un puzzle sudoku uriaș, cu dimensiunea de câteva mii de rânduri și coloane. Dacă vă arăta un puzzle completat, îl puteți verifica destul de repede. Totuși, dacă puzzelul are câteva pătrățele completate și restul sunt goale, este nevoie de multă muncă pentru a fi rezolvat! Dificultatea jocului de sudoku poate fi ajustată schimbându-i dimensiunile (mai multe sau mai puține rânduri și coloane), dar poate fi totuși verificat destul de ușor chiar dacă este foarte mare. "Puzzle-ul" folosit în bitcoin se bazează pe un rezumat (hash) criptografic și prezintă caracteristici similare: există o asimetrie între cât este de greu de rezolvat și cât este de ușor de verificat, iar dificultatea poate fi ajustată.

În [Utilizările Bitcoin, Utilizatorii, și Poveștile Lor](#), l-am introdus pe Jing, un antreprenor din Shanghai. Jing operează o *fermă de minerit*, care este o afacere ce operează mii de calculatoare specializate în minerit, concurând pentru recompensă. La aproximativ fiecare 10 minute, calculatoarele de minerit ale lui Jing concurează împotriva a mii de sisteme similare într-o cursă globală pentru a găsi o soluție la un bloc de tranzacții. Găsirea unei asemenea soluții, așa numita *Dovadă-de-Lucru* (Proof-of-Work), necesită cvadrilaje de operații de rezumare (hashing) pe secundă în întreaga rețea bitcoin. Algoritmul pentru Dovadă-de-Lucru implică rezumarea (hashing-ul) antetului blocului împreună cu un număr aleator, folosind algoritmul criptografic SHA256 până este găsită o soluție care se potrivește cu un şablon predefinit.

Jing a început mineritul în 2010 folosind un calculator desktop foarte rapid pentru a găsi o Dovadă-de-Lucru potrivită pentru blocuri noi. Pe măsură ce mai mulți mineri s-au alăturat rețelei bitcoin, dificultatea problemei de rezolvat a crescut rapid. În curând, Jing și alții mineri au avansat la hardware mai specializat, cu plăci dedicate de procesare grafică (GPU) ca cele folosite la jocurile pe desktop sau consolă. La momentul redactării, dificultatea este atât de ridicată încât mineritul este profitabil doar folosind circuite integrate specifice aplicației (application-specific integrated circuits - ASIC), în esență sute de algoritmi de minerit întipăriți în hardware, rulând în paralel pe un singur chip de silicon. Compania lui Jing participă de asemenea într-un *bazin de minerit* care, asemenea unui grup de oameni care joacă la loterie, permite mai multor participanți să își împartă eforturile și recompensele. Compania lui Jing acum operează un depozit care conține mii de mineri ASIC pentru a mina bitcoin 24 de ore pe zi. Compania plătește costrurile pentru electricitate vânzând bitcoin-ul pe care îl generează din minerit, obținând un profit din venituri.

# Mineritul Tranzacțiilor în Blocuri

Tranzacții noi ajung constant în rețea de la portofelele utilizatorilor și de la alte aplicații. Imediat ce sunt văzute de către nodurile bitcoin din rețea, ele sunt adăugate într-un bazin de tranzacții neverificate, menținut de fiecare nod. Pe măsură ce minerii construiesc un nou bloc, ei adaugă tranzacții neverificate din acest bazin la noul bloc și apoi încearcă să dovedească validitatea acestui nou bloc, cu algoritmul de minerit (Dovadă-de-Lucru). Procesul de minerit este explicat în detaliu în [Minerit și Consens](#).

Tranzacțiile sunt adăugate la noul bloc, prioritizate descrescător după valoarea comisionului și după alte câteva criterii. Fiecare miner începe procesul de minerit al unui nou bloc imediat ce primește blocul precedent de la rețea, cunoscând că a pierdut precedentă runda a competiției. El creează imediat un nou bloc, îl umple cu tranzacții și cu amprenta blocului precedent, și începe să calculeze Dovada-de-Lucru pentru noul bloc. Fiecare miner include o tranzacție specială în blocul lui, una care efectuează o plată la propria sa adresă bitcoin pentru recompensa blocului (în prezent 6,25 bitcoin nou creați) plus suma comisioanelor de tranzacție de la toate tranzacțiile incluse în bloc. Dacă găsește o soluție care face blocul valid, el "câștigă" recompensa pentru că blocul lui este adăugat la lanțul-de-blocuri global și tranzacția recompensă pe care a inclus-o devine cheltuibilă. Jing, care participă într-un bazin de minerit, și-a configurat software-ul să creeze noi blocuri care atribuie recompensa unei adrese a bazinului. De acolo, o parte din recompensă este distribuită lui Jing și altor mineri în proporție cu cantitatea de lucru cu care au contribuit la runda precedentă.

Tranzacția lui Alice a fost preluată de rețea și inclusă în bazinul de tranzacții neverificate. Odată validată de software-ul de minerit, aceasta a fost inclusă într-un bloc nou, numit *bloc candidat*, generat de bazinul de minerit a lui Jing. Toți minerii care participă în acel bazin de minerit au început imediat să calculeze Dovada-de-Lucru pentru blocul candidat. La aproximativ cinci minute după ce tranzacția a fost transmisă de portofelul lui Alice, unul din minerii ASIC a lui Jing a găsit o soluție pentru blocul candidat și l-a anunțat în rețea. Odată ce ceilalți mineri au validat blocul câștigător, ei încep cursa să genereze blocul următor.

Blocul câștigător a lui Jing a devenit parte din lanțul-de-blocuri ca fiind blocul #277316, conținând 419 tranzacții, inclusiv tranzacția lui Alice. Blocul care conține tranzacția lui Alice este socotit ca o (una) "confirmare" a acelei tranzacții.

**TIP** Puteti vedea blocul care include [tranzacția lui Alice](#).

Aproximativ 19 minute mai târziu, un nou bloc, #277317, este minat de un alt miner. Pentru că acest nou bloc este construit peste blocul #277316 care conținea tranzacția lui Alice, a adăugat și mai multă putere de calcul lanțului-de-blocuri, consolidând astfel încrederea în acele tranzacții. Fiecare bloc minat peste cel care conține tranzacția se numără ca o confirmare pentru tranzacția lui Alice. Pe măsură ce blocurile se acumulează unul peste altul, devine exponențial mai grea inversarea tranzacției, facând-o astfel din ce în ce mai de încredere pentru rețea.

În diagrama [Tranzacția lui Alice inclusă în blocul #277316](#), putem vedea blocul #277316, care conține tranzacția lui Alice. Sub acesta sunt 277316 blocuri (incluzând blocul #0), legat unul de altul într-un lanț-de-blocuri (blockchain) până la blocul #0, cunoscut ca *blocul geneză*. Cu timpul, pe măsură ce "înalțarea" blocurilor crește, la fel se întâmplă și cu dificultatea de calcul pentru fiecare

bloc și pentru lanț ca întreg. Blocurile minate după cel care conține tranzacția lui Alice acționează ca o asigurare suplimentară, deoarece acumulează mai mult calcul într-un lanț din ce în ce mai lung. Prin convenție, orice bloc cu mai mult de șase confirmări este considerat irevocabil, pentru că ar necesita o cantitate imensă de calcul pentru a invalida și recalcula șase blocuri. Vom examina procesul de minerit și modul în care creează încredere mai în detaliu în [Minerit și Consens](#).

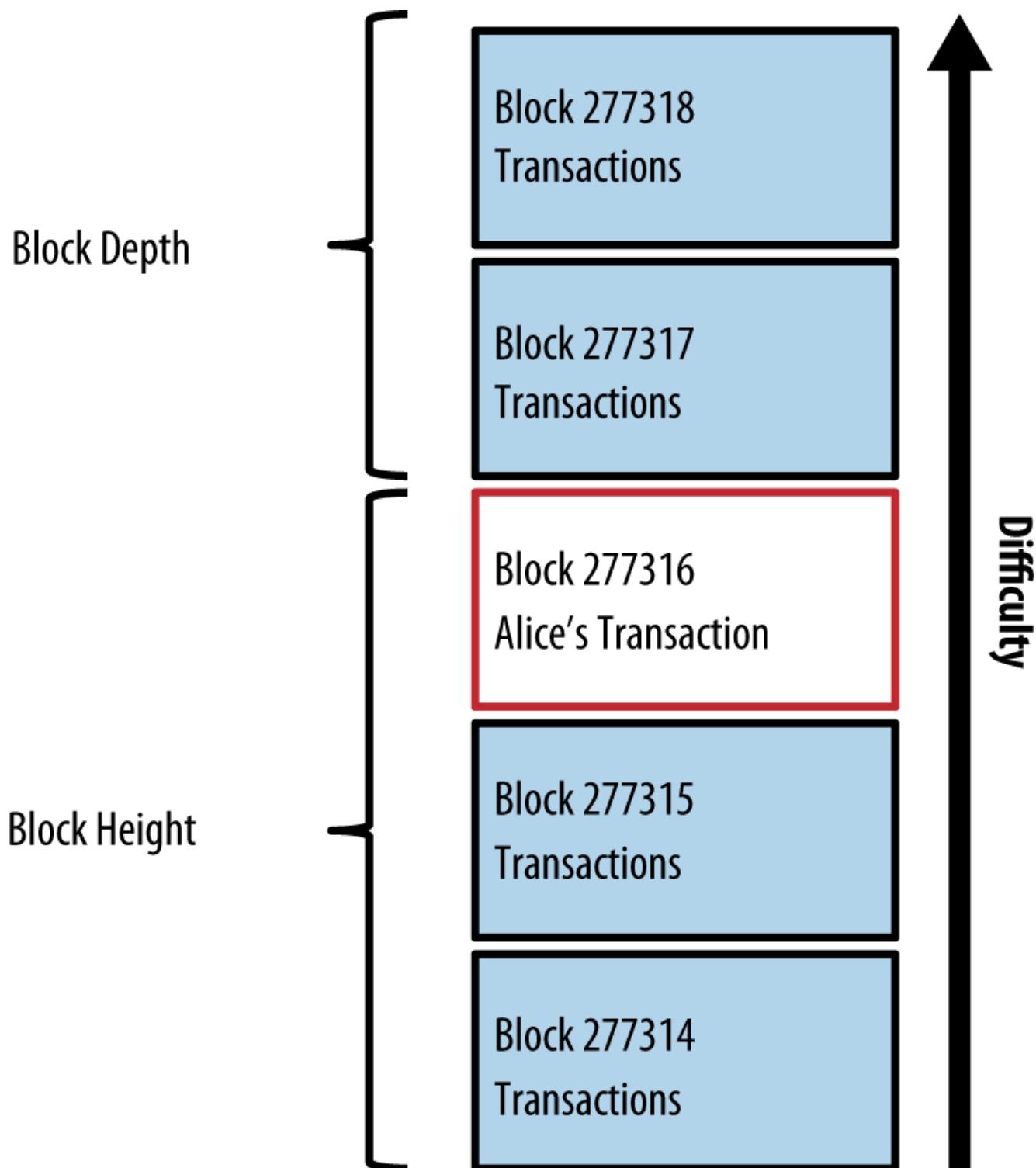


Figure 11. Tranzacția lui Alice inclusă în blocul #277316

## Cheltuirea Tranzacției

Acum că tranzacția lui Alice a fost încorporată ca parte a unui bloc în lanțul-de-blocuri, ea este parte din registrului distribuit al bitcoin și este vizibilă tuturor aplicațiilor bitcoin. Fiecare client

bitcoin poate verifica independent tranzacția ca fiind validă și cheltuibilă. Clienții nod-complet pot urmări sursa fondurilor de la momentul în care monedele bitcoin au fost generate pentru prima dată într-un bloc, treptat, de la tranzacție la tranzacție, până când ajung la adresa lui Bob. Clienții supli (lightweight) pot efectua ceea ce se numește o verificare simplificată a plășii (vezi [Noduri de Verificare Simplificată a Plășii \(SPV\)](#)) confirmând că tranzacția se află în lanșul-de-blocuri și are câteva blocuri minate după ea, oferind astfel sigurană că minerii au acceptat-o ca fiind validă.

Bob poate acum să cheltuiască ieșirea acesteia și a altor tranzacții. De exemplu, Bob poate să plătească un contractor sau un furnizor transferând valoare de la plata pe care Alice a făcut-o pentru ceașca de cafea, către acești noi proprietari. Cel mai probabil, software-ul bitcoin a lui Bob va adăuga multe plășii mici într-o plată mai mare, concentrând probabil toate încasările zilei într-o singură tranzacție. Aceasta ar adăuga diferite plășii într-o singură ieșire (și o singură adresă). Pentru diagrama unei tranzacții de agregare vezi [Tranzacție care agregă fonduri](#).

Pe măsură ce Bob cheltuiște plășile primite de la Alice și de la alți clienți, el extinde lanșul de tranzacții. Să presupunem că Bob îl plătește pe designer-ul său web Gopesh din Bangalore pentru o nouă pagină web. Acum lanșul de tranzacții va arăta ca în [Tranzacția lui Alice ca parte a unui lanș de tranzacții de la Joe la Gopesh](#).

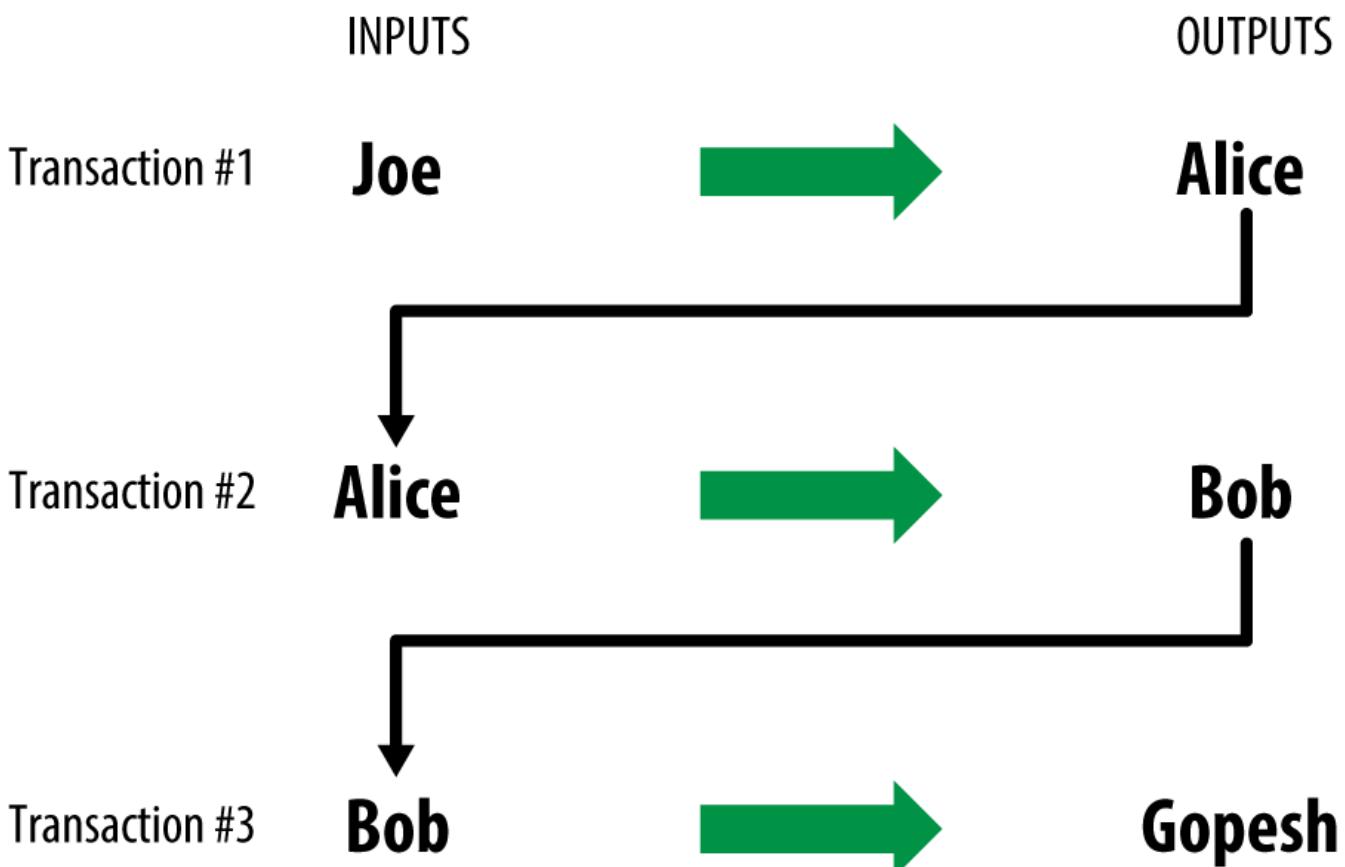


Figure 12. Tranzacția lui Alice ca parte a unui lanș de tranzacții de la Joe la Gopesh

În acest capitol, am văzut cum tranzacțiile formează un lanș care mută valoare de la un proprietar la altul. Am urmărit de asemenea tranzacția lui Alice, din momentul când a fost creată în portofelul ei, prin reșeaua bitcoin, până la minerii care au înregistrat-o în lanșul-de-blocuri. În restul acestei cărți vom examina tehnologiile specifice din spatele portofelelor, adreselor, semnăturilor, tranzacțiilor, reșelei, și în final mineritul.

# Bitcoin Core: Implementarea Referință

Bitcoin core este un proiect *open source*, iar codul sursă este disponibil sub o licență *open source* MIT. El poate fi descărcat gratis și folosit pentru orice scop. Conceptul "open source" înseamnă mai mult decât simpla gratuitate a proiectului. Înseamnă de asemenea că bitcoin este dezvoltat de o comunitate deschisă de voluntari. La început acea comunitate era formată doar din Satoshi Nakamoto. Până în 2016, codul sursă bitcoin a avut mai mult de 400 de contribuitori cu aproximativ o duzină de programatori lucrând aproape full-time, iar alte câteva desigurări lucrând part-time. Aproape oricine poate contribui la cod—înclusiv dumneavoastră!

Când bitcoin a fost creat de Satoshi Nakamoto, software-ul a fost de fapt finalizat înainte ca referatul reprodus în [Referatul Bitcoin de Satoshi Nakamoto](#) să fi fost scris. Satoshi a vrut să se asigure că funcționează înainte de a scrie despre asta. Acea primă implementare, atunci cunoscută simplu ca "Bitcoin" sau "clientul Satoshi," a fost puternic modificată și îmbunătățită. A evoluat în ceea ce este cunoscut ca *Bitcoin Core*, pentru a o diferenția de alte implementări compatibile. Bitcoin Core este *implementarea referință* a sistemului bitcoin, însemnând că este referință autoritativă a modului în care fiecare parte a tehnologiei ar trebui implementată. Bitcoin Core implementează toate aspectele bitcoin, inclusiv portofele, un motor de validare a tranzacțiilor și a blocurilor, și un nod-complet în rețeaua de-la-egal-la-egal bitcoin.

## WARNING

Chiar dacă Bitcoin Core include o implementare referință a unui portofel, acesta nu este intenționat să fie folosit ca portofel de producție pentru utilizatori sau pentru aplicații. Dezvoltatorii de aplicații sunt sfătuți să construiesc portofele folosind standarde moderne cum ar fi BIP-39 și BIP-32 (vezi [Cuvinte Cod Mnemonic \(BIP-39\)](#) și [Portofele HD \(BIP-32 / BIP-44\)](#)). BIP vine de la *Bitcoin Improvement Proposal* (Propunere de Îmbunătățire Bitcoin).

[Arhitectura Bitcoin Core \(Sursa: Eric Lombrozo\)](#) arată arhitectura Bitcoin Core.

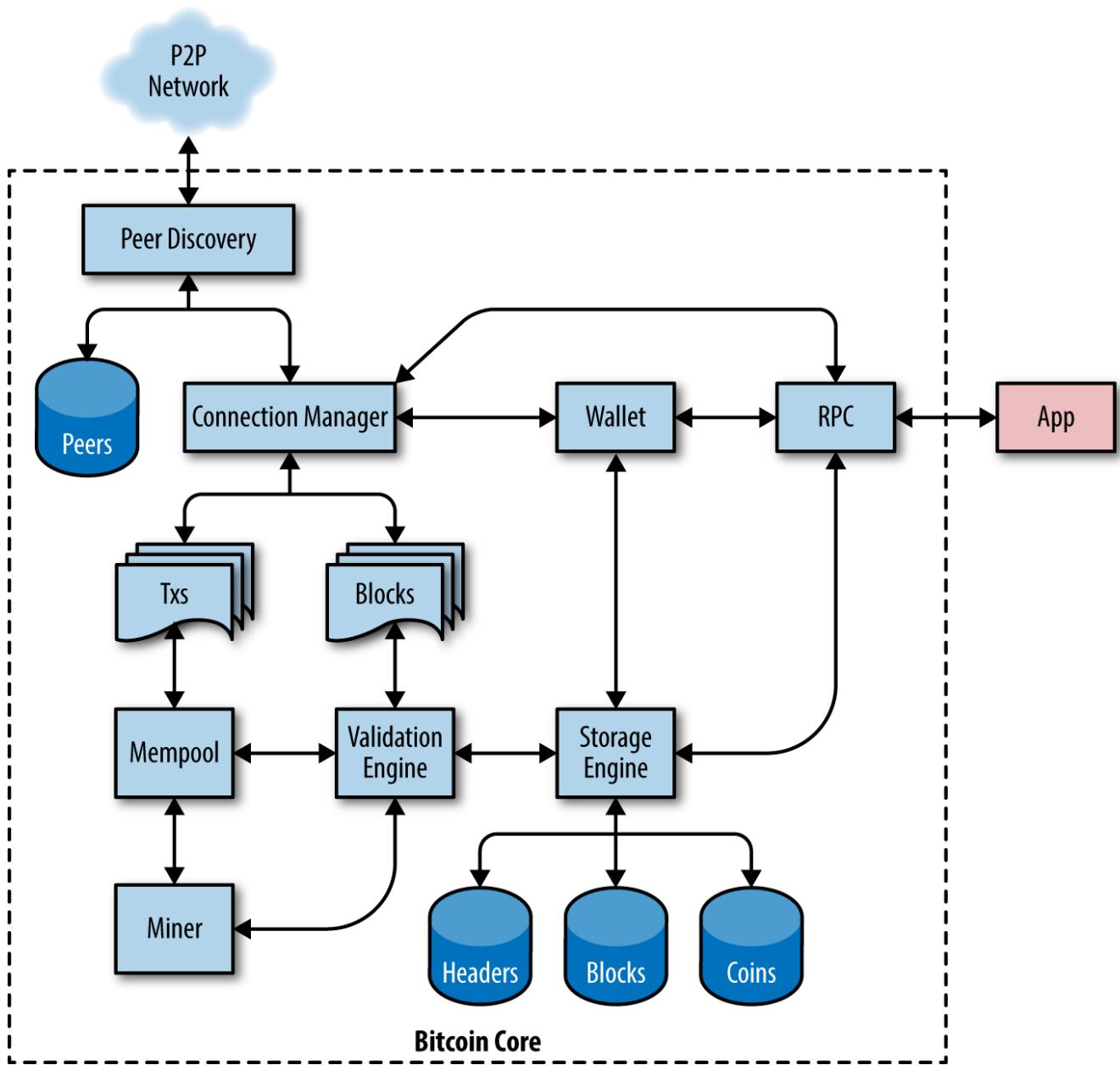


Figure 13. Arhitectura Bitcoin Core (Sursa: Eric Lombrozo)

## Mediul de Dezvoltare Bitcoin

Dacă sunteți un programator, o să doriți să vă configurați un mediu de dezvoltare cu toate instrumentele, bibliotecile, și software-ul de suport pentru scrierea aplicațiilor bitcoin. În acest capitol foarte tehnic, vom parcurge acest proces pas cu pas. Dacă materia devine prea încărcată (și de fapt nu doriți să creați un mediu de dezvoltare) nu ezitați să treceți la capitolul următor, care e mai puțin tehnic.

## Compilarea Bitcoin Core din Codul Sursă

Codul sursă Bitcoin Core poate fi descărcat ca arhivă sau clonând repozitoriuul GitHub. La [pagina de descărcare Bitcoin Core](#), selectați cea mai recentă versiune și descărcați arhiva comprimată a codului sursă, de exemplu, *bitcoin-0.15.0.2.tar.gz*. Alternativ, puteți folosi git din linia de comandă pentru a crea o copie locală a codului sursă de la [pagina GitHub a bitcoin](#).

**TIP**

În multe dintre exemplele din acest capitol vom folosi linia de comandă a sistemului de operare (cunoscută ca "shell"), accesată printr-o aplicație terminal. Acest shell va afișa un prompt; dumneavoastră introduceți o comandă; iar shell-ul răspunde cu niște text și cu un nou prompt pentru următoarea comandă. Prompt-ul s-ar putea să arate diferit pe calculatorul dumneavoastră, dar în exemplele următoare este notat cu simbolul \$. În exemple, când veți text după simbolul \$, nu introduceți simbolul \$, ci doar comanda care urmează imediat după el, apoi apăsați Enter pentru a executa comanda. În exemple, liniile de sub fiecare comandă sunt răspunsul sistemului de operare la acea comandă. Când veți următorul prefix \$, veți ști că este o comandă și ar trebui să repetați procesul.

În acest exemplu, folosim comanda *git* pentru a crea o copie locală ("clona") a codului sursă:

```
$ git clone https://github.com/bitcoin/bitcoin.git
Cloning into 'bitcoin'...
remote: Counting objects: 102071, done.
remote: Compressing objects: 100% (10/10), done.
Receiving objects: 100% (102071/102071), 86.38 MiB | 730.00 KiB/s, done.
remote: Total 102071 (delta 4), reused 5 (delta 1), pack-reused 102060
Resolving deltas: 100% (76168/76168), done.
Checking connectivity... done.
$
```

**TIP**

Git este cel mai folosit sistem distribuit de control al versiunilor, o parte esențială a setului de instrumente al oricărui programator. Este posibil să fie nevoie să instalați comanda *git*, sau o interfață grafică pentru git pe sistemul dumneavoastră de operare dacă nu o aveți deja.

Când operația git de clonare a fost finalizată, veți avea o copie locală completă a codului sursă în directorul *bitcoin*. Accesați acest director introducând **cd bitcoin** la prompt:

```
$ cd bitcoin
```

## Selectarea unei Versiuni Bitcoin Core

Implicit, copia locală va fi sincronizată cu cea mai recentă versiune a codului, care ar putea fi o versiune instabilă sau beta. Înainte de a compila codul, selectați o versiune specifică alegând o *etichetă*. Aceasta va sincroniza copia locală cu un instantaneu specific al codului identificat prin etichetă. Etichetele sunt folosite de programatori pentru a marca versiuni specifice ale codului cu un număr de versiune. În primul rând, pentru a găsi etichetele disponibile vom folosi comanda *git tag*:

```
$ git tag
v0.1.5
v0.1.6test1
v0.10.0
...
v0.11.2
v0.11.2rc1
v0.12.0rc1
v0.12.0rc2
...
```

Lista de etichete arată toate versiunile bitcoin lansate. Prin convenție, *candidații la lansare* (release candidates), care sunt destinați testării, au sufixul "rc". Versiunile stabile care pot fi rulate în sisteme de producție nu au niciun sufix. Din lista precedentă, selectați versiunea cea mai mare, care la momentul scrierii era v0.15.0. Pentru a sincroniza codul local cu această versiune, folosiți comanda *git checkout*:

```
$ git checkout v0.15.0
HEAD is now at 3751912... Merge #11295: doc: Old fee_estimates.dat are discarded by
0.15.0
```

Puteți verifica că aveți versiunea dorită executând comanda *git status*:

```
$ git status
HEAD detached at v0.15.0
nothing to commit, working directory clean
```

## Configurarea Build-ului Bitcoin Core

Codul sursă include documentația, care poate fi găsită în câteva fișiere. Examinați documentația localizată în *README.md* în directorul *bitcoin*, tastând la prompt **more README.md** și folosind bara de spațiu pentru a progresă la pagina următoare. În acest capitol, vom crea build-ul pentru clientul bitcoin Linux pentru linia de comandă, cunoscut ca *bitcoind*. Examinați instrucțiunile pentru compilarea clientului bitcoin pentru linia de comandă tastând **more doc/build-unix.md**. Instrucțiuni pentru macOS și Windows pot fi găsite în directorul *doc*, cu numele de *build-osx.md* sau respectiv *build-windows.md*.

Examinați cu atenție precondițiile pentru build, care sunt localizate în prima parte a documentației. Acestea sunt biblioteci ce trebuie să fie prezente pe sistemul dumneavoastră înainte să puteți începe compilarea bitcoin. Dacă aceste biblioteci lipsesc, procesul de build va eşua cu o eroare. Dacă aceasta se întâmplă pentru că ați omis o bibliotecă, o puteți instala, iar apoi să reluați procesul din punctul în care a rămas. Presupunând că dependințele sunt instalate, puteți porni procesul de build prin generarea unor scripturi folosind scriptul *autogen.sh*.

```
$ ./autogen.sh
...
glibtoolize: copying file 'build-aux/m4/libtool.m4'
glibtoolize: copying file 'build-aux/m4/ltoptions.m4'
glibtoolize: copying file 'build-aux/m4/ltsugar.m4'
glibtoolize: copying file 'build-aux/m4/ltversion.m4'
...
configure.ac:10: installing 'build-aux/compile'
configure.ac:5: installing 'build-aux/config.guess'
configure.ac:5: installing 'build-aux/config.sub'
configure.ac:9: installing 'build-aux/install-sh'
configure.ac:9: installing 'build-aux/missing'
Makefile.am: installing 'build-aux/depcomp'
...
```

Scriptul *autogen.sh* creează un set de scripturi automate de configurare care vor interoga sistemul dumneavoastră pentru a descoperi setările corecte și a se asigura că aveți toate bibliotecile necesare pentru a compila codul. Cel mai important dintre toate este scriptul *configure* care oferă o serie de opțiuni pentru a personaliza procesul de build. Tastați **./configure --help** pentru a vedea diversele opțiuni:

```
$ ./configure --help
'configure' configures Bitcoin Core 0.15.0 to adapt to many kinds of systems.

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

...
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]

--enable-wallet         enable wallet (default is yes)

--with-gui[=no|qt4|qt5|auto]
...
```

Scriptul *configure* vă permite să activați sau să dezactivați anumite funcționalități ale *bitcoind* cu ajutorul indicatoarelor (flags) **--enable-FEATURE** și **--disable-FEATURE**, unde **FEATURE** este înlocuit de numele funcționalității, după cum este descris în documentul de ajutor. În acest capitol vom construi clientul *bitcoind* cu toate funcționalitățile implicate. Nu vom folosi indicatorii de configurare, dar ar trebui să îi examinăm pentru a înțelege ce funcționalități opționale fac parte din client. Dacă sunteți un cadru didactic, restricțiile de pe calculatoarele din laborator s-ar putea să vă solicite să instalați aplicația în directorul utilizatorului (home directory, de exemplu, folosind **--prefix=\$HOME**).

Acstea sunt unele opțiuni utile care înlocuiesc comportamentul implicit al scriptului de configurare.

**--prefix=\$HOME** Aceasta înlocuiește locația implicită de instalare (care este */usr/local/*) pentru executabilul rezultat. Folosiți *\$HOME* pentru a pune totul în directorul userului (home directory), sau la o locație diferită.

**--disable-wallet** Aceasta este folosită pentru a dezactiva implementarea portofelului referință.

**--with-incompatible-bdb** Dacă faceți build la un portofel, permiteți folosirea unei versiuni incompatibile a bibliotecii Berkeley DB.

**--with-gui=no** Nu va face build la interfața grafică cu utilizatorul, care necesită biblioteca QT. Aceasta face build doar la server și la linia de comandă bitcoin.

Apoi, execuți scriptul *configure* pentru a descoperi automat toate bibliotecile și a crea un script de build personalizat pentru sistemul dumneavoastră:

```
$ ./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... gawk
checking whether make sets $(MAKE)... yes
...
[many pages of configuration tests follow]
...
$
```

Dacă totul a decurs bine, execuția comenții *configure* va crea scripturile de build personalizate care ne vor permite să compilăm *bitcoind*. Dacă sunt biblioteci care lipsesc sau dacă există erori, comanda *configure* se va termina cu o eroare în loc să creeze scripturile de build. Dacă apare o eroare, cel mai probabil se datorează unei biblioteci care lipsește sau care este incompatibilă. Examinați documentația de build din nou și asigurați-vă că instalați bibliotecile lipsă. Apoi rulați *configure* din nou și vedeți dacă asta rezolvă eroarea.

## Build-uirea Executabilelor Bitcoin Core

În continuare, veți compila codul sursă, un proces care poate dura până la o oră, depinzând de viteza CPU-ului și a memoriei disponibile. În timpul procesului de compilare ar trebui să vedeți ieșiri la consolă la fiecare câteva secunde sau minute, sau o eroare dacă ceva nu merge bine. Dacă apare o eroare, sau dacă procesul de compilare este întrerupt, poate fi reluat oricând tastând *make* din nou. Tastați **make** pentru a începe compilarea aplicației executabile:

```
$ make
Making all in src
  CXX  crypto/libbitcoinconsensus_la-hmac_sha512.lo
  CXX  crypto/libbitcoinconsensus_la-ripemd160.lo
  CXX  crypto/libbitcoinconsensus_la-sha1.lo
  CXX  crypto/libbitcoinconsensus_la-sha256.lo
  CXX  crypto/libbitcoinconsensus_la-sha512.lo
  CXX  libbitcoinconsensus_la-hash.lo
  CXX  primitives/libbitcoinconsensus_la-transaction.lo
  CXX  libbitcoinconsensus_la-pubkey.lo
  CXX  script/libbitcoinconsensus_la-bitcoinconsensus.lo
  CXX  script/libbitcoinconsensus_la-interpreter.lo
```

[... many more compilation messages follow ...]

```
$
```

Pe un sistem rapid cu mai mult de un nucleu CPU, s-ar putea să doriți să configurați numărul de procese paralele de compilare. De exemplu, *make -j 2* va folosi două nuclee dacă sunt disponibile. Dacă totul merge bine, Bitcoin Core este acum compilat. Ar trebui să rulați suita de teste unitare (unit test suite) cu *make check* pentru a vă asigura că bibliotecile folosite nu sunt deteriorate. Ultimul pas este instalarea diferitor executabile pe sistemul dumneavoastră folosind comanda *make install*. Vă s-ar putea solicita parola de utilizator, pentru că acest pas necesită drepturi de administrator.

```
$ make check && sudo make install
Password:
Making install in src
  ./build-aux/install-sh -c -d '/usr/local/lib'
libtool: install: /usr/bin/install -c bitcoind /usr/local/bin/bitcoind
libtool: install: /usr/bin/install -c bitcoin-cli /usr/local/bin/bitcoin-cli
libtool: install: /usr/bin/install -c bitcoin-tx /usr/local/bin/bitcoin-tx
...
$
```

Instalarea implicită a *bitcoind* îl plasează în */usr/local/bin*. Puteți confirma că Bitcoin Core este corect instalat interogând sistemul pentru calea executabilelor, după cum urmează:

```
$ which bitcoind
/usr/local/bin/bitcoind

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

# Rularea unui Nod Bitcoin Core

Rețeaua de la egal la egal a bitcoin este compusă din "noduri" de rețea, operate în mare parte de voluntari și de unele afaceri care construiesc aplicații bitcoin. Cei care rulează noduri bitcoin au o vedere directă și autoritară asupra lanțului-de-blocuri bitcoin, având o copie locală a tuturor tranzacțiilor, validate independent de către sistemele lor. Rulând un nod, nu trebuie să vă bazați pe nici un terț pentru a valida o tranzacție. Mai mult decât atât, rulând un nod bitcoin contribuiți la rețeaua bitcoin făcând-o mai robustă.

Rularea unui nod necesită, însă, un sistem conectat permanent, cu suficient resurse pentru a procesa toate tranzacțiile bitcoin. Dacă alegeți să indexați toate tranzacțiile și să mențineți o copie locală completă a lanțului-de-blocuri, este posibil să aveți nevoie de mult spațiu pe disc și de multă memorie RAM. De la începutul anului 2018, un nod index-complet necesită 2 GB de RAM și minimum 160 GB de spațiu pe disc (vezi <https://blockchain.info/charts/blocks-size>). Bitcoin necesită de asemenea să transmită și să primească tranzacții și blocuri bitcoin, consumând lățime de bandă. Dacă conexiunea dumneavoastră la internet este limitată, are o capacitate de date scăzută, sau este contorizată (taxată per gigabit), probabil nu ar trebui să rulați un nod bitcoin pe ea, sau să rulați într-un mod care îi restrâne lungimea de bandă (vezi [Exemplu de configurare a unui sistem cu resurse limitate](#)).

Bitcoin Core menține implicit o copie completă a lanțului-de-blocuri, cu fiecare tranzacție care a avut loc vreodată în rețeaua bitcoin de la înființare, în 2009. Acest set de date are mărimea de zeci de gigabytes și este descărcat incremental pe parcursul câtorva zile sau săptămâni, în funcție de viteza procesorului și a conexiunii la internet. Bitcoin Core nu va putea procesa tranzacții sau actualiza soldurile contului până când întreg setul de date al lanțului-de-blocuri nu este descărcat.

**TIP** Asigurați-vă că aveți destul spațiu pe disc, lățime de bandă, și timp pentru a finaliza sincronizarea inițială. Puteți configura Bitcoin Core să reducă marimea lanțului-de-blocuri eliminând blocurile vechi (vezi [Exemplu de configurare a unui sistem cu resurse limitate](#)), dar tot va descărca întreg setul de date înainte de a elimina blocurile vechi.

În ciuda acestor cerințe de resurse, mii de voluntari rulează noduri bitcoin. Unii rulează pe sisteme simple, cum ar fi un Raspberry Pi (un calculator de dimensiunea unui pachet de cărți care costă 35\$). Mulți voluntari rulează de asemenea noduri bitcoin pe servere închiriate, de obicei o variantă de Linux. O instanță de *Server Virtual Privat* (Virtual Private Server - VPS) sau de *Server în Cloud* (Cloud Computing Server) poate fi folosită pentru a rula un nod bitcoin. Asemenea servere pot fi închiriate pentru sume între 25\$ și 50\$ pe lună de la o varietate de furnizori.

De ce ați vrea să rulați un nod? Iată câteva dintre cele mai întâlnite motive:

- Dacă dezvoltați software bitcoin și trebuie să folosiți un nod bitcoin pentru access programabil (API) la rețea și la lanțul-de-blocuri.
- Dacă dezvoltați aplicații care trebuie să valideze tranzacții conform regulilor de consens ale bitcoin. De obicei, companiile de software bitcoin rulează câteva noduri.
- Dacă doriți să sprijiniți bitcoin. Rularea unui nod face rețeaua mai robustă și îi permite să servească mai multe portofele, mai mulți utilizatori, și mai multe tranzacții.

- Dacă nu vreți să vă bazați pe un terț pentru a procesa sau valida tranzacțiile dumneavoastră.

Dacă citiți această carte și sunteți interesați să dezvoltați software bitcoin, ar trebui să rulați propriul nod.

## Configurarea Nodului Bitcoin Core

Bitcoin Core va căuta un fișier de configurare în directorul său de date de fiecare dată când pornește. În această secțiune vom examina diferite opțiuni de configurare și vom pregăti un fișier de configurare. Pentru a localiza fișierul de configurare, rulați *bitcoind -printtoconsole* în terminal și inspectați primele linii.

```
$ bitcoind -printtoconsole
Bitcoin version v0.15.0
Using the 'standard' SHA256 implementation
Using data directory /home/ubuntu/.bitcoin/
Using config file /home/ubuntu/.bitcoin/bitcoin.conf
...
[a lot more debug output]
...
```

Puteți apăsa Ctrl+C pentru a opri nodul după ce ați determinat locația fișierului de configurări. De obicei fișierul de configurări este în directorul utilizatorului (home directory) în *.bitcoin*, în directorul de date. Nu este creat automat, dar puteți crea un fișier de configurare de pornire copiind din exemplul [Exemplu de configurare a unui nod indexat complet](#), de mai jos. Puteți crea sau modifica fișierul de configurări în editorul dumneavoastră preferat.

Bitcoin Core oferă mai mult de 100 de opțiuni de configurare care modifică comportamentul nodului de rețea, stocarea lanțului-de-blocuri, și multe alte aspecte ale operării sale. Pentru a vedea o listă a acestor opțiuni, rulați *bitcoind -help*:

```

$ bitcoind --help
Bitcoin Core Daemon version v0.15.0

Usage:
  bitcoind [options]                                     Start Bitcoin Core Daemon

Options:
  -?                                         Print this help message and exit
  -version                                    Print version and exit
  -alertnotify=<cmd>
    Execute command when a relevant alert is received or we see a really
    long fork (%s in cmd is replaced by message)
  ...
  [many more options]
  ...
  -rpcthreads=<n>
    Set the number of threads to service RPC calls (default: 4)

```

Iată câteva dintre cele mai importante opțiuni pe care le puteți seta în fișierul de configurări, sau ca parametri de la linia de comandă pentru *bitcoind*:

### **alertnotify**

Rulează o comandă sau un script specificat pentru a trimite alerte de urgență proprietarului acestui nod, de obicei prin email.

### **conf**

O locație alternativă pentru fișierul de configurări. Această configurare are sens doar ca un parametru de la linia de comandă pentru *bitcoind*, întrucât nu poate fi în interiorul fișierului de configurare la care face referire.

### **datadir**

Selectează directorul și sistemul de fișiere în care să pună toate datele lanțului-de-blocuri. În mod implicit acesta este subdirectorul *.bitcoin* în interiorul directorului utilizatorului (home directory). Asigurați-vă că partitia are disponibili câțiva GB.

### **prune**

Reduce cerințele de spațiu pe disc la acest număr de megabytes, prin ștergerea blocurilor vechi. Folosiți această configurare pe un nod cu resurse limitate care nu poate stoca tot lanțul-de-blocuri.

### **txindex**

Menține un index al tuturor tranzacțiilor. Acest lucru înseamnă o copie completă a lanțului-de-

blocuri care vă permite să găsiți programatic după ID orice tranzacție.

### dbcache

Dimensiunea cache-ului UTXO. Valoarea implicită este de 300 MiB. Măriți această limită pe hardware-uri performante și reduceți-o pe cele mai puțin performante pentru a economisi memorie în detrimentul operațiilor încete de IO.

### maxconnections

Setează numărul maxim de noduri de la care să accepte conexiuni. Reducerea acestei valori față de valoarea implicită va reduce lățimea de bandă consumată. Folosiți opțiunea aceasta dacă aveți o limită la transferul de date sau dacă plătiți per gigabyte.

### maxmempool

Limităza pool-ul de memorie pentru tranzacții la numărul specificat de megabytes. Folosiți opțiunea pentru a reduce consumul de memorie pe nodurile cu memorie limitată.

### maxreceivebuffer/maxsendbuffer

Limitați bufferul de memorie per conexiune la acest multiplu de 1000 de bytes. Folosiți pe nodurile cu memorie limitată.

### minrelaytxfee

Setați rata minimă de comision pentru tranzacțiile pe care le veți transmite. Sub această valoare, tranzacția este tratată în mod diferit, este respinsă din bazinul de tranzacții și nu este transmisă mai departe.

## Indexul Bazei de Date pentru Tranzacții și Opțiunea txindex

În mod implicit Bitcoin Core construiește o bază de date care conține *doar* tranzacțiile legate de portofelul utilizatorului. Dacă doriți să puteți accesa *orice* tranzacție folosind comenzi precum *getrawtransaction* (vezi [Explorarea și Decodarea Tranzacțiilor](#)), atunci trebuie să configurați Bitcoin Core să construiască un index complet al tranzacțiilor, lucru care poate fi obținut folosind opțiunea *txindex*. Setați *txindex=1* în fișierul de configurații al Bitcoin Core. Dacă nu setați opțiunea aceasta la început și o setați mai târziu, atunci va trebui să restartați *bitcoind* cu opțiunea *-reindex* și să așteptați să reconstruiască indexul.

[Exemplu de configurare a unui nod indexat complet](#) arată cum puteți combina opțiunile precedente, pentru un nod complet indexat, care rulează ca API de backend pentru o aplicație bitcoin.

*Example 3. Exemplu de configurare a unui nod indexat complet*

```
alertnotify=myemailscript.sh "Alert: %s"
datadir=/lotsofspace/bitcoin
txindex=1
```

[Exemplu de configurare a unui sistem cu resurse limitate](#) arată un nod cu resurse limitate rulând

pe un server mai mic.

*Example 4. Exemplu de configurare a unui sistem cu resurse limitate*

```
alertnotify=myemailscript.sh "Alert: %s"
maxconnections=15
prune=5000
dbcache=150
maxmempool=150
maxreceivebuffer=2500
maxsendbuffer=500
```

După ce ați editat fișerul de configurări și ați setat opțiunile care reprezintă cel mai bine necesitățile dumneavoastră, puteți testa *bitcoind* cu această configurare. Rulați Bitcoin Core cu opțiunea *printtconsole* pentru a rula în prim-plan (foreground) cu scriere la consolă:

```
$ bitcoind -printtoconsole

Bitcoin version v0.15.0
InitParameterInteraction: parameter interaction: -whitelistforcerelay=1 -> setting
-whitelistrelay=1
Assuming ancestors of block
00000000000000000000003b9ce759c2a087d52abc4266f8f4ebd6d768b89defa50a have valid
signatures.
Using the 'standard' SHA256 implementation
Default data directory /home/ubuntu/.bitcoin
Using data directory /lotsofspace/.bitcoin
Using config file /home/ubuntu/.bitcoin/bitcoin.conf
Using at most 125 automatic connections (1048576 file descriptors available)
Using 16 MiB out of 32/2 requested for signature cache, able to store 524288 elements
Using 16 MiB out of 32/2 requested for script execution cache, able to store 524288
elements
Using 2 threads for script verification
HTTP: creating work queue of depth 16
No rpcpassword set - using random cookie authentication
Generated RPC authentication cookie /lotsofspace/.bitcoin/.cookie
HTTP: starting 4 worker threads
init message: Verifying wallet(s)...
Using BerkeleyDB version Berkeley DB 4.8.30: (April 9, 2010)
Using wallet wallet.dat
CDBEnv::Open: LogDir=/lotsofspace/.bitcoin/database
ErrorFile=/lotsofspace/.bitcoin/db.log
scheduler thread start
Cache configuration:
* Using 250.0MiB for block index database
* Using 8.0MiB for chain state database
* Using 1742.0MiB for in-memory UTXO set (plus up to 286.1MiB of unused mempool space)
init message: Loading block index...
Opening LevelDB in /lotsofspace/.bitcoin/blocks/index
Opened LevelDB successfully

[... more startup messages ...]
```

Puteți apăsa Ctrl+C pentru a întrerupe procesul odată ce sunteți convins că încarcă setările corecte și că rulează aşa cum vă așteptați.

Pentru a rula Bitcoin Core ca proces în fundal (background), porniți-l cu opțiunea *daemon*, ca *bitcoind -daemon*.

Pentru a monitoriza progresul și statusul de rulare a nodului bitcoin, folosiți comanda *bitcoin-cli getblockchaininfo*:

```
$ bitcoin-cli getblockchaininfo
```

Aceasta arată un nod cu o înalțime a lanțului-de-blocuri de 0 blocuri și 83999 de antete. Momentan, nodul aduce antetele blocurilor celui mai bun lanț, iar apoi va continua să descarce blocurile în întregime.

Odată ce sunteți mulțumit de configurarea pe care ați ales-o, ar trebui să adăugați bitcoin la scriptul de pornire al sistemului dumneavoastră de operare, astfel încât să ruleze continuu și să repornească atunci când sistemul de operare repornește. Veți găsi mai multe exemple de scripturi de pornire pentru diferite sisteme de operare în directorul codului sursă bitcoin în *contrib/init* și un fișier *README.md* care explică ce script folosește fiecare sistem.

# API-ul (Interfața de Programare a Aplicațiilor) Bitcoin Core

Clientul Bitcoin Core implementează o interfață JSON-RPC care poate fi accesată și din linia de comandă folosind *bitcoin-cli*. Linia de comandă ne permite să experimentăm interactiv cu funcționalități care sunt de asemenea disponibile programatic folosind API-ul. Pentru început, invocați comanda *help* pentru a vedea lista comenziilor bitcoin RPC (Remote Procedure Call) disponibile:

```
$ 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"
...
...
verifymessage "bitcoinaddress" "signature" "message"
walletlock
walletpassphrase "passphrase" timeout
walletpassphrasechange "oldpassphrase" "newpassphrase"
```

Fiecare din aceste comenzi poate primi o serie de parametri. Pentru a primi mai mult ajutor, o descriere detaliată, și informații despre parametri, adăugati numele comenzi după *help*. De

exemplu, pentru a vedea pagina de ajutor la comanda RPC *getblockhash*:

```
$ bitcoin-cli help getblockhash
getblockhash height
```

Returns hash of block in best-block-chain at height provided.

Arguments:

1. height (numeric, required) The height index

Result:

"hash" (string) The block hash

Examples:

```
> bitcoin-cli getblockhash 1000
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getblockhash", "params": [1000 ]}' -H 'content-type: text/plain;' http://127.0.0.1:8332/
```

La sfârșitul informațiilor de ajutor veți vedea două exemple ale comenzi RPC, unul folosind *bitcoin-cli*, iar celălalt folosind clientul HTTP *curl*.

```
$ bitcoin-cli getblockhash 1000
0000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09
```

Rezultatul este un rezumat de bloc (block hash), care este descris mai în detaliu în capitolele următoare. Însă deocamdată această comandă ar trebui să returneze același rezultat și pe sistemul dumneavoastră, demonstrând astfel că nodul Bitcoin Core rulează, acceptă comenzi, și are informații despre blocul 1000.

În secțiunile următoare vom demonstra câteva comenzi RPC foarte utile și rezultatele așteptate ale acestora.

## Obținerea Informațiilor despre Statusul Clientului Bitcoin Core

Bitcoin Core oferă rapoarte de stare privind diferite module prin interfața JSON-RPC. Cele mai importante comenzi includ *getblockchaininfo*, *getmempoolinfo*, *getnetworkinfo* și *getwalletinfo*.

Comanda RPC *getblockchaininfo* a fost prezentată mai devreme. Comanda *getnetworkinfo* afișează informații de bază despre statusul nodului de rețea bitcoin. Folosiți *bitcoin-cli* pentru a o rula:

```
$ bitcoin-cli getnetworkinfo
```

```

"version": 150000,
"subversion": "/Satoshi:0.15.0/",
"protocolversion": 70015,
"localservices": "000000000000000d",
"localrelay": true,
"timeoffset": 0,
"networkactive": true,
"connections": 8,
"networks": [
    ...
    detailed information about all networks (ipv4, ipv6 or onion)
    ...
],
"relayfee": 0.00001000,
"incrementalfee": 0.00001000,
"localaddresses": [
],
"warnings": ""
}

```

Datele sunt returnate în format JSON (JavaScript Object Notation), un format care poate fi ușor "consumat" de către toate limbajele de programare, dar care este în același timp destul de ușor de citit de către oameni. Printre aceste date vedem numerele de versiune pentru clientul software bitcoin (150000) și pentru protocolul bitcoin (70015). Vedem numărul curent de conexiuni (8) și diferite informații despre rețeaua bitcoin și despre setările legate de acest client.

**TIP** Va dura o vreme, probabil mai mult de o zi, pentru ca clientul *bitcoind* să "ajungă" la înălțimea curentă a lanțului-de-blocuri, pe măsură ce descarcă blocuri de la alții clienti bitcoin. Puteți verifica progresul folosind *getblockchaininfo* pentru a vedea numărul de blocuri cunoscute.

## Explorarea și Decodarea Tranzacțiilor

Comenzi: *getrawtransaction*, *decoderawtransaction*

În [Cumpărarea unei Cești de Cafea](#), Alice a cumpărat o ceașcă de cafea de la Cafeneaua lui Bob. Tranzacția ei a fost înregistrată în lanțul-de-blocuri cu ID-ul de tranzacție (*txid*) *0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2*. Vom folosi API-ul pentru a prelua și examina acea tranzacție folosind ID-ul de tranzacție ca parametru:

```
$ bitcoin-cli getrawtransaction 0627052b6f28912f2703066a912ea577f2ce4da4caa5a05fb0a57286c345c2f2
```

```
0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd734d2804fe65fa3577900000008b483045022100884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c40ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813014010484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc54123363767089d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adfffffffff0260e31600000000001976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac00000000
```

**TIP** ID-ul unei tranzacții nu este autoritar până când o tranzacție nu a fost confirmată. Absența unui rezumat (hash) al tranzacției în lanțul-de-blocuri nu înseamnă că tranzacția nu a fost procesată. Aceasta este cunoscută sub numele de "maleabilitatea tranzacțiilor," pentru că rezumatul (hash-ul) tranzacției poate fi modificat înainte de confirmarea într-un bloc. După confirmare, *txid*-ul este imutabil și autoritar.

Comanda *getrawtransaction* returnează o tranzacție serializată în notație hexazecimală. Pentru a o decoda, vom folosi comanda *decoderawtransaction*, oferind valoarea în hexa ca parametru. Puteți copia valoarea hexa returnată de *getrawtransaction* și să o oferiți ca parametru pentru *decoderawtransaction*:

```
$ bitcoin-cli decoderawtransaction 0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd734d2804fe65fa35779000000008b483045022100884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813014010484ecc0d46f1918b30928fa0e4ed99f16a0fb4fd0e0735e7ade8416ab9fe423cc54123363767089d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adfffffffff0260e31600000000001976a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef800000000001976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a88ac00000000
```

```
{
  "txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2",
  "size": 258,
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "7957a35fe64f80d234d76d83a2...8149a41d81de548f0a65a8a999f6f18",
      "vout": 0,
      "scriptSig": {
        "asm": "3045022100884d142d86652a3f47ba4746ec719bbfb040a570b1decc...",
        "hex": "483045022100884d142d86652a3f47ba4746ec719bbfb040a570b1de..."
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01500000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ab68...5f654e7 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"
        ]
      }
    },
    {
      "value": 0.08450000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 7f9b1a...025a8 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK"
        ]
      }
    }
  ]
}
```

Valoarea decodată a tranzacției ne arată toate componentele acestei tranzacții, inclusiv intrările și ieșirile tranzacției. În acest caz putem să vedem că tranzacția care a creditat noua noastră adresă cu 15 millibits a folosit o intrare și două ieșiri. Intrarea acestei tranzacții a fost ieșirea unei

tranzacții confirmate anterior (prezentă la vin *txid* și începând cu *7957a35fe*). Cele două ieșiri corespund cheltuirii celor 15 millibit și respectiv cu restul returnat expeditorului.

Putem explora în continuare lanțul-de-blocuri examinând tranzacția anterioară referențiată prin *txid* ei în tranzacția curentă folosind aceleași comenzi (e.g., *getrawtransaction*). Să rind de la tranzacție la tranzacție putem urmări un lanț de tranzacții deoarece monedele sunt transmise de la adresa unui proprietar la adresa altui proprietar.

## Explorarea Blocurilor

Comenzi: *getblock*, *getblockhash*

Explorarea blocurilor este similară cu explorarea tranzacțiilor. Totuși, la blocuri se poate face referire fie prin *înălțimea* blocului fie prin *rezumatul (hash-ul)* blocului. În primul rând, să găsim un bloc după *înălțimea* la care se află. În [Cumpărarea unei Cești de Cafea](#), am văzut că tranzacția lui Alice a fost inclusă în blocul 277316.

Folosim comanda *getblockhash*, care primește *înălțimea* blocului ca parametru și returnează rezumatul blocului respectiv:

```
$ bitcoin-cli getblockhash 277316
000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

Acum că știm în ce bloc a fost inclusă tranzacția lui Alice, putem interoga acel bloc. Folosim comanda *getblock* cu un rezumat (hash) de bloc ca parametru:

```
$ bitcoin-cli getblock 000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

Blocul conține 419 tranzacții, iar a 64-a tranzacție listată (0627052b...) este plata pentru cafea a lui Alice. Valoarea câmpului *height* ne spune că acesta este al 277316-lea bloc din lanțul-de-blocuri.

## Folosirea Interfeței Programatice a Bitcoin Core

Utilitarul *bitcoin-cli* este foarte folositor pentru a explora API-ul Bitcoin Core și pentru testarea funcțiilor. Dar întregul scop al unei interfețe de programare a aplicației este să acceseze funcțiile programatic. În această secțiune vom demonstra accesarea Bitcoin Core dintr-un alt program.

API-ul Bitcoin Core este o interfață JSON-RPC. JSON vine de la JavaScript Object Notation și este o modalitate foarte convenabilă de prezentare a datelor pe care o înțeleg cu ușurință și oamenii și programele. RPC vine de la Remote Procedure Call, ceea ce înseamnă că apelăm proceduri (funcții) care sunt la distanță (pe nodul Bitcoin Core) folosind un protocol de rețea. În acest caz, protocolul de rețea este HTTP, sau HTTPS (pentru conexiuni criptate).

Când am folosit comanda *bitcoin-cli* pentru a vedea pagina de ajutor pentru o comandă, ne-a arătat un exemplu de folosire a *curl*, versatilul client HTTP din linia de comandă, pentru a construi unul din aceste apeluri JSON-RPC:

```
$ curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest", "method": "getblockchaininfo", "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:8332/
```

Această comandă ne arată că *curl* trimite o cerere (request) HTTP la gazda locală (127.0.0.1), conectându-se la portul bitcoin implicit (8332), și trimițând o cerere (request) *jsonrpc* pentru metoda *getblockchaininfo* folosind codificarea *text/plain*.

S-ar putea să observați că *curl* vă va cere trimitera credențialelor împreună cu cererea (request-ul). Bitcoin Core va crea aleator o parolă la fiecare pornire și o va stoca în directorul de date sub numele *.cookie*. Utilitarul *bitcoin-cli* poate citi acest fișier de parolă dacă are acces la directorul de date. Similar, puteți copia parola și să o transmități cu *curl* (sau cu orice alt utilitar pentru API-ul Bitcoin Core RPC). În mod alternativ, puteți crea o parolă statică cu scriptul utilitar furnizat în */share/rpcauth/rpcauth.py* în directorul sursă al Bitcoin Core.

Dacă implementați un apel JSON-RPC în programul dumea voastră, puteți folosi o bibliotecă HTTP generică pentru a construi apelul, similar cu ce este prezentat în exemplul *curl* precedent.

Totuși, există biblioteci în aproape orice limbaj de programare care ”învelesc” API-ul Bitcoin Core într-un mod care face utilizarea mult mai simplă. Vom folosi biblioteca *python-bitcoinlib* pentru a simplifica accesul la API. Rețineți că acest lucru necesită să aveți un nod Bitcoin Core care rulează și care va fi folosit pentru a efectua apelurile JSON-RPC.

Scriptul Python de la [Rularea \*getblockchaininfo\* prin API-ul JSON-RPC al Bitcoin Core](#) efectuează un apel simplu la *getblockchaininfo* și afișează parametrul *blocks* din datele returnate de Bitcoin Core.

*Example 5. Rularea *getblockchaininfo* prin API-ul JSON-RPC al Bitcoin Core*

```
from bitcoin.rpc import RawProxy

# Create a connection to local Bitcoin Core node
p = RawProxy()

# Run the getblockchaininfo command, store the resulting data in info
info = p.getblockchaininfo()

# Retrieve the 'blocks' element from the info
print(info['blocks'])
```

Rularea ne dă următorul rezultat:

```
$ python rpc_example.py
394075
```

Ne spune că nodul nostru local Bitcoin Core are 394075 de blocuri în lanțul-de-blocuri. Nu este un rezultat spectaculos, dar demonstrează utilizarea de bază a bibliotecii ca o interfață simplificată la API-ul JSON-RPC al Bitcoin Core.

În continuare, vom folosi apelurile *getrawtransaction* și *decoderawtransaction* pentru a primi detaliiile plății pentru cafeaua lui Alice. În [Preluarea tranzacției și iterarea peste ieșiri](#), preluăm tranzacția lui Alice și enumerăm ieșirile tranzacției. Pentru fiecare ieșire, afișăm adresa destinatarului și valoarea. Să ne amintim că tranzacția lui Alice a avut o ieșire pentru plata la Cafeneaua lui Bob și o ieșire pentru restul primit de Alice.

*Example 6. Preluarea tranzacției și iterarea peste ieșiri*

```
from bitcoin.rpc import RawProxy

p = RawProxy()

# Alice's transaction ID
txid = "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2"

# First, retrieve the raw transaction in hex
raw_tx = p.getrawtransaction(txid)

# Decode the transaction hex into a JSON object
decoded_tx = p.decoderawtransaction(raw_tx)

# Retrieve each of the outputs from the transaction
for output in decoded_tx['vout']:
    print(output['scriptPubKey']['addresses'], output['value'])
```

Rulând acest cod, vom obține:

```
$ python rpc_transaction.py
([u'1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA'], Decimal('0.01500000'))
([u'1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK'], Decimal('0.08450000'))
```

Ambele exemple precedente sunt destul de simple. Nu aveți cu adevărat nevoie de un program pentru a le rula; la fel de simplu ați fi putut folosi utilitarul *bitcoin-cli*. Următorul exemplu, totuși, necesită câteva sute de apeluri RPC și demonstrează mai clar folosirea unei interfețe programatice.

În [Preluarea unui bloc și însumarea tuturor ieșirilor tranzacțiilor](#), am preluat mai întâi blocul 277316, apoi am preluat fiecare din cele 419 tranzacții conținute referențiind ID-ul fiecărei tranzacții. Apoi am iterat peste fiecare din ieșirile tranzacției și am însumat valorile.

*Example 7. Preluarea unui bloc și însumarea tuturor ieșirilor tranzacțiilor*

```
from bitcoin.rpc import RawProxy

p = RawProxy()

# The block height where Alice's transaction was recorded
blockheight = 277316

# Get the block hash of block with height 277316
blockhash = p.getblockhash(blockheight)

# Retrieve the block by its hash
block = p.getblock(blockhash)

# Element tx contains the list of all transaction IDs in the block
transactions = block['tx']

block_value = 0

# Iterate through each transaction ID in the block
for txid in transactions:
    tx_value = 0
    # Retrieve the raw transaction by ID
    raw_tx = p.getrawtransaction(txid)
    # Decode the transaction
    decoded_tx = p.decoderawtransaction(raw_tx)
    # Iterate through each output in the transaction
    for output in decoded_tx['vout']:
        # Add up the value of each output
        tx_value = tx_value + output['value']

    # Add the value of this transaction to the total
    block_value = block_value + tx_value

print("Total value in block: ", block_value)
```

Rulând acest cod, vom obține:

```
$ python rpc_block.py

('Total value in block: ', Decimal('10322.07722534'))
```

Codul nostru calculează valoarea totală tranzacționată în acest bloc ca fiind de 10322,07722534 BTC (inclusiv recompensa de 25 BTC și 0,0909 BTC în comisioane). Comparați valoarea cu una raportată de un site de explorare a blocurilor căutând după rezumatul blocului sau după înalțimea lui. Unele exploratoare de blocuri raportează valoarea totală excludând recompensa și

comisioanele. Vedeți dacă puteți observa diferența.

## Clienti, Biblioteci și Instrumente Alternative

Există mulți clienti, biblioteci, instrumente, și chiar implementări nod-complet alternative în ecosistemul bitcoin. Acestea sunt implementate într-o multitudine de limbaje de programare, oferind programatorilor interfețe native în limbajul lor preferat.

Următoarele secțiuni enumeră câteva din cele mai bune biblioteci, clienti, și instrumente, organizate în funcție de limbajul de programare.

### C/C++

#### **Bitcoin Core**

Implementarea referință a bitcoin

#### **libbitcoin**

Toolkit, nod, și biblioteca de conses pentru C++

#### **bitcoin explorer**

Utilitarul din linia de comandă al Libbitcoin

#### **picocoин**

O bibliotecă client suplă (lightweight) pentru bitcoin scrisă în C de către Jeff Garzik

### JavaScript

#### **bcoin**

O implementare modulară și scalabilă a unui nod-complet cu API

#### **Bitcore**

Nod complet, API, și bibliotecă de către Bitpay

#### **BitcoinJS**

O bibliotecă pură JavaScript pentru node.js și pentru browsere

### Java

#### **bitcoinj**

O bibliotecă client nod-complet în Java

#### **Bits of Proof (BOP)**

O implementare profesionistă a bitcoin în Java

### PHP

#### **bitwasp/bitcoin**

O bibliotecă bitcoin în PHP, și proiectele adiționale

## Python

### [python-bitcoinlib](#)

O bibliotecă bitcoin, bibliotecă de consens, și nod scrise în Python de Peter Todd

### [pycoin](#)

O bibliotecă bitcoin scrisă în Python de Richard Kiss

### [pybitcointools](#)

O bibliotecă bitcoin scrisă în Python de Vitalik Buterin

## Ruby

### [bitcoin-client](#)

O bibliotecă "wrapper" pentru API-ul JSON-RPC scrisă în Ruby

## Go

### [btcd](#)

Un client bitcoin nod-complet scris în Go

## Rust

### [rust-bitcoin](#)

Biblioteca bitcoin Rust pentru serializare, analizare și apeluri API

## C#

### [NBitcoin](#)

Bibliotecă bitcoin cuprinzătoare pentru platforma .NET

## Objective-C

### [CoreBitcoin](#)

Toolkit bitcoin pentru ObjC și Swift

Și mai multe biblioteci există într-o multitudine de alte limbaje de programare, iar multe sunt create tot timpul.

## Chei, Adrese

S-ar putea să fi auzit că bitcoin se bazează pe *criptografie*, care este o ramură a matematicii folosită extensiv în securitatea sistemelor informatici. Criptografie înseamnă "scriere secretă" în greacă, dar domeniul criptografiei conține mult mai multe pe lângă o simplă scriere secretă, care este denumită criptare. Criptografia poate fi folosită și pentru a dovedi cunoașterea unui secret fără a dezvălui acel secret (semnătura digitală), sau pentru a dovedi autenticitatea datelor (amprentă digitală). Aceste tipuri de dovezi criptografice sunt instrumentele matematice critice pentru bitcoin

și folosite extensiv în aplicațiile bitcoin. În mod ironic, criptarea nu este o parte importantă a bitcoin, deoarece datele legate de comunicare și tranzacții nu sunt criptate și nu este nevoie să fie criptate pentru a proteja fondurile. În acest capitol vom prezenta o parte din criptografia folosită în bitcoin pentru a controla deținerea de fonduri, sub forma de chei, adrese, și portofele.

## Introducere

Deținerea de bitcoin este stabilită folosind *chei digitale, adrese bitcoin, și semnături digitale*. Cheile digitale nu sunt stocate de fapt în rețea, ci sunt în schimb create și stocate de către utilizatori într-un fișier, sau o bază de date simplă, numită *portofel*. Cheile digitale din portofelul unui utilizator sunt complet independente față de protocolul bitcoin și pot fi generate și administrate de către software-ul portofel al utilizatorului fără a face referire la lanțul-de-blocuri și fără acces la internet. Cheile oferă mijloacele pentru multe dintre proprietățile interesante ale bitcoin, inclusiv încrederea și controlul descentralizat, atestarea proprietății, și modelul de securitate dovedit-criptografic.

Majoritatea tranzacțiilor bitcoin necesită o semnătură digitală validă pentru a fi incluse în lanțul-de-blocuri, semnătura care poate fi generată doar folosind o cheie secretă; prin urmare, oricine are o copie a acelei chei are control asupra bitcoin. Semnătura digitală folosită pentru a cheltui fonduri este de asemenea denumită *martor*, un termen folosit în criptografie. Datele martor dintr-o tranzacție bitcoin depun mărturie despre cine deține cu adevărat fondurile care sunt cheltuite.

Cheile vin în perechi formate dintr-o cheie privată (secretă) și o cheie publică. Ne putem gândi la cheia publică ca la un număr de cont bancar, iar la cheia privată ca la codul PIN secret, sau ca la o semnătură pe un cec, care oferă controlul asupra contului. Aceste chei digitale sunt foarte rare să fie văzute de către utilizatorii bitcoin. În cea mai mare parte, ele sunt stocate în fișierul unui portofel și sunt gestionate de software-ul portofel bitcoin.

În secțiunea de plată a unei tranzacții bitcoin, cheia publică a destinatarului este reprezentată prin amprentă digitală, numită *adresă bitcoin*, care este folosită în același mod ca numele beneficiarului pe un cec (i.e. "Denumire Beneficiar"). În cele mai multe cazuri, o adresă bitcoin este generată și corespunde unei chei publice. Totuși, nu toate adresele bitcoin reprezintă chei publice; ele pot reprezenta de asemenea alți beneficiari cum ar fi scripturi, după cum vom vedea mai târziu în acest capitol. În acest mod, adresele bitcoin abstractizează destinatarul fondurilor, făcând destinațiile tranzacțiilor flexibile, similar cu cecurile de hârtie: un singur instrument de plată care poate fi folosit pentru a plăti în contul unor persoane, în contul unor companii, la plata facturilor, sau la extragerea de numerar. Adresa bitcoin este singura reprezentare a cheilor pe care utilizatorii o vor vedea în mod obișnuit, pentru că aceasta este partea pe care trebuie să o comunice celorlalți.

Mai întâi vom face o introducere în criptografie și vom explica matematica folosită în bitcoin. Apoi, vom analiza cum sunt generate, stocate și gestionate cheile. Vom trece în revistă diferitele formate de codificare folosite pentru a reprezenta cheile private, cheile publice, adresele, și adresele de scripturi. În cele din urmă vom analiza utilizările avansate ale cheilor: vanitate, semnătură multiplă, adrese de script și portofele de hârtie.

# Criptografie cu Cheie Publică și Criptomonedă

Criptografia cu cheie publică a fost inventată în anii 1970 și este baza matematică pentru securitatea calculatoarelor și a informațiilor.

De la inventarea criptografiei cu cheie publică, au fost descoperite câteva funcții matematice adecvate, cum ar fi exponenta numerelor prime și înmulțirea curbei eliptice. Aceste funcții matematice sunt practic ireversibile, însemnând ca sunt ușor de calculat într-o direcție, dar imposibil de calculat în sens invers. Pe baza acestor funcții matematice, criptografia permite crearea de secrete digitale și semnături digitale nefalsificabile. Bitcoin folosește înmulțirea curbei eliptice ca bază pentru criptografia sa.

În bitcoin, folosim criptografia cu cheie publică pentru a crea o pereche de chei ce controlează accesul la bitcoin. Perechea de chei constă dintr-o cheie privată și—derivată din ea—o cheie publică unică. Cheia publică este folosită pentru a primi fonduri, iar cheia privată este folosită pentru a semna tranzacții în scopul cheltuirii fondurilor.

Există o relație matematică între cheia publică și cea privată care permite utilizarea cheii private pentru a genera semnături pe mesaje. Aceste semnături pot fi validate cu cheia publică fără a dezvăluui cheia privată.

Când cheltuieste bitcoin, actualul proprietar își prezintă cheia publică și o semnătură (diferită de fiecare dată, dar ceeașă folosind aceeași cheie privată) ca parte a unei tranzacții pentru a cheltui acei bitcoin. Prin prezentarea cheii publice și a semnăturii, toți cei din rețeaua bitcoin pot verifica și accepta tranzacția ca fiind validă, confirmând că persoana care a transferat bitcoin îi deținea la momentul transferului.

**TIP** În majoritatea implementărilor de portofele, cheile private și publice sunt stocate împreună ca o *pereche de chei* pentru comoditate. Totuși, cheia publică poate fi calculată din cheia privată, deci este posibil să se stocheze doar cheia privată.

## Cheile Private și Publice

Un portofel bitcoin conține o colecție de perechi de chei, fiecare pereche constând dintr-o cheie privată și o cheie publică. Cheia privată ( $k$ ) este un număr, de obicei ales aleator. Din cheia privată, folosim înmulțirea curbei eliptice, o funcție criptografică unidirecțională, pentru a genera o cheie publică ( $K$ ). Din cheia publică ( $K$ ), folosim o funcție criptografică unidirecțională de rezumare (hash) pentru a genera o adresă bitcoin ( $A$ ). În această secțiune, vom începe prin a genera o cheie privată, vom analiza logica matematică din spatele curbei eliptice care este folosită pentru a transforma cheia privată într-o cheie publică, și în cele din urmă, vom genera o adresă bitcoin din cheia publică. Relația dintre cheia privată, cheia publică și adresa bitcoin este prezentată în [Cheie privată, cheie publică, și adresă bitcoin](#).

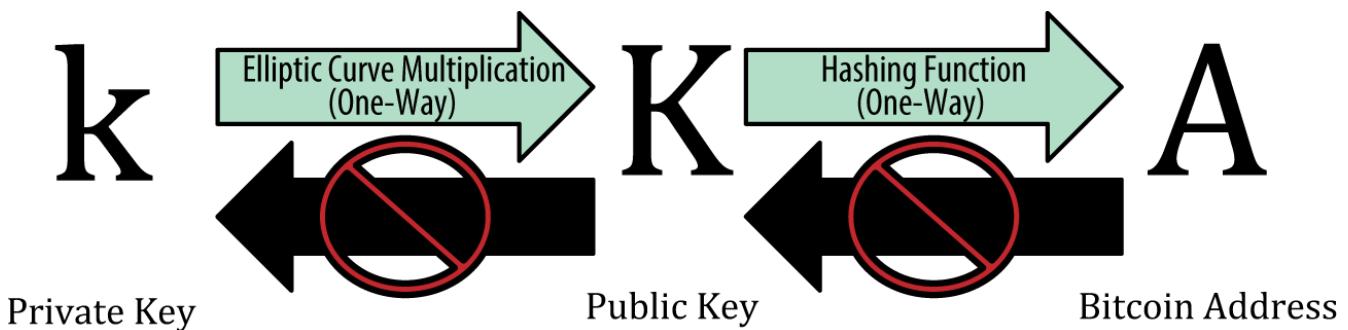


Figure 14. Cheie privată, cheie publică, și adresă bitcoin

### De ce se Folosește Criptografia Asimetrică (Chei Publice/Private)

De ce este criptografia asimetrică folosită în bitcoin? Nu este folosită pentru a "cripta" (a face secrete) tranzacțiile. Mai degrabă, proprietatea utilă a criptografiei asimetrice este capacitatea sa de a genera *semnături digitale*. O cheie privată poate fi aplicată pe amprenta digitală a unei tranzacții pentru a produce o semnătură numerică. Această semnătură poate fi produsă doar de cineva care cunoaște cheia privată. Totuși, oricine are acces la cheia publică și la amprenta tranzacției le poate folosi pentru a *verifica* semnătura. Această proprietate a criptografiei asimetrice face posibil ca oricine să poată verifica fiecare semnătură de pe fiecare tranzacție, asigurând în același timp că doar deținătorii cheilor private pot produce semnături valide.

### Chei Private

O cheie privată este pur și simplu un număr ales la întâmplare. Deținerea și controlul cheii private este sursa controlului utilizatorului asupra tuturor fondurilor asociate cu adresa bitcoin corespunzătoare. Cheia privată este folosită pentru a crea semnăturile necesare cheltuirii bitcoin, dovedind deținerea fondurilor folosite într-o tranzacție. Cheia privată trebuie să rămână secretă tot timpul, pentru că dezvăluirea ei este echivalentă cu a ceda controlul asupra monedelor bitcoin securizate de acea adresă. Cheia privată trebuie de asemenea să aibă o copie de rezervă și să fie protejată împotriva pierderii accidentale, deoarece dacă este pierdută nu poate fi recuperată, iar fondurile securizate de ea sunt pierdute și ele pentru totdeauna.

**TIP**

Cheia privată bitcoin este doar un număr. Puteți să vă alegeti cheile private aleator folosind doar o monedă, un creion și o foaie de hârtie: aruncați o monedă de 256 de ori și aveți cifrele binare ale unei chei private aleatorii pe care o puteți folosi într-un portofel bitcoin. Apoi cheia publică poate fi generată din cheia privată.

### Generarea unei chei private dintr-un număr aleator

Primul și cel mai important pas în generarea cheilor este găsirea unei surse sigure de entropie, sau de aleatoriu. Crearea unei chei bitcoin este în esență același lucru cu a spune: "Alegeți un număr între 1 și  $2^{256}$ ." Metoda exactă folosită pentru a alege acel număr nu contează cât timp nu este previzibil sau repetabil. Software-ul bitcoin folosește generatorul de numere aleatoare al sistemului de operare pentru a produce 256 de biți de entropie (de aleatoriu). De obicei, generatorul de numere aleatoare al sistemului de operare este inițializat de o sursă umană de aleatoriu, de aceea s-ar putea să vi se ceară să plimbați mouse-ul pe ecran pentru câteva secunde.

Mai precis, cheia privată poate fi orice număr între 0 și  $n - 1$  inclusiv, unde  $n$  este o constantă ( $n = 1.1578 * 10^{77}$ , ceva mai puțin decât  $2^{256}$ ) definită ca ordinea curbei eliptice folosite în bitcoin (vezi [Criptografia Curbei Eliptice Explicată](#)). Pentru a crea o astfel de cheie, alegem la întâmplare un număr format din 256 biți și verificăm dacă este mai mic decât  $n$ . În materie de programare, acest lucru este de obicei obținut pasând un sir mai mare de biți aleatori, creați dintr-o sursă criptografică sigură de aleatoriu, unui algoritm SHA256, care va produce în mod convenabil un număr pe 256 de biți. Dacă rezultatul este mai mic decât  $n$  atunci avem o cheie privată adecvată. În caz contrar încercăm din nou cu un alt număr aleator.

**WARNING**

Nu este indicat să scrieți propriul cod care să creeze un număr aleator sau să folosiți un generator "simplu" de numere aleatorii oferit de limbajul dumneavoastră de programare. Folosiți un pseudogenerator de numere aleatorii care este sigur din punct de vedere criptografic (CSPRNG) cu o sămână (seed) obținută dintr-o sursă sigură de entropie. Studiați documentația bibliotecii de generare a numerelor aleatorii pe care o alegeti pentru a vă asigura că este sigură din punct de vedere criptografic. Implementarea corectă a CSPRNG este critică pentru asigurarea securității cheilor.

Mai jos este o cheie privată (k) generată aleator afișată în format hexazecimal (256 de biți afișați sub forma a 64 de cifre hexazecimale, fiecare având 4 biți):

```
1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
```

**TIP**

Dimensiunea spațiului ( $2^{256}$ ) pentru cheia privată bitcoin este un număr inexplicabil de mare. În format zecimal este aproximativ  $10^{77}$ . Pentru comparație, este estimat că universul vizibil conține  $10^{80}$  atomi.

Pentru a genera o nouă cheie folosind clientul Bitcoin Core (vezi [Bitcoin Core: Implementarea Referință](#)), folosiți comanda `getnewaddress`. Din motive de securitate comanda afișează doar cheia publică, nu și pe cea privată. Pentru a cere `bitcoind` să expună cheia privată, folosiți comanda `dumpprivkey`. Comanda `dumpprivkey` afișează cheia privată codificată într-un format Base58 cu sumă de control numit *Formatul de Import pentru Portofel* (Wallet Import Format - WIF), pe care îl vom examina în detaliu în [Formate de Chei Private](#). Iată un exemplu de generare și afișare a unei chei private folosind cele două comenzi:

```
$ bitcoin-cli getnewaddress
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
$ bitcoin-cli dumpprivkey 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

Comanda `dumpprivkey` deschide portofelul și extrage cheia privată care a fost generată de comanda `getnewaddress`. Nu este posibil pentru `bitcoind` să afle cheia privată din cheia publică decât dacă ambele sunt stocate în portofel.

**TIP** Comanda `dumpprivkey` nu generează o cheie privată dintr-o cheie publică, acest lucru fiind imposibil. Comanda pur și simplu dezvăluie cheia privată care este deja cunoscută de către portofel și care a fost generată de comanda `getnewaddress`.

Puteți de asemenea folosi utilitarul din linia de comandă Bitcoin Explorer (vezi [Comenzi Bitcoin Explorer \(bx\)](#)) pentru a genera și afișa cheile private folosind comenziile `seed`, `ec-new`, și `ec-to-wif`:

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

## Chei Publice

Cheia publică este calculată din cheia privată folosind înmulțirea curbei eliptice, care este ireversibilă:  $K = k * G$ , unde  $k$  este cheia privată,  $G$  este un punct fix numit *punct generator*, și  $K$  este cheia publică rezultată. Operația inversă, cunoscută ca "găsirea logaritmului discret"—calcularea lui  $k$  dacă îl știți pe  $K$ —este la fel de dificilă ca a încerca toate valorile posibile pentru  $k$ , i.e. o căutare prin forță brută. Înainte de a demonstra cum să generăm o cheie publică dintr-o cheie privată, să ne uităm la criptografia curbei eliptice în detaliu.

**TIP** Înmulțirea curbei eliptice este un tip de funcție pe care criptografii o numesc funcție "ușă capcană": este ușor de calculat într-o direcție (înmulțirea) și imposibil de a calcula inversă (împărțirea). Proprietarul cheii private poate crea cu ușurință cheia publică și apoi poate să o împărtășească cu restul lumii știind că nimeni nu poate să inverseze funcția și să calculeze cheia privată din cheia publică. Acest artificiu matematic devine baza pentru semnăturile digitale securizate și nefalsificabile care dovedesc deținerea fondurilor bitcoin.

## Criptografia Curbei Eliptice Explicată

Criptografia curbei eliptice este un gen de criptografie asimetrică sau cu cheie publică bazată pe problema logaritmului discret exprimată de adăugarea și înmulțirea pe punctele unei curbe eliptice.

O curbă eliptică este un exemplu de curbă eliptică, similară cu cea folosită de bitcoin.

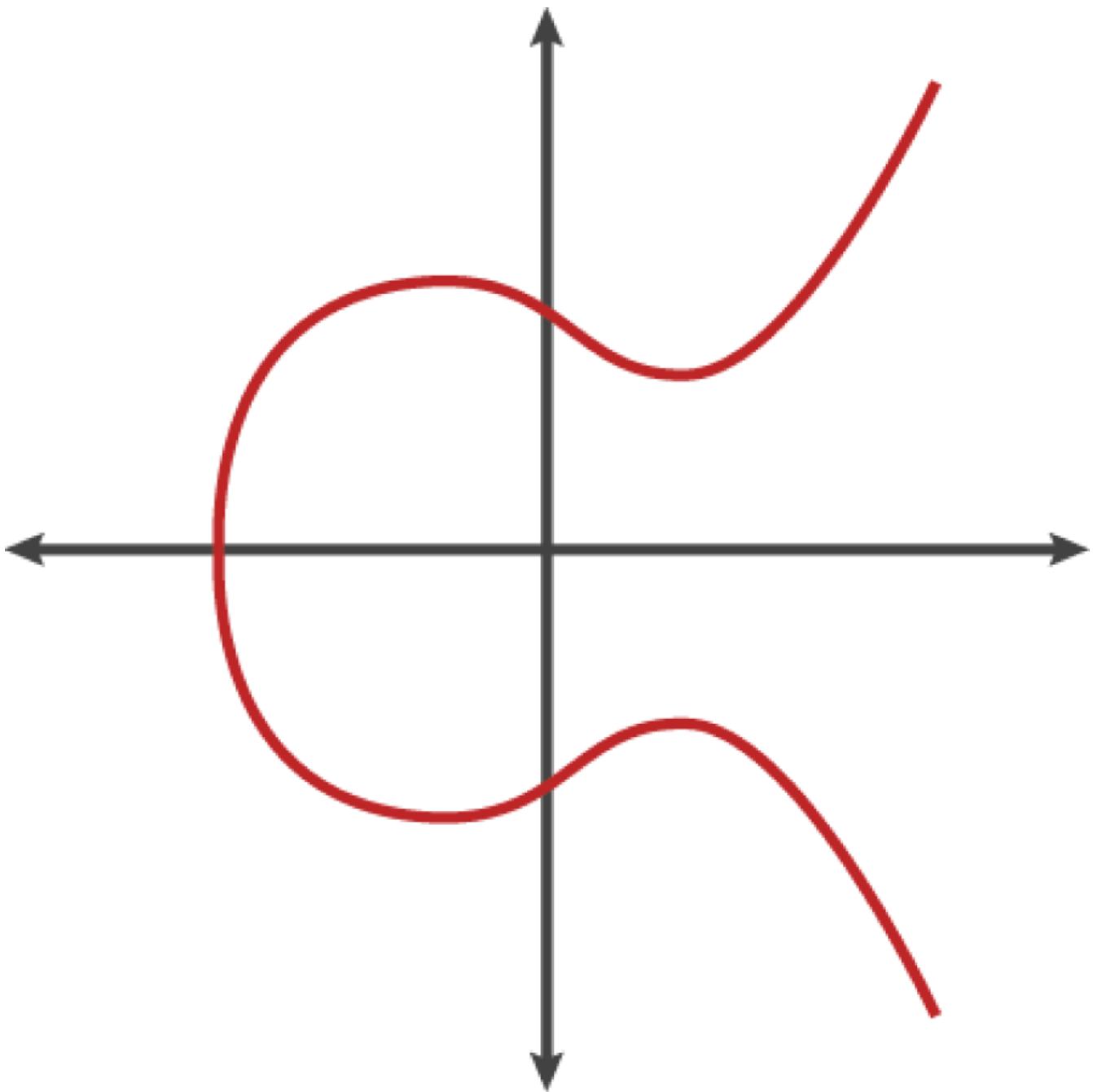


Figure 15. O curbă eliptică

Bitcoin folosește o anumită curbă eliptică și un set de constante matematice, după cum este definit într-un standard denumit *secp256k1*, stabilit de Institutul Național de Standarde și Tehnologie (NIST) din SUA. Curbă *secp256k1* este definită de următoarea funcție, care produce o curbă eliptică:

$$y^2 = (x^3 + 7) \text{ over } (\mathbb{F}_p) \quad (1)$$

sau

$$y^2 \bmod p = (x^3 + 7) \bmod p \quad (2)$$

Operația *mod p* (modulo număr prim p) indică faptul că această curbă este definită peste un domeniu finit de ordine primă p, scrisă în latexmath ca  $\mathbb{F}_p$ , unde  $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ , un număr prim foarte mare.

Deoarece această curbă este definită peste un domeniu finit de numere prime în loc să fie peste

domeniul numerelor reale, arată ca un model de puncte împrăștiate într-un spațiu bidimensional, ceea ce o face greu de vizualizat. Totuși, matematica este identică cu cea a unei curbe eliptice peste domeniul numerelor reale. Ca un exemplu, [Criptografia curbei eliptice: vizualizarea unei curbe eliptice peste  \$F\(p\)\$ , cu  \$p=17\$](#)  arată aceeași curbă eliptică peste un domeniu finit mult mai mic de ordine primă 17, înfățisând un model de puncte pe o grilă. Curba eliptică bitcoin  $secp256k1$  poate fi privită ca un model mult mai complex de puncte pe o grilă neimaginabil de mare.

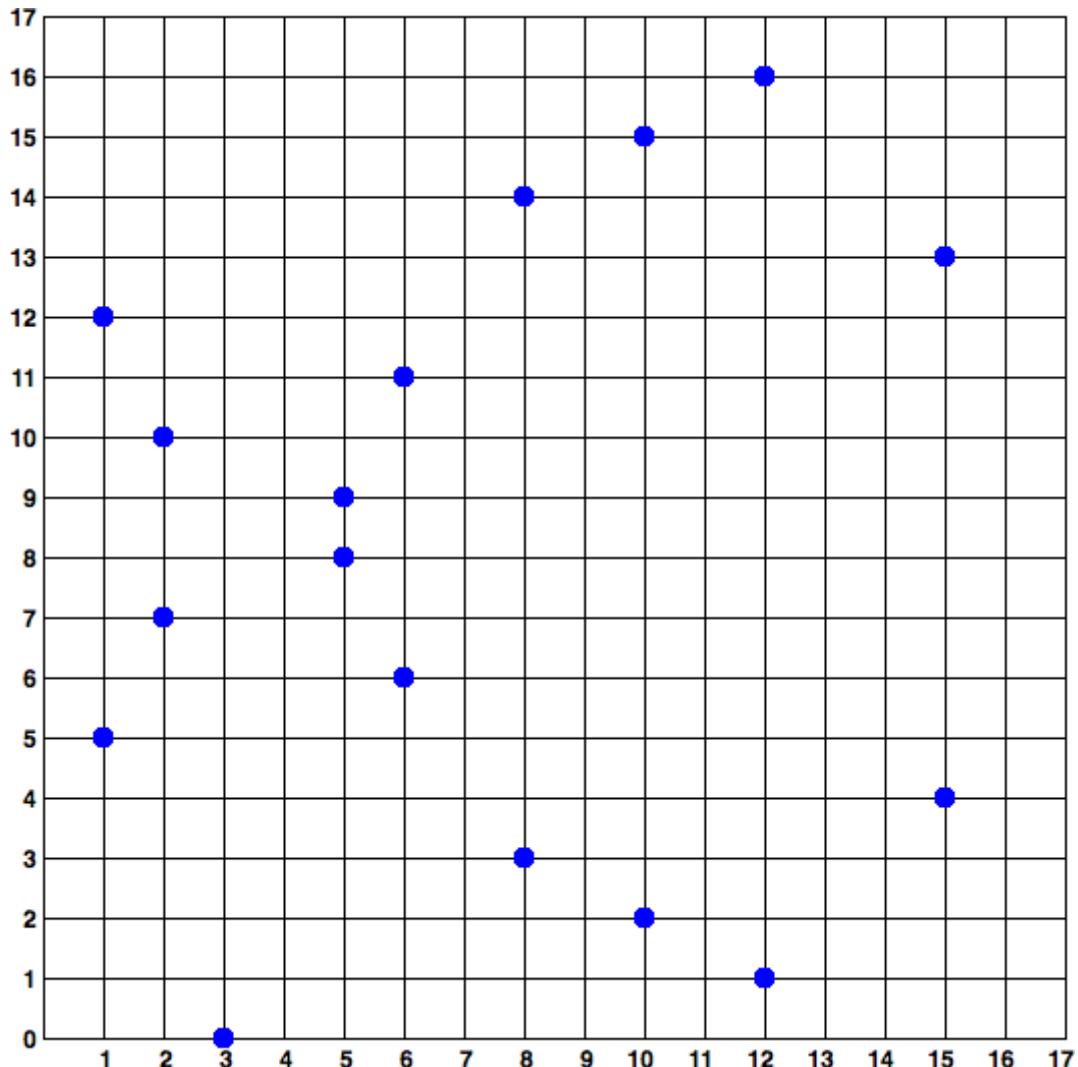


Figure 16. Criptografia curbei eliptice: vizualizarea unei curbe eliptice peste  $F(p)$ , cu  $p=17$

Deci, de exemplu, următorul punct P cu coordonate (x,y) este un punct pe curba  $secp256k1$ :

$$P = (55066263022277343669578718895168534326250603453777594175500187360389116729240, 32670510020758816978083085130507043184471273380659243275938904335757337482424)$$

Folosirea Python pentru a confirma că acest punct este pe curba eliptică arată cum puteți verifica asta singuri folosind Python:

### Example 8. Folosirea Python pentru a confirma că acest punct este pe curba eliptică

```
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 =
115792089237316195423570985008687907853269984665640564039457584007908834671663
>>> x =
55066263022277343669578718895168534326250603453777594175500187360389116729240
>>> y =
32670510020758816978083085130507043184471273380659243275938904335757337482424
>>> (x ** 3 + 7 - y**2) % p
0
```

În calculul curbei eliptice, există un punct numit "punct la infinit," care corespunde aproximativ cu rolul pe care zero îl are la adunare. Pe calculatoare, este uneori reprezentat de  $x = y = 0$  (care nu satisface ecuația curbei eliptice, dar este un caz separat care poate fi verificat ușor).

Există de asemenea un operator  $+$ , numit "adunare," care are unele proprietăți similare cu adunarea tradițională a numerelor reale pe care o învață copiii din clasele primare. Date fiind două puncte  $P_1$  și  $P_2$  pe curba eliptică, există un al treilea punct  $P_3 = P_1 + P_2$ , tot pe curba eliptică.

Geometric, acest punct  $P_3$  este calculat trasând o dreaptă între  $P_1$  și  $P_2$ . Această dreaptă va intersecta curba eliptică în exact încă un punct. Notați acest punct cu  $P_3' = (x, y)$ . Apoi reflectați-l pe axa  $x$  pentru a obține  $P_3 = (x, -y)$ .

Există câteva cazuri speciale care explică necesitatea existenței unui "punct la infinit".

Dacă  $P_1$  și  $P_2$  coincid, dreapta "dintre"  $P_1$  și  $P_2$  este tangentă la curbă în acest punct  $P_1$ . Această tangentă va intersecta curba în exact un punct. Puteți folosi analiza matematică pentru a calcula panta tangentei. Aceste mecanisme funcționează în mod interesant, chiar dacă restrângem domeniul punctelor de pe curbă doar la punctele cu coordonate numere întregi!

În unele cazuri (dacă  $P_1$  și  $P_2$  au aceleași valori  $x$ , dar valori  $y$  diferite), tangentă va fi verticală, caz în care  $P_3 = "punct la infinit"$ .

Dacă  $P_1$  este "punct la infinit," atunci  $P_1 + P_2 = P_2$ . Similar, dacă  $P_2$  este "punct la infinit," atunci  $P_1 + P_2 = P_1$ . Acest lucru arată cum "punctul la infinit" joacă rolul lui zero.

Se pare că  $+$  este asociativă, ceea ce înseamnă că  $(A + B) + C = A + (B + C)$ . Aceasta înseamnă că putem scrie  $A + B + C$  fără paranteze și fără ambiguitate.

Acum că am definit adunarea, putem defini înmulțirea în modul clasic în care extinde adunarea. Pentru un punct  $P$  aparținând curbei eliptice, dacă  $k$  este un număr întreg, atunci  $kP = P + P + \dots + P$  (de  $k$  ori). De reținut că uneori în acest caz  $k$  este numit în mod confuz "exponent".

## Generarea unei Chei Publice

Pornind de la o cheie privată sub forma unui număr generat aleator  $k$ , o multiplicăm cu un punct predeterminat  $G$  pe curbă numit *punct generator* pentru a produce un alt punct undeva pe curbă, care este cheia publică corespunzătoare  $K$ . Punctul generator este specificat în standardul *secp256k1* și este tot timpul același pentru toate cheile bitcoin.

$$K = k * G \quad (3)$$

unde  $k$  este cheia privată,  $G$  este punctul generator, iar  $K$  este cheia publică rezultată, un punct pe curbă. Deoarece punctul generator este tot timpul același pentru toți utilizatorii bitcoin, o cheie privată  $k$  înmulțită cu  $G$  va rezulta tot timpul în aceeași cheie publică  $K$ . Relația dintre  $k$  și  $K$  este fixă, dar poate fi calculată doar într-o direcție, de la  $k$  la  $K$ . De aceea o adresă bitcoin (derivată din  $K$ ) poate fi prezentată oricui și nu dezvăluie cheia privată a utilizatorului ( $k$ ).

**TIP** O cheie privată poate fi convertită într-o cheie publică, dar o cheie publică nu poate să fie convertită înapoi într-o cheie privată deoarece calculul funcționează doar într-un sens.

Implementând înmulțirea curbei eliptice, luăm cheia privată  $k$  generată anterior și o înmulțim cu punctul generator  $G$  pentru a găsi cheia publică  $K$ :

$$K = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD * G$$

Cheia publică  $K$  este definită ca un punct  $K = (x, y)$ :

$$K = (x, y)$$

unde,

$$x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A$$

$$y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB$$

Pentru a vizualiza multiplicarea unui punct cu un număr întreg, vom folosi curba eliptică mai simplă peste numere reale—de reținut, logica matematică e aceeași. Scopul nostru este să găsim multiplul  $kG$  al punctului generator  $G$ , care e același lucru cu adunarea lui  $G$  cu el însuși de  $k$  ori la rând. Pentru curbele eliptice, adunarea unui punct cu el însuși este echivalentul trasării unei drepte tangente în acel punct și aflarea punctului unde se intersecteză din nou cu curba, apoi oglindind acel punct față de axa x.

**Criptografia curbei eliptice: vizualizarea înmulțirii unui punct  $G$  cu un număr întreg  $k$  pe o curbă eliptică** prezintă procesul pentru derivarea  $G$ ,  $2G$ ,  $4G$ , ca o operație geometrică pe curbă.

**TIP** Bitcoin folosește [biblioteca secp256k1 C optimizată](#) pentru a efectua calculele matematice pentru curba eliptică.

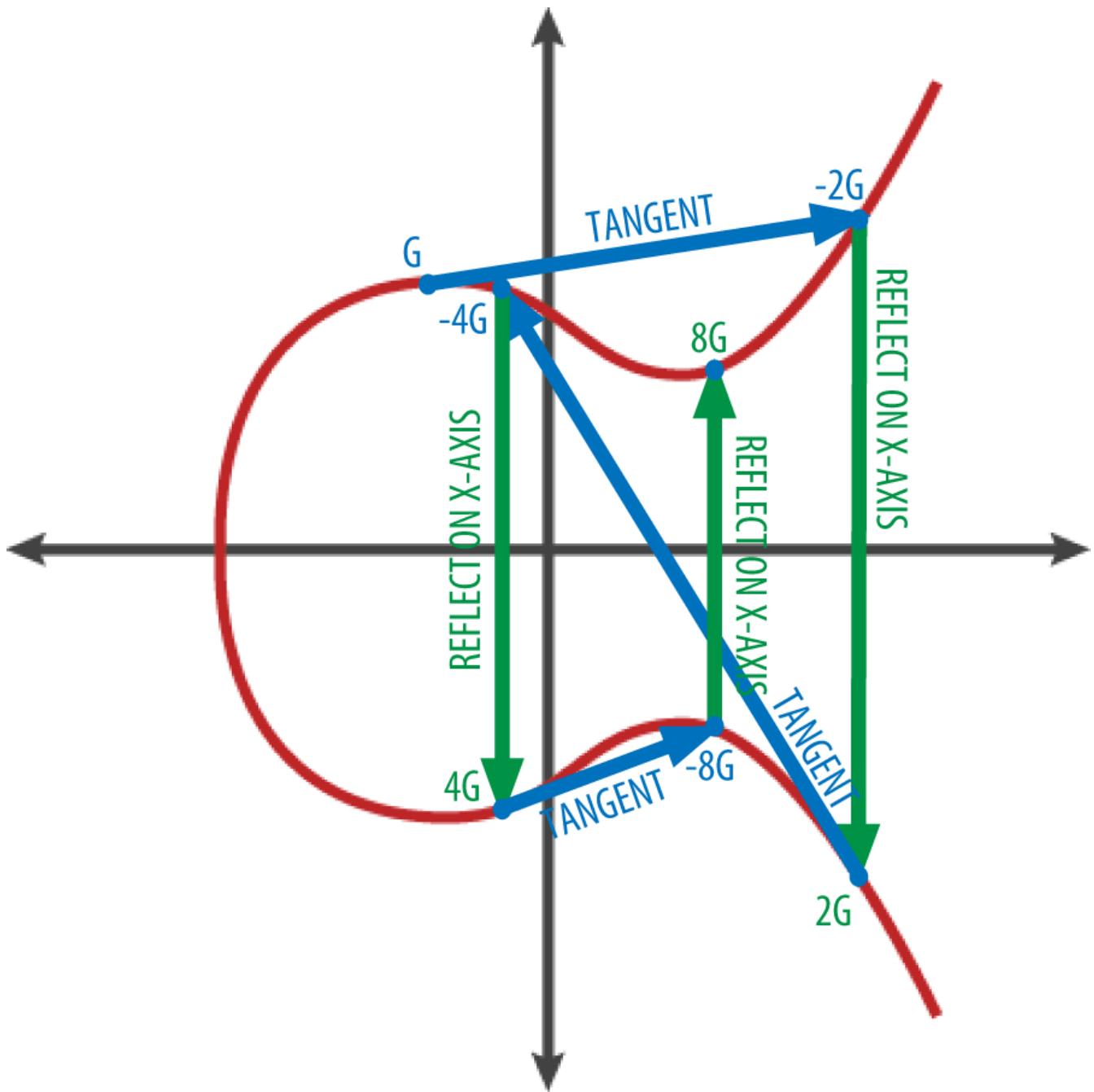


Figure 17. Criptografia curbei eliptice: vizualizarea înmulțirii unui punct  $G$  cu un număr întreg  $k$  pe o curbă eliptică

## Adrese Bitcoin

O adresă bitcoin este un sir de cifre și litere care poate fi prezentată oricui dorește să vă trimită bani. Adresele produse din cheile publice constau dintr-un sir de numere și litere, începând cu cifra "1". Iată un exemplu de adresă bitcoin:

1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy

Adresa bitcoin este cea care apare cel mai des într-o tranzacție ca "destinatar" al fondurilor. Dacă comparăm o tranzacție bitcoin cu un cec de hârtie, adresa bitcoin este beneficiarul, ceea ce scriem la linia "Denumire Beneficiar". La un cec de hârtie, acel beneficiar poate fi uneori numele titularului unui cont bancar, dar poate include și corporații, instituții, sau chiar numerar. Deoarece

cecurile de hârtie nu trebuie să specifice un număr de cont, ci mai degrabă să folosească un nume abstract pentru destinatarul fondurilor, ele sunt un instrument de plată foarte flexibil. Tranzacțiile bitcoin folosesc o abstracție similară, adresa bitcoin, pentru a le face foarte flexibile. O adresă bitcoin poate reprezenta deținătorul unei perechi de chei private/publice, sau poate reprezenta altceva, cum ar fi un script de plată, după cum vom vedea în [Plată-către-Rezumat-Script \(Pay-to-Script-Hash - P2SH\)](#). Deocamdată, vom examina cazul simplu, o adresă bitcoin care reprezintă și este derivată dintr-o cheie publică.

Adresa bitcoin este derivată dintr-o cheie publică prin folosirea unei funcții de rezumat unidirectională. Un algoritm de rezumare (hashing) sau pe scurt "algoritm rezumat" este o funcție unidirectională care produce o amprentă sau "rezumat" (hash) pentru o intrare de mărime variabilă. Funcțiile rezumat criptografice sunt folosite extensiv în bitcoin: în adresele bitcoin, în adresele scripturilor, și în algoritmul de minerit Dovadă-de-Lucru. Algoritmii folosiți pentru crearea unei adrese bitcoin dintr-o cheie publică sunt Secure Hash Algorithm (SHA) și RACE Integrity Primitives Evaluation Message Digest (RIPEMD), mai exact SHA256 și RIPEMD160.

Începând cu cheia publică  $K$ , calculăm rezumatul SHA256, iar apoi pentru rezultat calculăm rezumatul RIPEMD160, producând un număr pe 160 de biți (20 octeți):

$$A = \text{RIPEMD160}(\text{SHA256}(K)) \quad (4)$$

unde  $K$  este cheia publică, iar  $A$  este adresa bitcoin rezultată.

**TIP**

O adresă bitcoin *nu* este același lucru ca o cheie publică. Adresele bitcoin sunt derivate dintr-o cheie publică folosind o funcție unidirectională.

Adresele bitcoin sunt aproape întotdeauna codate ca "Base58Check" (vezi [Codare Base58 și Base58Check](#)), care folosește 58 de caractere (un sistem numeric în baza 58) și o sumă de control pentru fi mai ușor de citit pentru oameni, și pentru a proteja împotriva greșelilor în transcrierea și introducerea adreselor. Base58Check mai este folosit în multe alte moduri în bitcoin, oricând este nevoie ca un utilizator să citească și să transcrie corect un număr, cum ar fi o adresă bitcoin, o cheie privată, o cheie criptată, sau rezumatul unui script. În secțiunea următoare vom examina mecanismele codării și decodării Base58Check și reprezentările rezultate. [Cheia publică în adresă bitcoin: conversia unei chei publice într-o adresă bitcoin](#) ilustrează conversia unei chei publice într-o adresă bitcoin.

## Public Key to Bitcoin Address

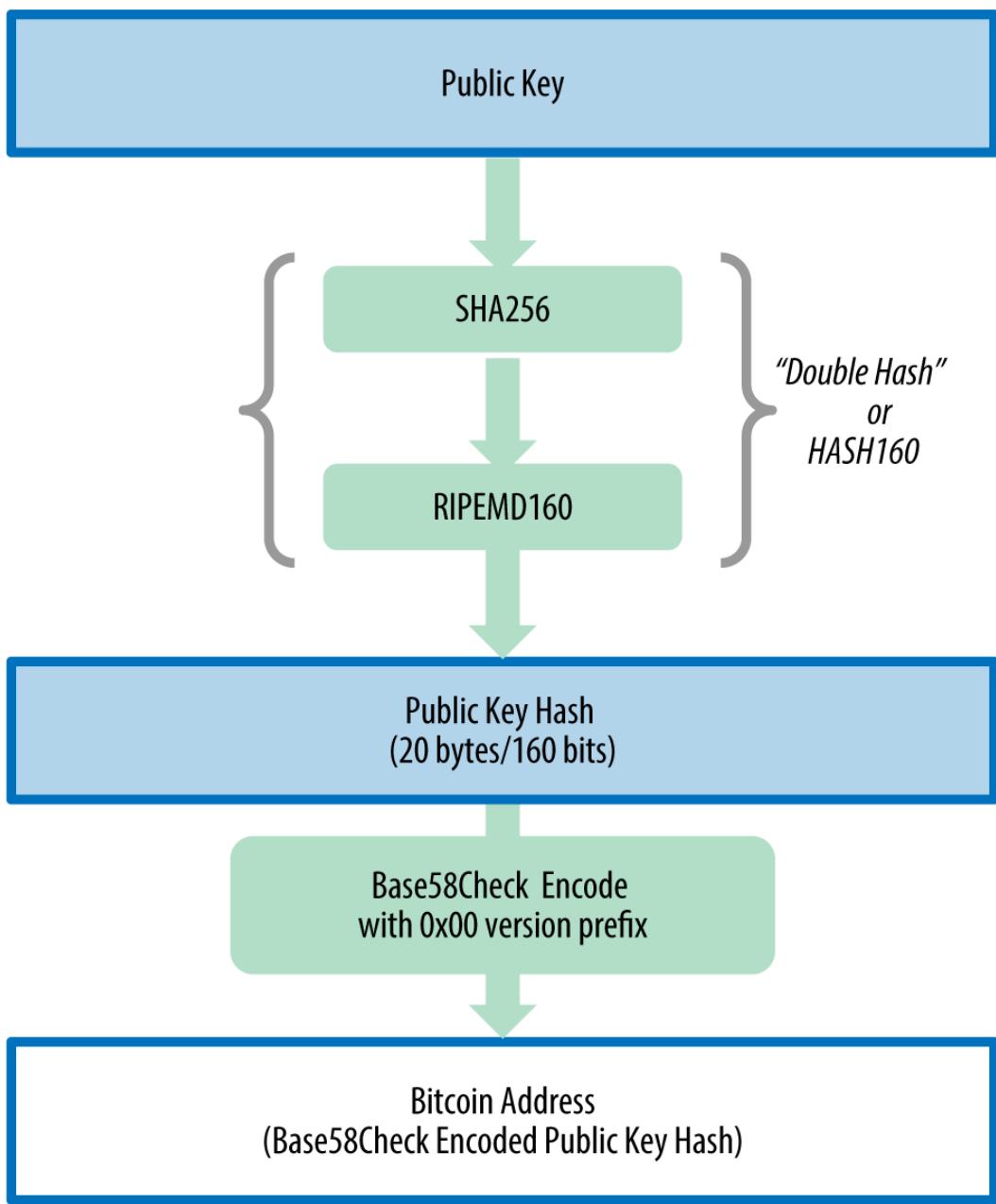


Figure 18. Cheia publică în adresă bitcoin: conversia unei chei publice într-o adresă bitcoin

### Codare Base58 și Base58Check

Pentru a reprezenta numerele lungi într-o formă compactă, folosind cât mai puține simboluri, multe programe folosesc un sistem de reprezentare alfanumeric într-o bază mai mare decât 10. De exemplu, în timp ce sistemul zecimal folosește 10 cifre de la 0 la 9, sistemul hexazecimal folosește 16 cifre având în plus literele de la A la F. Un număr în format hexazecimal e mai scurt decât unul în format zecimal. Reprezentarea Base64 este chiar și mai compactă folosind 26 de litere mici, 26 de litere mari, 10 cifre și încă două caractere, cum ar fi "+" și "/" pentru a transmite date binare pe

suporturi bazate pe text cum ar fi email-ul. Base64 este cel mai des folosit pentru a adăuga atașamente binare la email-uri. Base58 este un format de codare binară bazat pe text dezvoltat pentru folosirea în bitcoin și folosit în multe alte criptomonede. Oferă un echilibru între reprezentare compactă, lizibilitate, și detectarea și prevenirea erorilor. Base58 este un subset al Base64, ce folosește litere mici și mari, și numere, dar omițând unele caractere care sunt frecvent confundate între ele și pot părea identice când sunt afișate folosind anumite fonturi. Concret Base58 este Base64 fără 0 (zero), O (litera mare o), l (litera mică L), I (litera mare i), și simbolurile "+ și "/. Sau, și mai simplu, este un set de litere mici și mari și numere fără cele patru (0, O, l, I) deja menționate. [Alfabetul bitcoin Base58](#) prezintă alfabetul Base58 complet.

*Example 9. Alfabetul bitcoin Base58*

```
123456789ABCDEFGHIJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

Pentru a adăuga o verificare în plus împotriva greșelilor de dactilografie sau transcriere, Base58Check este un format de codare Base58 folosit frecvent în bitcoin, ce are încorporat un cod de verificare al erorilor. Suma de control constă din patru octeți în plus care sunt adăugați la sfârșitul datelor ce sunt codate. Suma de control este derivată din rezumatul datelor codate și prin urmare poate fi folosită la detectarea și prevenirea erorilor de dactilografie și transcriere. Când îi este prezentat un cod în format Base58Check software-ul de decodare va calcula suma de control a datelor și o va compara cu suma de control inclusă în cod. Dacă cele două nu se potrivesc, înseamnă că a fost introdusă o eroare, iar datele Base58Check sunt considerate invalide. Acest lucru previne ca adresele bitcoin care au fost introduse greșit să fie acceptate de software-ul portofelului ca destinații valide, o eroare care altfel ar fi rezultat în pierderea de fonduri.

Pentru a converti datele (un număr) în format Base58Check, întâi adăugam un prefix, numit "byte-ul de versiune", care indică ce tip de date sunt codate. De exemplu, în cazul unei adrese bitcoin prefixul este zero (0x00 în hexa), în timp ce prefixul folosit la codarea unei chei private este 128 (0x80 în hexa). O listă a celor mai des întâlnite prefixuri pentru versiuni este prezentată în [Prefixe de versiune Base58Check și exemplele codificate rezultate](#).

În continuare, calculăm suma de control "dublu-SHA", ceea ce înseamnă că aplicăm algoritmul rezumat SHA256 de două ori asupra rezultatului precedent (prefix și date):

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

Din rezumatul de 32 de octeți rezultat (rezumat-la-rezumat), extragem doar primii patru octeți. Acești patru octeți au rolul de cod de verificare al erorilor, sau sumă de control. Suma de control este concatenată la sfârșit.

Rezultatul este compus din trei elemente: un prefix, datele, și o sumă de control. Acest rezultat este codat folosind alfabetul Base58 descris anterior. [Codare Base58Check: un format Base58 pentru codarea datelor bitcoin, care este versionat și are sumă de control](#) ilustrează procesul de codare Base58.

## Base58Check Encoding

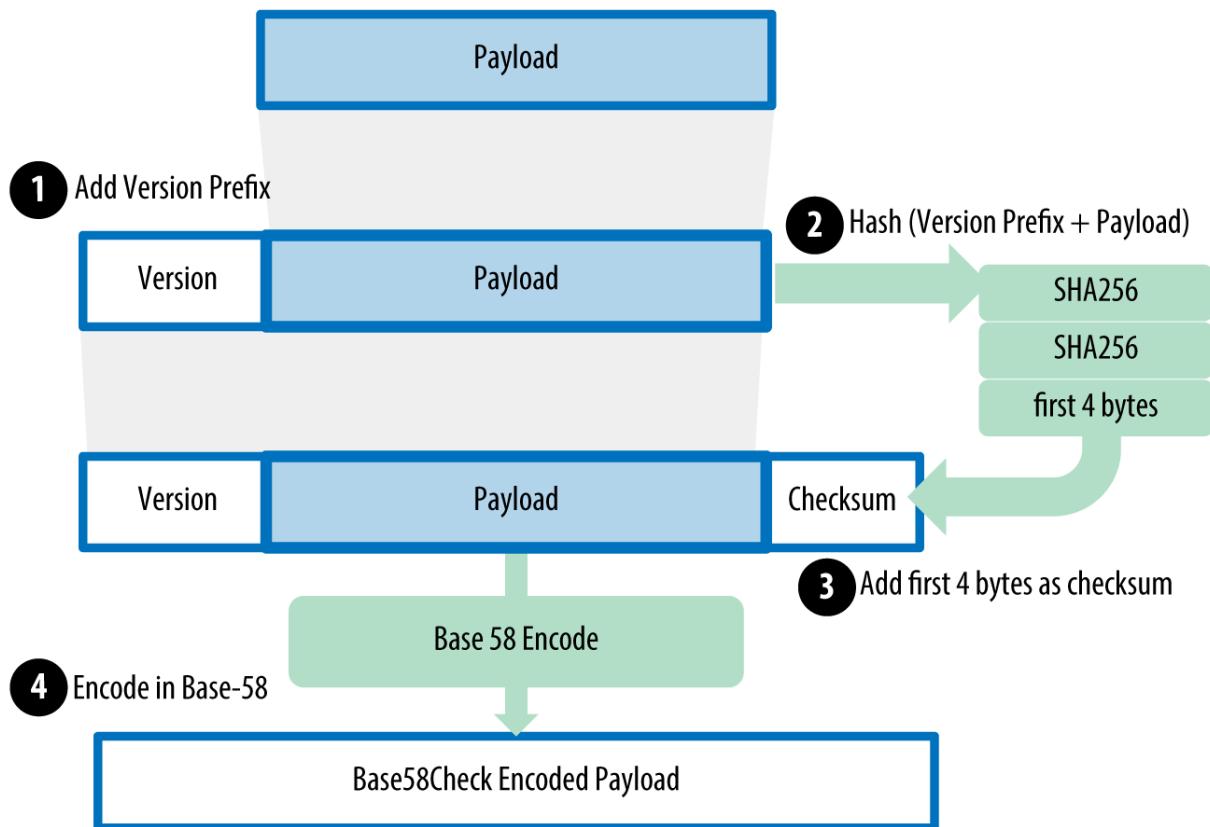


Figure 19. Codare Base58Check: un format Base58 pentru codarea datelor bitcoin, care este versionat și are sumă de control

În bitcoin, majoritatea datelor prezentate utilizatorului sunt codate în Base58Check pentru a le face compacte, ușor de citit, și ușor de detectat erori. Prefixul de versiune din codarea Base58Check este folosit pentru a crea formate ușor de diferențiat, care atunci când sunt codate în Base58 conțin anumite caractere la începutul textului generat. Aceste caractere facilitează identificarea tipului de date care sunt codate și a modului de utilizare al acestora. Acest lucru diferențiază, de exemplu, o adresă codată Base58Check care începe cu un 1 de o cheie privată WIF codată Base58Check care începe cu un 5. Unele exemple de prefixe de versiune precum și caracterele Base58 rezultate sunt prezentate în [Prefixe de versiune Base58Check și exemplele codificate rezultate](#).

Table 1. Prefixe de versiune Base58Check și exemplele codificate rezultate

Tip	Prefix de Versiune (hexa)	Prefix Base58 rezultat
Adresă Bitcoin	0x00	1
Adresă Plată-către-Rezumat-Script	0x05	3
Adresă Bitcoin Testnet	0x6F	m or n
Cheie Privată WIF	0x80	5, K, or L

Tip	Prefix de Versiune (hexa)	Prefix Base58 rezultat
Cheie Privată criptată BIP-38	0x0142	6P
Cheie Publică Extinsă BIP-32	0x0488B21E	xpub

## Formate de Chei

Atât cheile private cât și cele publice pot fi reprezentate într-o serie de formate diferite. Toate reprezentările codifică același număr, chiar dacă arată diferit. Aceste formate sunt folosite în primul rând pentru a facilita citirea și transcrierea cheilor fără a introduce erori.

### Formate de Chei Private

Cheia privată poate fi reprezentată într-o serie de formate diferite, toate corespunzând aceluiași număr de 256 de biți. [Reprezentările cheilor private \(formate de codare\)](#) prezintă trei formate comune utilizate pentru a reprezenta cheile private. Diferite formate sunt utilizate în circumstanțe diferite. Formatele hexazecimal și binar sunt folosite intern în software și rareori sunt afișate utilizatorului. Formatul WIF (Wallet Import Format) este folosit pentru importul/exportul cheilor între portofele și este folosit des în reprezentările codului QR al cheilor private.

Table 2. Reprezentările cheilor private (formate de codare)

Tip	Prefix	Descriere
Brut	Nici unul	32 de octeți
Hexa	Nici unul	64 de cifre hexazecimale
WIF	5	Codare Base58Check: Base58 cu prefix de versiune pe 128 biți, și 32-biți sumă de control
WIF-comprimat	K sau L	Ca mai sus, cu adăugarea sufixului 0x01 înainte de codare

[Exemplu: Aceiași cheie, formate diferite](#) prezintă cheia privată generată cu aceste trei formate.

Table 3. Exemplu: Aceiași cheie, formate diferite

Format	Cheia privată
Hexa	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
WIF-comprimat	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Toate aceste reprezentări sunt moduri diferite de a afișa același număr, aceeași cheie privată. Arată diferit, dar oricare dintre formate poate fi convertit cu ușurință în altul. Rețineți că formatul "binar brut" nu este prezentat în [Exemplu: Aceiași cheie, formate diferite](#) deoarece orice codare pentru afișare nu ar fi, prin definiție, binară.

Folosim comanda *wif-to-ec* a Bitcoin Explorer (vezi [Comenzi Bitcoin Explorer \(bx\)](#)) pentru a arăta că ambele chei WIF reprezintă aceeași cheie privată.

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

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

## Decodare din Base58Check

Comenziile Bitcoin Explorer (vezi [Comenzi Bitcoin Explorer \(bx\)](#)) facilitează scrierea scripturilor shell și a comenziilor înlăncuite (pipes) care manipulează cheile, adresele, și tranzacțiile bitcoin. Puteți folosi Bitcoin Explorer pentru a decoda formatul Base58Check de la linia de comandă.

Folosim comanda *base58check-decode* pentru a decoda cheia necomprimată:

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

Rezultatul conține cheia în câmpul "payload", prefixul de versiune WIF, și o sumă de control.

Observăm că câmpul "payload" al cheii comprimate are adăugat la sfârșit sufixul *01*, semnalând că cheia publică trebuie să fie compactată:

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

## Codificarea de la hexa la Base58Check

Pentru a codifica în Base58Check (opusul comenzi precendente), folosim comanda *base58check-encode* din Bitcoin Explorer (vezi [Comenzi Bitcoin Explorer \(bx\)](#)) și oferim cheia privată în hexa, urmată de prefixul de versiune WIF cu valoarea de 128:

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

### Codificare din hexa (cheie comprimată) în Base58Check

Pentru a codifica în Base58Check ca o cheie privată "comprimată" (vezi [Chei private comprimate](#)), adăugam sufixul *01* cheii în hexa și apoi codificăm la fel ca în secțiunea precedentă:

```
$ bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01 --version 128
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

Formatul WIF-comprimat începe cu un "K." Acest lucru indică faptul că cheia privată conținută are un sufix cu valoarea de "01" și va fi folosit pentru a produce doar chei publice comprimate (vezi [Chei publice comprimate](#)).

### Formate de chei publice

Cheile publice sunt, de asemenea, prezentate în moduri diferite, de obicei fie ca și chei publice *necomprimate* sau *comprimate*.

După cum am văzut anterior, cheia publică este un punct pe curba eliptică constând dintr-o pereche de coordonate  $(x, y)$ . De obicei este prezentată cu prefixul *04* urmat de două numere pe 256 de biți: unul pentru coordonata de pe axa *x*, iar celălalt pentru coordonata de pe axa *y*. Prefixul *04* este folosit pentru a deosebi cheile publice necomprimate de cheile publice comprimate care încep cu un *02* sau un *03*.

Iată cheia publică generată din cheia privată pe care am creat-o mai devreme, prezentată sub forma coordonatelor *x* și *y*:

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

Iată aceeași cheie publică prezentată ca un număr pe 520 de biți (130 de cifre hexa) având prefixul *04* urmat de coordonatele *x* și *y*, sub forma *04 x y*:

```
K = 04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

### Chei publice comprimate

Cheile publice comprimate au fost introduse în bitcoin pentru a reduce dimensiunea tranzacțiilor și pentru a economisi spațiu pe disc pe nodurile care stochează baza de date lanț-de-blocuri. Majoritatea tranzacțiilor includ cheia publică, care este necesară (dar nu suficientă) pentru a

valida legitimitatea proprietarului și pentru a cheltui bitcoin. Fiecare cheie publică necesită un spațiu de 520 de biți (prefix + x + y), care atunci când este înmulțit cu câteva sute de tranzacții per bloc, sau câteva zeci de mii de tranzacții pe zi, adaugă o cantitate semnificativă de date la lanțul-de-blocuri.

După cum am văzut în secțiunea [Chei Publice](#), o cheie publică este un punct  $(x, y)$  de pe o curbă eliptică. Pentru că această curbă exprimă o funcție matematică, un punct de pe curbă reprezintă o soluție a ecuației și, prin urmare, dacă știm valoarea coordonatei  $x$  putem calcula coordonata  $y$  rezolvând ecuația  $y^2 \bmod p = (x^3 + 7) \bmod p$ . Acest lucru ne permite să stocăm doar coordonata  $x$  a punctului cheii publice, omițând coordonata  $y$  și reducând marimea cheii și spațiul necesar de stocare cu 256 de biți. O reducere de aproape 50% a dimensiunii în fiecare tranzacție înseamnă mult spațiu economisit pe termen lung.

În timp ce cheile publice necomprimate au prefixul **04**, cheile publice comprimate încep fie cu un prefix de **02** fie cu unul de **03**. Să ne uităm de ce există două prefixe posibile: pentru că partea stângă a ecuației este  $y^2$ , soluția pentru  $y$  este o rădăcină pătrată, care poate avea o valoare pozitivă sau negativă. Vizual, aceasta înseamnă că coordonata  $y$  rezultată poate fi deasupra sau sub axa  $x$ . După cum puteți vedea din graficul curbei eliptice în [O curbă eliptică](#), curba este simetrică, însemnând că este reflectată ca o oglindă de către axa  $x$ . Deci, chiar dacă putem omite coordonata  $y$  tot trebuie să stocăm semnul lui  $y$  (pozitiv sau negativ); sau în alte cuvinte, trebuie să ținem minte dacă a fost deasupra sau sub axa  $x$  pentru că fiecare din aceste opțiuni reprezintă un punct diferit și o cheie publică diferită. Când calculăm curba eliptică în aritmetică binară pe domeniul finit de numere prime de ordinul  $p$ , coordonata  $y$  este fie pară fie impară, care corespunde cu semnul negativ/pozitiv explicitat mai devreme. Prin urmare, pentru a face distincția între cele două valori posibile ale lui  $y$ , stocăm o cheie publică comprimată cu prefixul **02** dacă  $y$  este par, și **03** dacă este impar, permitând software-ului să deducă corect coordonata  $y$  din coordonata  $x$  și să decomprime cheia publică la coordonatele complete ale punctului. Compresia cheii publice este ilustrată în [Comprimarea cheii publice](#).

Iată aceeași cheie publică generată anterior, prezentată ca și cheie publică comprimată, stocată pe 264 de biți (66 de cifre hexa) cu prefixul **03** indicând că coordonata  $y$  este impară:

```
K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
```

Această cheie publică comprimată corespunde aceleiași chei private, însemnând că este generată din aceeași cheie privată. Cu toate acestea, arată diferit față de o cheie publică necomprimată. și mai important, dacă convertim această cheie publică într-o adresă bitcoin folosind funcția de rezumat dublu (*RIPEMD160(SHA256(K))*) vom produce o adresă bitcoin *diferită*. Acest lucru poate crea confuzie, deoarece înseamnă că o singură cheie privată poate produce o cheie publică exprimată în două formate diferite (comprimat și necomprimat) care vor produce la rândul lor două adrese bitcoin diferite. Totuși, cheia privată este identică pentru ambele adrese bitcoin.

## Public Key Compression

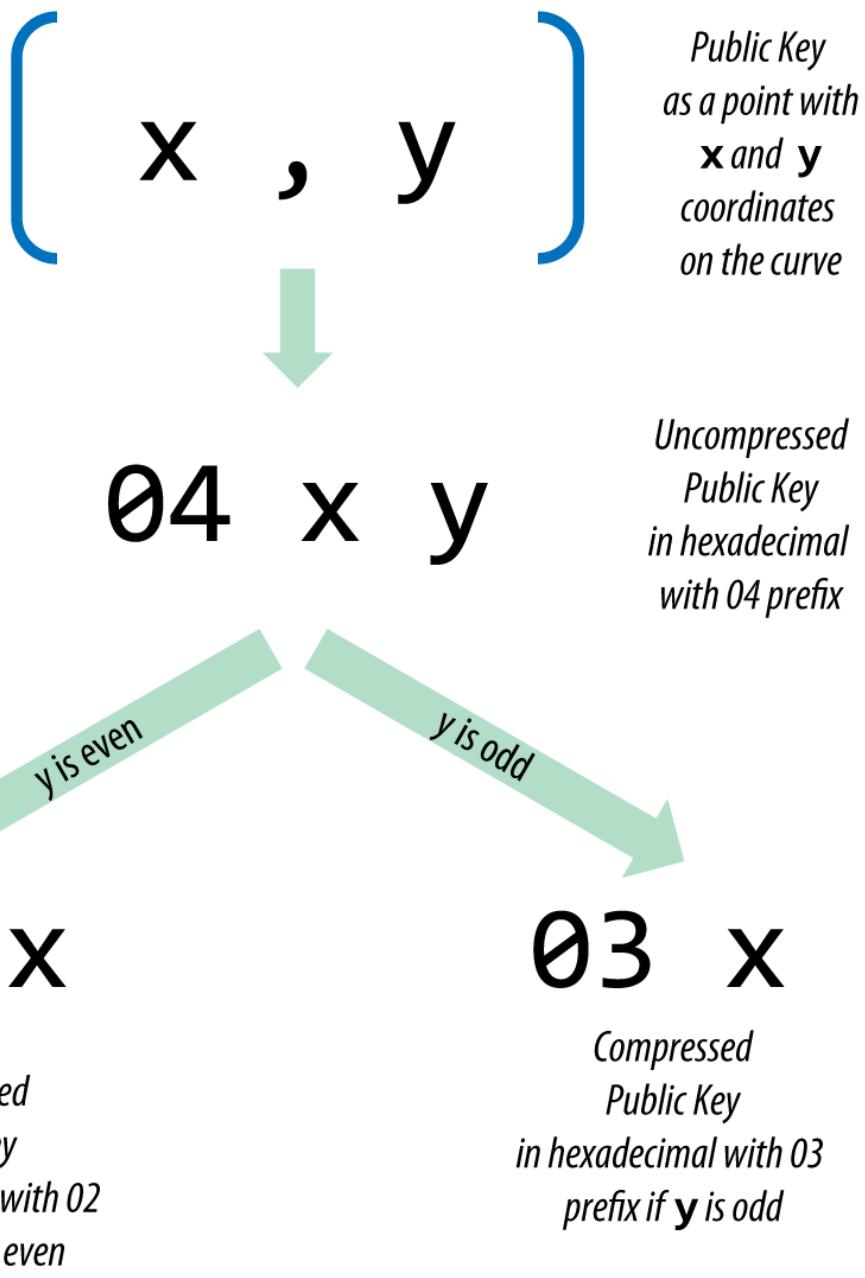


Figure 20. Comprimarea cheii publice

Cheile publice comprimate devin gradual standardul în rândul clienților bitcoin, ceea ce are un impact semnificativ în reducerea dimensiunii tranzacțiilor și, prin urmare, a lanțului-de-blocuri. Cu toate acestea, nu toți clienții acceptă încă chei publice comprimate. Clientii mai noi care acceptă cheile publice comprimate trebuie să țină cont de tranzacțiile de la clienți mai vechi care nu acceptă chei publice comprimate. Acest lucru este deosebit de important când o aplicație portofel importă cheile de la o altă aplicație portofel, pentru că noul portofel trebuie să scaneze lanțul-de-blocuri pentru a găsi tranzacțiile corespunzătoare acestor chei importate. Pentru ce

adresă bitcoin ar trebui să scaneze portofelul? Adresa bitcoin produsă din chei publice necomprimate, sau adresa bitcoin produsă de chei publice comprimate? Ambele sunt adrese bitcoin valide, iar cheia privată poate semna pentru ambele, dar sunt adrese diferite!

Pentru a rezolva această dilemă, atunci când cheile private sunt exportate dintr-un portofel, formatul WIF care este folosit pentru a le reprezenta este implementat diferit în portofelele bitcoin mai noi, pentru a indica că aceste chei private au fost folosite pentru a produce chei publice *comprimate* și, prin urmare, adrese publice *comprimate*. Acest lucru permite portofelului care le importă să facă diferență între chei private provenite de la portofele mai vechi sau mai noi și să caute în lanțul-de-blocuri tranzacțiile cu adrese bitcoin corespunzând cheii publice necomprimate sau, respectiv, comprimate. Să ne uitam mai în detaliu la cum funcționează, în secțiunea următoare.

### Chei private comprimate

În mod ironic, termenul "cheie privată comprimată" este un termen impropriu, deoarece când o cheie privată este exportată în format WIF-comprimat este de fapt cu un octet mai *lungă* decât o cheie privată "necomprimată". Acest lucru se datorează faptului că cheia privată are adăugat un sufix de un octet (prezentat ca 01 în hexa în [Exemplu: Aceiași cheie, formate diferite](#)), ceea ce indică că cheia privată provine de la un portofel mai nou și ar trebui să fie utilizată pentru a produce doar chei publice comprimate. Cheile private nu sunt comprimate și nici nu pot fi comprimate. Termenul "cheie privată comprimată" înseamnă de fapt "cheie privată din care doar chei publice comprimate ar trebui să fie derivate," în timp ce "cheie privată necomprimată" înseamnă cu adevărat "cheie privată din care doar chei publice necomprimate ar trebui să fie derivate." Pentru a evita confuziile ne vom referi doar la formatul de export ca "WIF-comprimat" sau "WIF", dar nu la cheia privată ca fiind "comprimată."

[Exemplu: Aceiași cheie, formate diferite](#) arată aceeași cheie, codificată în formatele WIF și WIF-comprimat.

Table 4. [Exemplu: Aceiași cheie, formate diferite](#)

Format	Cheia privată
Hexa	1E99423A4ED27608A15A2616A2B0E9E52CED330 AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2Jpb nkeyhfsYB1Jcn
Hexa-comprimat	1E99423A4ED27608A15A2616A2B0E9E52CED330 AC530EDCC32C8FFC6A526AEDD01
WIF-comprimat	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGm f6YwgdGWZgawvrtJ

Rețineți că, în formatul hexa-comprimat, cheia privată are un octet în plus la sfârșit (în hexa). În timp ce prefixul de versiune codificat Base58 e același (0x80) pentru ambele formate WIF și WIF-comprimat, adăugarea unui octet la sfârșitul numărului face ca primul caracter al codificării Base58 să se schimbe de la un 5 la fie un *K* sau un *L*. Gândiți-vă la acest lucru ca la echivalentul în Base58 a diferenței dintre numărul 99 și 100 în format zecimal. În timp ce 100 este cu o cifră mai lungă decât 99, în același timp are un prefix de 1 în loc de un prefix de 9. Când lungimea se schimbă,

este afectat și prefixul. În Base58, prefixul 5 se schimbă într-un  $K$  sau  $L$  deoarece lungimea numărului crește cu un octet.

Nu uitați, aceste formate *nu* sunt folosite în mod interschimbabil. Într-un portofel mai nou care implementează chei publice comprimate, cheile private vor fi exportate doar în format WIF-comprimat (cu un prefix de  $K$  sau  $L$ ). Dacă portofelul are o versiune mai veche și nu folosește chei publice comprimate, cheile private vor fi exportate doar ca WIF (cu un prefix de 5). Scopul este de a semnala portofelului care importă aceste chei private dacă trebuie să caute în lanțul-de-blocuri după chei publice și adrese comprimate sau necomprimate.

Dacă un portofel bitcoin este capabil să implementeze chei publice comprimate, le va folosi pe acelea în toate tranzacțiile. Cheile private din portofel vor fi folosite pentru a deriva punctul cheii publice pe curbă, care va fi comprimat. Cheile publice comprimate vor fi folosite pentru a genera adrese bitcoin, iar acelea vor fi folosite în tranzacții. Când exportați cheile private dintr-un portofel nou care implementează chei publice comprimate, fișerul WIF este modificat, prin adăugarea la cheia privată a unui octet sufix cu valoarea de 01. Cheia privată codificată în Base58Check este numită "WIF comprimat" și începe cu literele  $K$  sau  $L$ , în loc să înceapă cu un "5" cum este cazul pentru cheile necomprimate codificate WIF din portofelele mai vechi.

**TIP** "Chei private comprimate" este un termen impropriu! Ele nu sunt comprimate deloc; mai degrabă WIF-comprimat înseamnă că cheile ar trebui să fie folosite doar pentru a deriva chei publice comprimate și adresele lor bitcoin corespunzătoare. În mod ironic, o cheie privată codificată în "WIF-comprimat" este cu un octet mai lungă deoarece are adăugat sufixul 01 pentru a o deosebi de una "necomprimată".

## Implementarea cheilor și adreselor în C++

Să ne uităm la procesul complet de creare a unei adrese bitcoin, de la o cheie privată, la o cheie publică (un punct pe curba eliptică), la o adresă dublu rezumată, și în final, codificarea Base58Check. Codul C++ de la [Crearea unei adrese bitcoin codificate Base58Check dintr-o cheie privată](#) arată pas cu pas procesul complet, de la cheia privată la adresa bitcoin codificată Base58Check. Exemplul de cod folosește biblioteca libbitcoin introdusă în [Clienți, Biblioteci și Instrumente Alternative](#) pentru niște funcții ajutătoare.

Example 10. Crearea unei adrese bitcoin codificate Base58Check dintr-o cheie privată

```
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Private secret key string as base16
    bc::ec_secret decoded;
    bc::decode_base16(decoded,
                      "038109007313a5807b2ecc082c8c3fbb988a973cacf1a7df9ce725c31b14776");

    bc::wallet::ec_private secret(
        decoded, bc::wallet::ec_private::mainnet_p2kh);

    // Get public key.
    bc::wallet::ec_public public_key(secret);
    std::cout << "Public key: " << public_key.encoded() << std::endl;

    // Create Bitcoin address.
    // Normally you can use:
    //     bc::wallet::payment_address payaddr =
    //         public_key.to_payment_address(
    //             bc::wallet::ec_public::mainnet_p2kh);
    // const std::string address = payaddr.encoded();

    // Compute hash of public key for P2PKH address.
    bc::data_chunk public_key_data;
    public_key.to_data(public_key_data);
    const auto hash = bc::bitcoin_short_hash(public_key_data);

    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;
}
```

Codul folosește o cheie privată predefinită pentru a produce aceeași adresă bitcoin de fiecare dată când este rulat, după cum este prezentat în [Compilarea și rularea codului addr](#).

*Example 11. Compilarea și rularea codului addr*

```
# Compile the addr.cpp code
$ g++ -o addr addr.cpp -std=c++11 $(pkg-config --cflags --libs libbitcoin)
# Run the addr executable
$ ./addr
Public key: 0202a406624211f2abbdc68da3df929f938c3399dd79fac1b51b0e4ad1d26a47aa
Address: 1PRTTaJesdNovgne6Ehcd1fpEdX7913CK
```

**TIP** Codul de la [Compilarea și rularea codului addr](#) produce o adresă bitcoin (1PRTT...) dintr-o cheie publică *comprimată* (vezi [Chei publice comprimate](#)). Dacă foloseați în loc cheia publică necomprimată, ar fi produs o adresă bitcoin diferită (14K1y...).

## Implementarea Cheilor și Adreselor în Python

Cea mai cuprinzătoare bibliotecă bitcoin în Python este [pybitcointools](#) scrisă de Vitalik Buterin. În [Generarea și formatarea cheilor și adreselor folosind biblioteca pybitcointools](#), folosim biblioteca pybitcointools (importată ca "bitcoin") pentru a genera și afișa chei și adrese în diferite formate.

Example 12. Generarea și formatarea cheilor și adreselor folosind biblioteca pybitcointools

```
from __future__ import print_function
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
compressed_prefix = '02' if (public_key_y % 2) == 0 else '03'
hex_compressed_public_key = compressed_prefix + (bitcoin.encode(public_key_x, 16).zfill(64))
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))
```

▶ [Running key-to-address-ecc-example.py](#) arată rezultatul produs de rularea acestui script.

*Example 13. Running key-to-address-ecc-example.py*

```
$ python key-to-address-ecc-example.py
Private Key (hex) is:
3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa6
Private Key (decimal) is:
26563230048437957592232553826663696440606756685920117476832299673293013768870
Private Key (WIF) is:
5JG9hT3beGTJuUAmCQEmNaxAuMacCTfXuw1R3FCXig23RQHMr4K
Private Key Compressed (hex) is:
3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa601
Private Key (WIF-Compressed) is:
KyBsPXxTuVD82av65KZkrGrWi5qLMah5SdNq6uftawDbgKa2wv6S
Public Key (x,y) coordinates is:
(41637322786646325214887832269588396900663353932545912953362782457239403430124L,
16388935128781238405526710466724741593761085120864331449066658622400339362166L)
Public Key (hex) is:
045c0de3b9c8ab18dd04e3511243ec2952002dbfadc864b9628910169d9b9b00ec&#x21b5;
243bcefdd4347074d44bd7356d6a53c495737dd96295e2a9374bf5f02ebfc176
Compressed Public Key (hex) is:
025c0de3b9c8ab18dd04e3511243ec2952002dbfadc864b9628910169d9b9b00ec
Bitcoin Address (b58check) is:
1thMirt546nngXqyPEz532S8fLwbozud8
Compressed Bitcoin Address (b58check) is:
14cxpo3MBCYYWCgF74SWTdcmxipnGUsPw3
```

Un script care demonstrează calcului curbei eliptice folosită pentru cheile bitcoin este un alt exemplu, folosind biblioteca Python ECDSA pentru calculul curbei eliptice și fără a folosi vreo bibliotecă bitcoin specializată.

*Example 14. Un script care demonstrează calculul curbei eliptice folosită pentru cheile bitcoin*

```

print("Elliptic Curve point:", point)

print("BTC public key:", get_point_pubkey(point))

# 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)

```

Instalarea bibliotecii Python ECDSA și rularea scriptului `ec_math.py` arată rezultatul produs de rularea acestui script.

**WARNING**

Un script care demonstrează calcului curbei eliptice folosită pentru cheile bitcoin folosește `os.urandom`, care reflectă un generator de numere aleatoare securizat criptografic (CSRNG) furnizat de sistemul de operare pe care rulează. Atenție: în funcție de sistemul de operare, `os.urandom` s-ar putea să nu fie destul de securizat și s-ar putea să *nu* fie potrivit pentru generarea de chei bitcoin de calitate.

*Example 15. Instalarea bibliotecii Python ECDSA și rularea scriptului `ec_math.py`*

```

$ # 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:
38090835015954358862481132628887443905906204995912378278060168703580660294000
EC point:
(70048853531867179489857750497606966272382583471322935454624595540007269312627,
105262206478686743191060800263479589329920209527285803935736021686045542353380)
BTC public key: 029ade3effb0a67d5c8609850d797366af428f4a0d5194cb221d807770a1522873

```

## Chei și Adrese Avansate

În secțiunea următoare vom analiza forme avansate de chei și adrese, cum ar fi chei private criptate, adrese script și semnătură multiplă, adrese de vanitate și portofele de hârtie.

### Rezumat Plată-către-Script (Pay-to-Script Hash - P2SH) și Adrese Semnătură Multiplă

După cum știm, adresele bitcoin tradiționale încep cu cifra "1" și provin din cheia publică, care este derivată din cheia privată. Deși oricine poate trimite bitcoin la o adresă care începe cu "1", acel bitcoin poate fi cheltuit doar de cineva care prezintă semnătura corespunzătoare cheii private și rezumatul cheii publice.

Adresele bitcoin care încep cu cifra "3" sunt adrese rezumat plată-către-script (pay-to-script hash - P2SH), uneori numite în mod eronat adrese semnătură multiplă (multisig). Ele desemnează beneficiarul unei tranzacții bitcoin ca fiind rezumatul unui script, în loc de proprietarul unei chei publice. Funcționalitatea a fost introdusă în ianuarie 2012 cu BIP-16 (vezi [Propunerile de Îmbunătățire Bitcoin](#)), și a fost adoptată pe scară largă deoarece oferă posibilitatea de a adăuga funcționalitate adresei în sine. Spre deosebire de tranzacțiile care "trimit" fonduri la adrese bitcoin tradiționale (care încep cu "1") cunoscute și ca rezumat-plată-către-cheie-publică (pay-to-public-key-hash P2PKH), fondurile trimise la adrese care încep cu "3" necesită ceva mai mult decât prezentarea unei chei publice și a unei semnături provenite de la cheia privată pentru a dovedi proprietatea. Cerințele sunt specificate la momentul când adresa este creată, în cadrul scriptului, iar toate intrările către această adresă vor fi subordonate acelorași cerințe.

O adresă P2SH este creată dintr-un script de tranzacție, care definește cine poate cheltui ieșirea unei tranzacții (pentru mai multe detalii vezi [Plată-către-Rezumat-Script \(Pay-to-Script-Hash - P2SH\)](#)). Codificarea unei adrese P2SH implică folosirea aceleiași funcții dublu-rezumat ca cea folosită la crearea unei adrese bitcoin, doar că aplicată pe script în loc de cheia publică.

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

"Rezumatul scriptului" rezultat este codificat cu Base58Check cu un prefix de versiune de 5, ceea ce rezultă într-o adresă codificată care începe cu un 3. Un exemplu de o adresă P2SH este *3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM*, care poate fi derivată folosind comenzi Bitcoin Explorer *script-encode*, *sha256*, *ripemd160*, și *base58check-encode* (vezi [Comenzi Bitcoin Explorer \(bx\)](#)) după cum urmează:

```
$ echo \
'DUP HASH160 [89abcdefabbaabbaabbaabbaabbaabba] EQUALVERIFY CHECKSIG' > script
$ bx script-encode < script | bx sha256 | bx ripemd160 \
| bx base58check-encode --version 5
3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM
```

**TIP** P2SH nu este neapărat același lucru cu o tranzacție standard cu semnătură multiplă. O adresă P2SH *cel mai des* reprezintă un script cu semnătură multiplă, dar ar putea de asemenea să reprezinte un script care să codifice alte tipuri de tranzacții.

## Adrese cu mai multe semnături (multisig) și P2SH

În prezent, cea mai întâlnită implementare a funcției P2SH este scriptul adresă cu mai multe semnături. După cum sugerează numele, scriptul necesită mai mult de o semnătură pentru a dovedi deținerea fondurilor și a le putea cheltui. Funcționalitatea bitcoin pentru semnătură multiplă este gândită să necesite  $M$  semnături (cunoscut sub denumirea de prag) dintr-un total de  $N$  chei, cunoscut drept semnătură multiplă (multisig)  $M$ -din- $N$ , unde  $M$  este mai mic sau egal cu  $N$ . De exemplu, Bob, proprietarul cafenelei de la [Introducere](#) ar putea să folosească o adresă cu semnătură multiplă care necesită 1-din-2 semnături de la o cheie care îi aparține lui și una care îi aparține soției lui, asigurându-se că oricare dintre ei poate semna pentru a cheltui ieșirea unei tranzacții blocate la această adresă. Acest lucru ar fi similar cu un "cont comun", așa cum este el

folosit în sistemul bancar tradițional unde oricare dintre soți poate cheltui doar cu o singură semnătură. Sau Gopesh, designer-ul web plătit de către Bob pentru a crea un site web, ar putea avea o adresă cu semnătură multiplă 2-din-3 pentru afacerea lui, lucru care l-ar asigura că fondurile nu pot fi cheltuite decât dacă cel puțin doi dintre partenerii de afaceri semnează o tranzacție.

Vom explora cum să creăm tranzacții care cheltuiesc fonduri din adrese P2SH (și semnătură multiplă) în [Tranzacții](#).

## Adrese de Vanitate

Adresele de vanitate sunt adrese bitcoin valide care conțin mesaje ușor de citit de către oameni. De exemplu, `1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33` este o adresă de vanitate care conține litere ce formează cuvântul "Love" pe primele patru poziții ale valorii Base58. Adresele de vanitate necesită generarea și testarea a miliarde de chei private candidat, până când o adresă cu forma dorită este găsită. Deși există unele optimizări în algoritmul de generare a adreselor de vanitate, procesul constă din alegerea unei chei private aleator, derivarea cheii publice, derivarea adresei, și verificarea dacă adresa găsită se potrivește cu forma dorită a adresei de vanitate, apoi se repetă acești pași de miliarde de ori până când se găsește o cheie care se potrivește.

Odată ce este găsită o adresă de vanitate care se potrivește cu modelul dorit, cheia privată din care a fost derivată poate fi folosită de către proprietar pentru a cheltui bitcoin în exact același mod ca orice altă adresă. Adresele de vanitate nu sunt în nici un fel mai sigure sau mai puțin sigure decât oricare altă adresă. Ele depind de aceeași Criptografie a Curbei Eliptice (ECC) și SHA ca oricare altă adresă. Nu putem găsi mai ușor cheia privată a unei adrese care începe cu un model de vanitate decât am putea pentru orice altă adresă.

În [Introducere](#), am prezentat-o pe Eugenia, directoarea unui centru de ajutorare a copiilor care operează în Filipine. Să presupunem că Eugenia organizează o strângere de fonduri și vrea să folosească o adresă de vanitate pentru a-și face publicitate. Eugenia va crea o adresă de vanitate care începe cu "1Kids" pentru a promova strângerea de fonduri pentru copii. Să vedem cum această adresă de vanitate va fi creată și ce înseamnă pentru securitatea colectei Eugeniei.

## Generarea adreselor de vanitate

Este important să înțelegem că o adresă bitcoin este doar un număr reprezentat prin simboluri din alfabetul Base58. Căutarea unui model de genul "1Kids" poate fi vazută drept căutarea unei adrese în intervalul începând de la `1Kids11111111111111111111111111111111` la `1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzzz`. Sunt aproximativ  $58^{29}$  (aproximativ  $1.4 * 10^{51}$ ) adrese în acest interval, toate începând cu "1Kids." [Intervalul adreselor de vanitate care încep cu "1Kids"](#) arată intervalul de adrese care au prefixul "1Kids".

Table 5. Intervalul adreselor de vanitate care încep cu "1Kids"

De la	<code>1Kids11111111111111111111111111111111</code>
	<code>1Kids11111111111111111111111111111112</code>
	<code>1Kids11111111111111111111111111111113</code>
	...

Să privim la modelul "1Kids" ca la un număr și să vedem cât de frecvent am putea găsi acest model într-o adresă bitcoin (vezi [Frecvența unui model de vanitate \(1KidsCharity\)](#) și [timpul mediu de căutare pe un calculator desktop](#)). Un calculator mediu, fără hardware specializat, poate căuta aproximativ 100.000 de chei pe secundă.

Table 6. Frecvența unui model de vanitate (1KidsCharity) și timpul mediu de căutare pe un calculator desktop

Lungime	Model	Frecvență	Timpul mediu de căutare
1	1K	1 din 58 chei	< 1 milisecunde
2	1Ki	1 din 3.364	50 milisecunde
3	1Kid	1 în 195.000	< 2 secunde
4	1Kids	1 din 11 milioane	1 minut
5	1KidsC	1 din 656 milioane	1 oră
6	1KidsCh	1 din 38 miliarde	2 zile
7	1KidsCha	1 în 2,2 trilioane	3–4 luni
8	1KidsChar	1 în 128 trilioane	13-18 ani
9	1KidsChari	1 în 7 cvadrilioane	800 de ani
10	1KidsCharit	1 din 400 cvadrilioane	46.000 de ani
11	1KidsCharity	1 din 23 de quintillioane	2,5 milioane de ani

După cum se pare, Eugenia nu va crea prea curând adresa de vanitate "1KidsCharity", chiar dacă ar avea acces la câteva mii de calculatoare. Fiecare caracter în plus mărește dificultatea cu un factor de 58. Modele cu mai mult de șapte caractere sunt de obicei găsite de hardware specializat, cum ar fi un calculator asamblat special cu mai multe plăci GPU. Acestea sunt adesea echipamente de minerit care nu mai sunt rentabile pentru minerit bitcoin, dar pot fi folosite pentru a găsi adrese de vanitate. Căutarea de adrese de vanitate folosind sisteme cu GPU este de câteva ordine de mărime mai rapidă decât pe un sistem obișnuit care folosește CPU.

Un alt mod de a găsi o adresă de vanitate este să externalizați munca unui bazin de mineri de vanitate, cum ar fi bazinul [Vanity Pool](#). Un bazin este un serviciu care le permite celor cu hardware GPU să câștige bitcoin căutând adrese de vanitate pentru alții. Pentru o mică plată (0,01 bitcoin sau aproximativ 5\$ la momentul scrierii), Eugenia poate externaliza căutarea unui model de adresă cu șapte caractere și să obțină un rezultat în câteva ore, în loc să trebuiască să ruleze o căutare CPU pentru câteva luni.

Generarea unei adrese de vanitate este un exercițiu de forță brută: încercați o cheie aleator, verificați dacă adresa rezultată se potrivește cu modelul dorit, repetați până când ați reușit. [Miner de adresă de vanitate](#) arată un exemplu de "miner de vanitate," un program conceput să găsească adrese de vanitate, scris în C++. Exemplul folosește biblioteca libbitcoin, pe care am introdus-o în [Clienti, Biblioteci și Instrumente Alternative](#).

*Example 16. Miner de adresă de vanitate*

```
#include <random>
#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_base16(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() & 255;
```

```

    // Return result.
    return secret;
}

std::string bitcoin_address(const bc::ec_secret& secret)
{
    // Convert secret to payment address
    bc::wallet::ec_private private_key(secret);
    bc::wallet::payment_address payaddr(private_key);
    // 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;
}

```

#### NOTE

Compilarea și rularea exemplului `vanity-miner` folosește `std::random_device`. Depinzând de implementare, ar putea reflecta o CSRNG furnizată de sistemul de operare. În cazul unui sistem de operare bazat pe Unix, cum ar fi Linux, se bazează pe `/dev/urandom`. Generatorul de numere aleatoare folosit aici este folosit doar în scopuri demonstrative și nu este potrivit pentru generarea de chei bitcoin de calitate din moment ce nu este destul de sigur.

Codul exemplu trebuie să fie compilat folosind un compilator C++ și legat de biblioteca `libbitcoin` (care trebuie instalată mai întâi pe acel sistem). Pentru a rula exemplul, rulați executabilul `vanity-miner` fără parametri (vezi [Compilarea și rularea exemplului `vanity-miner`](#)), iar acesta va încerca să găsească o adresă de vanitate care începe cu "1kid."

### Example 17. Compilarea și rularea exemplului vanity-miner

```
$ # Compile the code with g++  
$ g++ -o vanity-miner vanity-miner.cpp $(pkg-config --cflags --libs libbitcoin)  
$ # Run the example  
$ ./vanity-miner  
Found vanity address! 1KiDzkG4MxmovZryZRj8tK81oQRhbZ46YT  
Secret: 57cc268a05f83a23ac9d930bc8565bac4e277055f4794cbd1a39e5e71c038f3f  
$ # Run it again for a different result  
$ ./vanity-miner  
Found vanity address! 1Kidxr3wsmMzzouwXibKfwTYs5Pau8TUFn  
Secret: 7f65bbbbe6d8caae74a0c6a0d2d7b5c6663d71b60337299a1a2cf34c04b2a623  
# Use "time" to see how long it takes to find a result  
$ time ./vanity-miner  
Found vanity address! 1KidPWhKgGRQWD5PP5TAnGfDyfWp5yceXM  
Secret: 2a802e7a53d8aa237cd059377b616d2bfcfa4b0140bc85fa008f2d3d4b225349  
  
real    0m8.868s  
user    0m8.828s  
sys     0m0.035s
```

Codul exemplu va dura câteva secunde pentru a găsi o potrivire pentru modelul de trei caractere "kid," după cum putem vedea când folosim comanda Unix *time* pentru a măsura timpul de execuție. Schimbați modelul *search* în codul sursă și observați cât durează pentru un model de patru sau cinci caractere.

### Securitatea adresei de vanitate

Adresele de vanitate pot fi folosite pentru a îmbunătăți și pentru a zădărnici măsurile de siguranță; ele sunt cu adevărat o sabie cu două tăișuri. Folosîtă să îmbunătățească securitatea, o adresă deosebită face mai dificil pentru adversari să o înlocuiască cu adresele lor proprii și să păcăleacă clienții dumneavoastră în a-i plăti pe ei în locul dumneavoastră. Din păcate, adresele de vanitate permit de asemenea oricui să creeze o adresă care seamănă cu orice altă adresă, sau chiar cu o altă adresă de vanitate, prin urmare înselând clienții dumneavoastră.

Eugenia ar putea publica o adresă generată aleatoriu (de exemplu, 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy) către care oamenii își pot trimite donațiile. Sau ar putea genera o adresă de vanitate care începe cu 1Kids, pentru a o face mai distinctivă.

În ambele cazuri, unul dintre riscurile utilizării unei singure adrese fixe (în loc de o adresă dinamică separată pentru fiecare donator) este că un hoț ar putea să se infiltreze pe site-ul dumneavoastră web și să înlocuiască adresa cu propria sa adresă, redirectând astfel donațiile către el însuși. Dacă ați publicat adresa dumneavoastră de donare în mai multe locuri diferite, utilizatorii dumneavoastră pot inspecta vizual adresa înainte de a efectua o plată pentru a se asigura că este aceeași adresă pe care au văzut-o pe site-ul dumneavoastră web, pe e-mail și pe pliantul dumneavoastră. În cazul unei adrese aleatorii de forma 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy, utilizatorul obișnuit va inspecta probabil primele caractere "1J7mdg" și va fi mulțumit că adresa corespunde. Folosind un generator de adrese de

vanitate, cineva care are intenția de a fura prin înlocuirea cu o adresă similară poate genera rapid adrese care se potrivesc cu primele câteva caractere, aşa cum se vede în [Generarea adreselor de vanitate pentru a se potrivi cu o adresă aleatorie](#).

Table 7. Generarea adreselor de vanitate pentru a se potrivi cu o adresă aleatorie

Adresa aleatorie originală	1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
Vanitate (potrivire de 4 caractere)	1J7md1QqU4LpctBetHS2ZoyLV5d6dShhEy
Vanitate (potrivire de 5 caractere)	1J7mdgYqyNd4ya3UEcq31Q7sqRMXw2XZ6n
Vanitate (potrivire de 6 caractere)	1J7mdg5WxGENmwyJP9xuGhG5KRzu99BBCX

Deci, o adresă de vanitate crește securitatea? Dacă Eugenia generează adresa de vanitate *1Kids33q44erFfpeXrmDSz7zEqG2FesZEN*, utilizatorii probabil vor acorda atenție modelului de vanitate și câtorva caractere după, de exemplu vor observa partea "1Kids33" din adresă. Acest lucru ar obliga un atacator să genereze o adresă de vanitate care să se potrivească cu cel puțin șase caractere (încă două), investind un efort care este de 3364 (58 x 58) mai mare decât efortul pe care Eugenia l-a investit pentru cele 4 caractere de vanitate. În esență, efortul pe care Eugenia îl depune (sau pentru care plătește un bazin de vanitate) "împinge" atacatorul să producă o adresă de vanitate cu un model mai lung. Dacă Eugenia plătește un bazin pentru a genera o adresă de vanitate cu un model de 8 caractere, atacatorul va fi împins în zona de 10 caractere, care nu este fezibil pe un calculator personal și este scump chiar cu un echipament de minerit specializat sau folosind un bazin de mineri de vanitate. Ce este accesibil pentru Eugenia, devine inaccesibil pentru atacator, mai ales dacă recompensa potențială în urma fraudei nu este destul de mare pentru a acoperi costurile generării adresei de vanitate.

## Portofele de Hârtie

Portofelele de hârtie sunt chei private bitcoin tipărite pe hârtie. Adesea portofelul de hârtie include și adresa bitcoin corespunzătoare pentru comoditate, dar acest lucru nu este necesar deoarece adresa poate fi derivată din cheia privată. Portofele de hârtie sunt o modalitate foarte eficientă pentru a crea copii de rezervă sau pentru stocare bitcoin offline, cunoscută și ca "stocare la rece." Ca mecanism de rezervă, un portofel de hârtie poate oferi securitate împotriva pierderii cheii private datorită unor probleme cu calculatorul, cum ar fi un hard-disk defect, furt, sau ștergere accidentală. Ca mecanism de "stocare la rece", dacă cheile portofelului de hârtie sunt generate offline și nu au fost stocate niciodată pe un calculator, atunci ele sunt mult mai în siguranță față de un hacker, un logger de taste, și orice alte amenințări online.

Portofelele de hârtie vin în multe forme, dimensiuni și modele, dar la un nivel foarte de bază sunt doar o cheie și o adresă tipărite pe hârtie. [Cea mai simplă formă a unui portofel de hârtie - o imprimare a adresei bitcoin și a cheii private](#) prezintă cea mai simplă formă de portofel de hârtie.

Table 8. Cea mai simplă formă a unui portofel de hârtie - o imprimare a adresei bitcoin și a cheii private

Adresa publică	Cheie privată (WIF)
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2Jpb nkeyhfsYB1Jcn

Portofelele de hârtie pot fi generate ușor folosind un instrument de tipul generatorului client-side

JavaScript de la [bitaddress.org](http://bitaddress.org). Această pagină conține codul necesar pentru a genera chei și portofele de hârtie, chiar atunci când este complet deconectată de la internet. Pentru a o folosi, salvați pagina HTML pe disc sau pe un stick USB extern. Deconectați-vă de la internet și deschideți fișerul într-un browser. Chiar mai bine, porniți calculatorul folosind un sistem de operare curat, cum ar fi un Linux bootabil de pe CD-ROM. Orice chei generate cu acest instrument în timp ce sunteți offline, pot fi printate la o imprimantă conectată prin cablu (nu wireless), prin urmare creând portofele de hârtie a căror chei există doar pe hârtie și nu au fost niciodată stocate pe un sistem online. Puneți aceste portofele într-un loc ferit de foc și "trimiteți" bitcoin la adresele lor, pentru a implementa o soluție de "stocare la rece" simplă dar foarte eficientă. [Un exemplu de portofel simplu de hârtie de la bitaddress.org](#) prezintă un portofel de hârtie generat cu ajutorul site-ului bitaddress.org.



Figure 21. Un exemplu de portofel simplu de hârtie de la bitaddress.org

Dezavantajul unui portofel de hârtie simplu este că cheile tipărite sunt vulnerabile la furt. Un hoț care reușește să obțină acces la hârtie o poate fura sau poate fotografia cheile obținând controlul asupra fondurilor blocate cu acele chei. Un sistem mai sofisticat de stocare a portofelului de hârtie folosește chei private criptate BIP-38. Cheile tipărite pe hârtie sunt protejate de o frază-de-access pe care proprietarul a memorat-o. Fără fraza-de-access, cheile criptate nu sunt de nici un folos. Cu toate acestea, ele sunt superioare unui portofel protejat cu frază-de-access pentru că aceste chei nu au fost niciodată online și trebuie să fie preluate fizic dintr-un seif sau din altă locație fizică sigură. [Un exemplu de portofel de hârtie criptat de la bitaddress.org. Parola este "test".](#) arată un portofel de hârtie cu o cheie privată criptată (BIP-38) creat pe site-ul bitaddress.org.



Figure 22. Un exemplu de portofel de hârtie criptat de la bitaddress.org. Parola este "test".

#### WARNING

Chiar dacă puteți depozita fonduri într-un portofel de hârtie de mai multe ori, ar trebui să retrageți fondurile doar o dată, cheltuind totul. Motivul este că în procesul de deblocare și cheltuire a fondurilor unele portofele s-ar putea să genereze o adresă de rest dacă cheltuiți mai puțin decât suma totală. Dacă calculatorul pe care l-ați folosit să semnați tranzacțiile este apoi compromis, riscați expunerea cheii private, oferind access la fondurile din adresa de rest. Cheltuind întreaga sumă a unui portofel de hârtie doar o dată, reduceți riscul de a compromite cheile. Dacă trebuie să cheltuiți doar o sumă mai mică, trimiteți fondurile rămase la un nou portofel de hârtie în aceeași tranzacție.

Portofelele de hârtie au mai multe modele și dimensiuni, cu multe funcționalități diferite. Unele sunt destinate a fi oferite ca și cadouri și au teme de sezon, cum ar fi teme de Crăciun și Anul Nou. Altele sunt concepute pentru depozitare într-un seif al băncii având cheile private ascunse cumva, sau cu colante opace răzuibile, sau împăturite și sigilate cu o folie adezivă. Imaginele următoare prezintă diferite exemple de portofele de hârtie având funcții de securitate și rezervă.

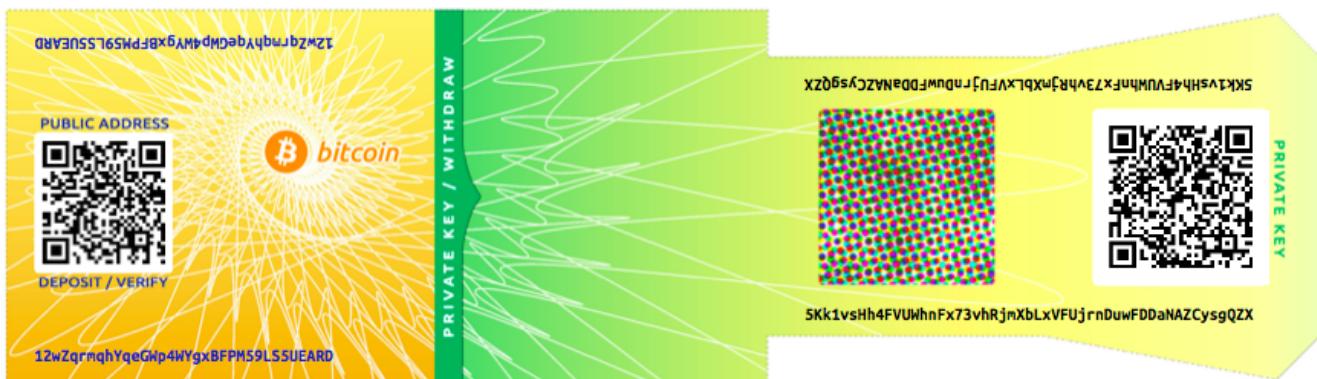


Figure 23. Un exemplu de portofel de hârtie de la bitcoinpaperwallet.com cu cheia privată pe o clapetă pliabilă



Figure 24. Portofelul de hârtie bitcoinpaperwallet.com cu cheia privată ascunsă

Alte modele oferă copii suplimentare ale cheii și adresei, sub formă de cioturi detașabile similare cu cioturile de bilete, permitându-vă să stocați mai multe exemplare pentru a vă proteja împotriva incendiilor, inundațiilor sau a altor dezastre naturale.



Figure 25. Un exemplu de portofel de hârtie având copii suplimentare ale cheilor pe un "ciot" de rezervă.

## Portofele

Cuvântul portofel este folosit la descrierea câtorva lucruri diferite în bitcoin.

La un nivel înalt, un portofel este o aplicație care are rolul de interfață cu utilizatorul. Portofelul controlează accesul la banii utilizatorului, gestionează cheile și adresele, urmărește soldul și crează și semnează tranzacții.

Mai restrâns, din perspectiva programatorului, cuvântul "portofel" se referă la structura de date utilizată pentru stocarea și gestionarea cheilor unui utilizator.

În acest capitol vom analiza al doilea sens, în care portofelele sunt containere pentru chei private, de obicei implementate ca fișiere structurate sau baze de date simple.

## Prezentare Generală a Tehnologiei Portofelului

În această secțiune vom rezuma diferențele tehnologii utilizate pentru a construi portofele bitcoin ușor de utilizat, sigure și flexibile.

O concepție greșită comună despre bitcoin este aceea că portofelele bitcoin conțin bitcoin. De fapt, portofelul conține doar chei. "Monedele" sunt înregistrate în lanțul-de-blocuri în rețeaua bitcoin. Utilizatorii controlează monedele din rețea prin semnarea tranzacțiilor cu cheile din portofele. Într-un sens, un portofel bitcoin este un *breloc*.

**TIP**

Portofelele Bitcoin conțin chei, nu monede. Fiecare utilizator are un portofel care conține chei. Portofelele sunt mai degrabă brelocuri care conțin perechi de chei private/publice (vezi [Cheile Private și Publice](#)). Utilizatorii semnează tranzacții cu cheile, dovedind astfel că dețin ieșirile tranzacției (monedele lor). Monedele sunt stocate în lanțul-de-blocuri sub formă de ieșiri de tranzacție (adesea notate ca *vout* sau *txout*).

Există două tipuri principale de portofele, deosebite prin faptul că cheile pe care le conțin sunt legate între ele sau nu.

Primul tip este un *portofel nedeterminist*, unde fiecare cheie este generată independent dintr-un număr aleatoriu. Cheile nu au legătură între ele. Acest tip de portofel este cunoscut și sub denumirea de portofel JBOK datorită sintagmei "Just a Bunch Of Keys" (Doar o grămadă de chei).

Cel de-al doilea tip de portofel este un *portofel determinist*, unde toate cheile sunt derive dintr-o singură cheie principală (master key), cunoscută sub numele de *sămânță* (seed). Toate cheile din acest tip de portofel sunt legate între ele și pot fi generate din nou dacă cineva are sămânța originală. Există o serie de metode diferite de *derivare a cheilor* utilizate în portofelele deterministe. Cea mai frecventă metodă de derivare folosește o structură de tip arbore și este cunoscută sub numele de portofel *determinist ierarhic* sau portofel *HD* (hierarchical deterministic).

Portofelele deterministe sunt inițializate dintr-o sămânță (seed). Pentru a le face mai ușor de utilizat, semințele sunt codate ca și cuvinte în limba engleză, cunoscute și sub denumirea de "cuvinte cod mnemonic".

Următoarele secțiuni prezintă fiecare dintre aceste tehnologii la un nivel înalt.

## Portofele Nedeterministe (Aleatoare)

În primul portofel bitcoin (acum numit Bitcoin Core), portofelele erau colecții de chei private generate aleatoriu. De exemplu, clientul inițial Bitcoin Core pre-generează 100 de chei private aleatorii la prima pornire și generează mai multe chei după cum este necesar, folosind fiecare cheie o singură dată. Astfel de portofele sunt înlocuite cu portofele deterministe, deoarece sunt greoi de gestionat, de creat copii de rezervă și de importat. Dezavantajul cheilor aleatorii este că, dacă generați multe, trebuie să păstrați copii ale tuturor, ceea ce înseamnă că portofelul trebuie să fie copiat în mod frecvent. Fiecare cheie trebuie să fie salvată sau fondurile pe care le controlează sunt pierdute irevocabil dacă portofelul devine inaccesibil. Acest lucru este în conflict direct cu principiul evitării reutilizării adreselor, utilizând fiecare adresă bitcoin pentru o singură tranzacție. Reutilizarea adreselor reduce confidențialitatea prin asocierea mai multor tranzacții și adrese între ele. Un portofel nedeterministic de Tip-0 este o alegere slabă, în special dacă doriți să evitați refolosirea adreselor, deoarece asta înseamnă gestionarea multor chei, ceea ce aduce cu sine necesitatea creării frecvente a copiilor de rezervă. Deși clientul Bitcoin Core include un portofel de Tip-0, utilizarea acestui portofel este descurajată de către dezvoltatorii Bitcoin Core. [Portofel nedeterminist de Tip-0 \(aleator\)](#): o colecție de chei generate aleatoriu prezintă un portofel

nedeterminist, care conține o colecție de chei aleatorii independente.

**TIP**

Utilizarea portofelelor nedeterministe este descurajată pentru orice altceva decât teste simple. Ele sunt pur și simplu prea greoale pentru a face copii de rezervă și a le utiliza. În schimb, utilizați un portofel bazat-pe-standardele-industriei cu o sămânță (seed) mnemonică ca rezervă.

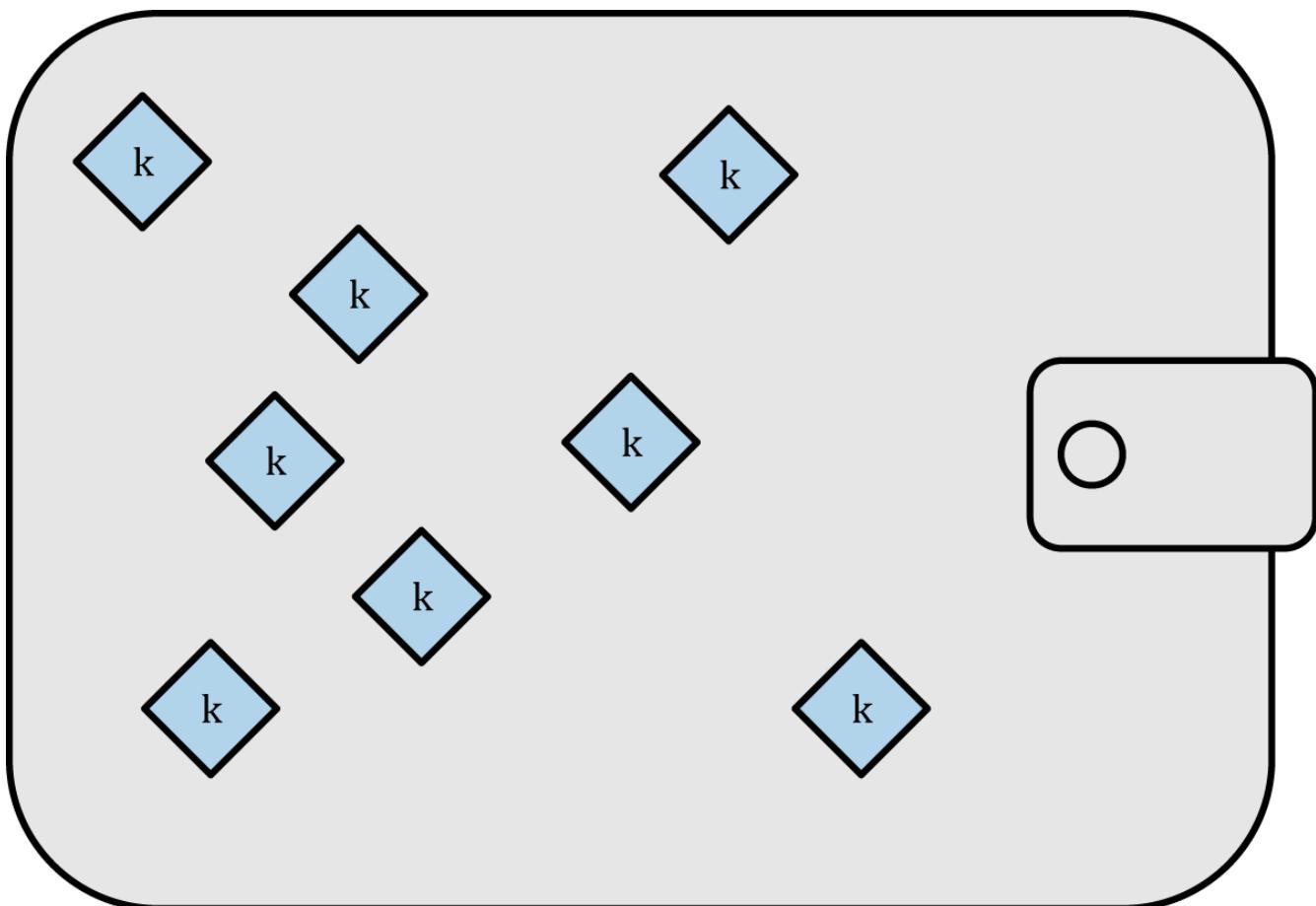


Figure 26. Portofel neteterminist de Tip-0 (aleator): o colecție de chei generate aleatoriu

## Portofele Deterministe (Cu Sămânță)

Portofelele deterministe sau cu "sămânță" (seed) sunt portofele care conțin chei private derivate dintr-o sămânță comună, prin utilizarea unei funcții rezumat (hash) unidirectionale. Sămânță este un număr generat la întâmplare combinat cu alte date, precum un număr de indice sau "cod de lanț" (vezi [Portofele HD \(BIP-32 / BIP-44\)](#)) pentru a obține cheile private. Într-un portofel determinist, sămânță este suficientă pentru a recupera toate cheile derivate și, prin urmare, o copie de rezervă la momentul creării este suficientă. Sămânță este de asemenea suficientă pentru un export sau import de portofel, permitând migrarea ușoară a tuturor cheilor utilizatorului între diferite implementări de portofel. [Portofel Tip-1 determinist \(cu sămânță\): o secvență deterministă de chei derivate dintr-o sămânță](#) prezintă o diagramă logică a unui portofel determinist.

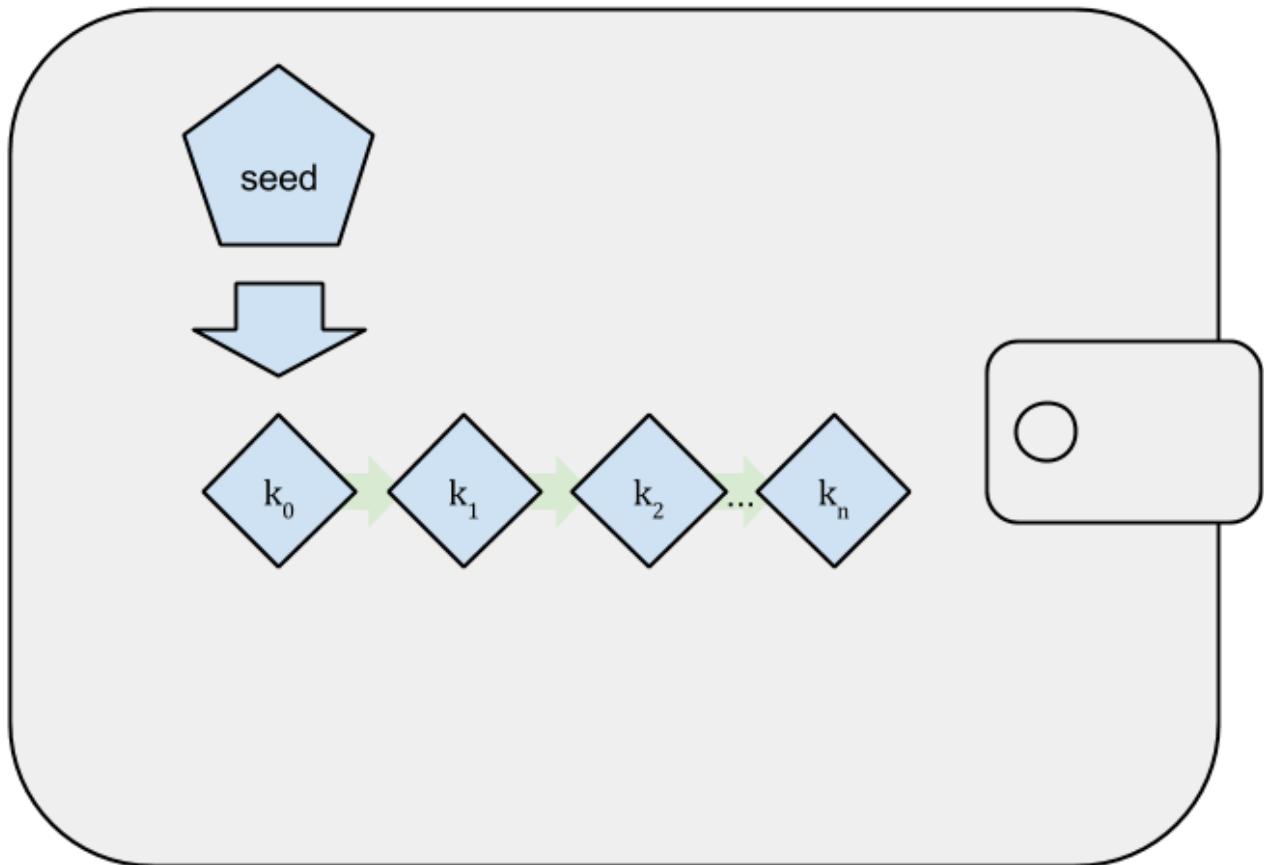


Figure 27. Portofel Tip-1 determinist (cu sămânță): o secvență deterministă de chei derivate dintr-o sămânță

### Portofele HD (BIP-32 / BIP-44)

Portofelele deterministe au fost dezvoltate pentru a facilita derivarea mai multor chei dintr-o singură "sămânță" (seed). Cea mai avansată formă de portofele deterministe este portofelul HD (Hierarchical Deterministic) definit de standardul BIP-32. Portofelele HD conțin chei derivate într-o structură de tip arbore, astfel încât o cheie părinte poate deriva o secvență de chei copii, fiecare dintre ele putând deriva o secvență de chei nepoți și aşa mai departe, la o adâncime infinită. Această structură de arbore este ilustrată în [Portofel HD Tip-2: un arbore de chei generate dintr-o singură sămânță](#).

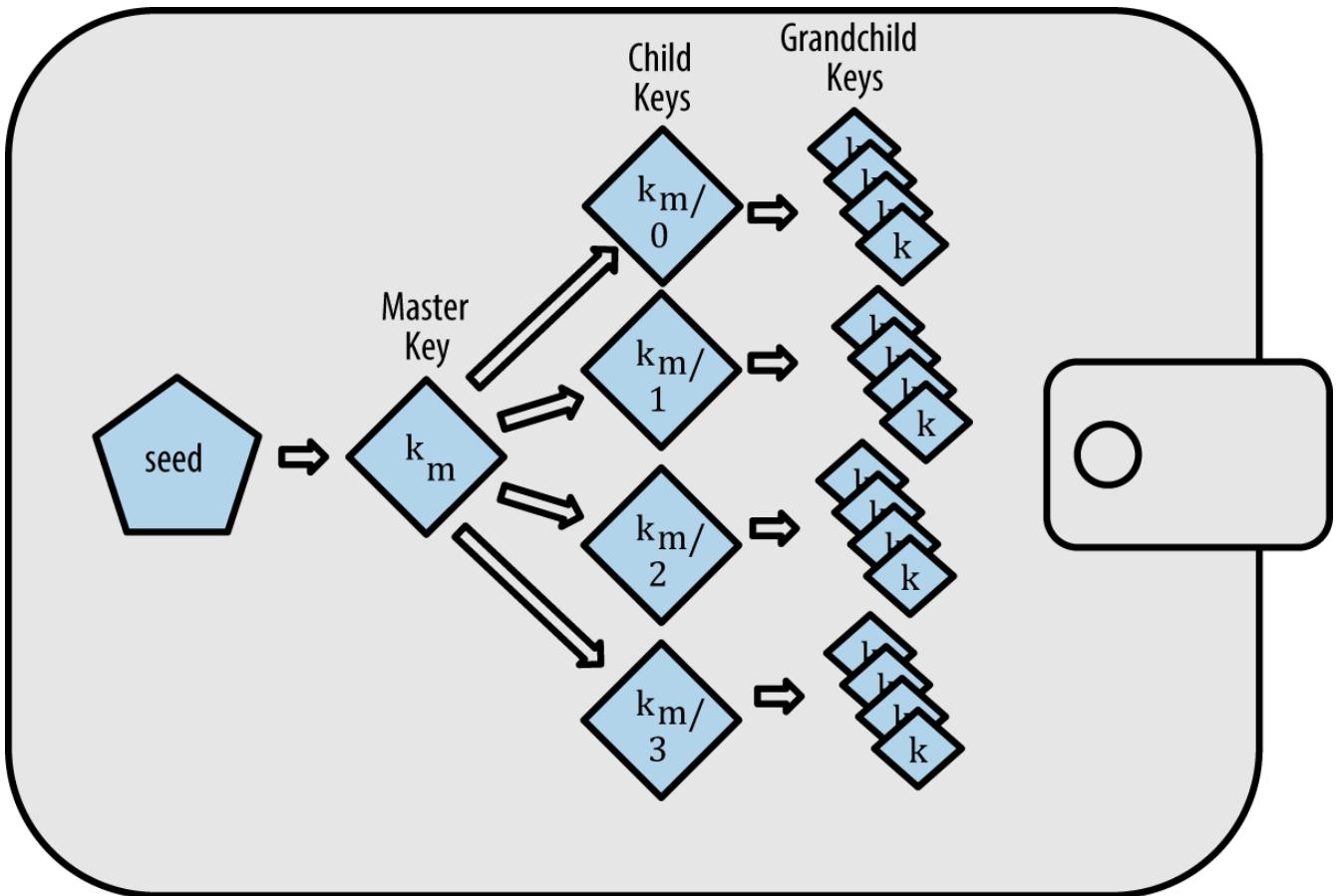


Figure 28. Portofel HD Tip-2: un arbore de chei generate dintr-o singură sămânță

Portofelele HD oferă două avantaje majore față de cheile aleatorii (nedeterministe). În primul rând, structura de arbore poate fi utilizată pentru a exprima o semnificație organizatorică suplimentară, cum ar fi atunci când o ramură specifică de sub-chei este folosită pentru a primi plăți și o ramură diferită este folosită pentru a primi restul la plățile efectuate. Ramurile de chei pot fi de asemenea utilizate în setări corporative, alocând ramuri diferite departamentelor, filialelor, funcțiilor specifice sau categoriilor de contabilitate.

Al doilea avantaj al portofelelor HD este că utilizatorii pot crea o secvență de chei publice fără a avea acces la cheile private corespunzătoare. Acest lucru permite portofelelor HD să fie utilizate pe un server nesigur sau să fie folosite doar pentru primire de fonduri, emițând o cheie publică diferită pentru fiecare tranzacție. Cheile publice nu trebuie să fie pre-încărcate sau derivate în avans, însă serverul nu are cheile private care pot cheltui fondurile.

## Semințe și Coduri Mnemonice (BIP-39)

Portofelele HD sunt un mecanism foarte puternic pentru a gestiona multe chei și adrese. Ele sunt chiar mai utile dacă sunt combinate cu un mod standardizat de a crea semințe dintr-o secvență de cuvinte în limba engleză ușor de transcris, exportat și importat în portofele. Aceasta este cunoscută sub numele de *mnemonică*, iar standardul este definit de BIP-39. Astăzi, majoritatea portofelelor bitcoin (precum și portofelele pentru alte criptomonede) folosesc acest standard și pot importa și exporta semințe pentru copii de rezervă și pentru recuperare folosind mnemonice interoperabile.

Să privim acest lucru dintr-o perspectivă practică. Care dintre următoarele semințe este mai ușor de transcris, înregistrat pe hârtie, citit fără greșală, exportat și importat într-un alt portofel?

O sămânță pentru un portofel determinist, în hexa

0C1E24E5917779D297E14D45F14E1A1A

O sămânță pentru un portofel determinist, dintr-o mnemonică de 12 cuvinte

army van defense carry jealous true  
garbage claim echo media make crunch

## Cele mai Bune Practici pentru Portofele

Pe măsură ce tehnologia portofelului bitcoin s-a maturizat, au apărut anumite standarde ale industriei care fac ca portofelele bitcoin să fie interoperabile, ușor de utilizat, sigure și flexibile. Aceste standarde sunt:

- Cuvinte cod mnemonic, bazate pe BIP-39
- Portofele HD, bazate pe BIP-32
- Structura portofel HD multifuncțională, bazată pe BIP-43
- Portofele multivalută și multicont, bazate pe BIP-44

Acste standarde s-ar putea schimba sau pot deveni învechite prin evoluțiile viitoare, dar deocamdată formează un set de tehnologii cuplăte care au devenit standardul de facto pentru portofelul bitcoin.

Standardele au fost adoptate de o gamă largă de portofele software și hardware bitcoin, ceea ce face ca aceste portofele să fie interoperabile. Un utilizator poate exporta o mnemonică generată pe unul dintre aceste portofele și să o importe într-un alt portofel, recuperând toate tranzacțiile, cheile și adresele.

Un exemplu de portofele software care acceptă aceste standarde include (listate alfabetic) Breadwallet, Copay, Multibit HD și Mycelium. Exemple de portofele hardware care acceptă aceste standarde includ (enumerate alfabetic) Keepkey, Ledger și Trezor.

Secțiunile următoare examinează fiecare dintre aceste tehnologii în detaliu.

**TIP**

Dacă implementați un portofel bitcoin, acesta ar trebui să fie construit ca un portofel HD, cu o sămânță (seed) codificată ca și cod mnemonic pentru recuperare, urmând standardele BIP-32, BIP-39, BIP-43 și BIP-44, așa cum este descris în următoarele secțiuni.

## Utilizarea unui Portofel Bitcoin

În [Utilizările Bitcoin, Utilizatorii, și Poveștile Lor](#) l-am introdus pe Gabriel, un Tânăr întreprinzător din Rio de Janeiro, care are un magazin web simplu care vinde tricouri, căni de cafea și autocolante cu brandul bitcoin.

Gabriel folosește un portofel bitcoin hardware Trezor ([Un dispozitiv Trezor: un portofel bitcoin HD](#)

în hardware) pentru a-și gestiona în siguranță bitcoin-ul. Trezor este un dispozitiv USB simplu cu două butoane care stochează cheile (sub forma unui portofel HD) și semnează tranzacții. Portofelele Trezor implementează toate standardele industriei discutate în acest capitol, astfel încât Gabriel nu se bazează pe nicio tehnologie proprie sau soluție de la un singur furnizor.



Figure 29. Un dispozitiv Trezor: un portofel bitcoin HD în hardware

Când Gabriel a folosit Trezor pentru prima dată, dispozitivul a generat o mnemonică și o sămânță (seed) dintr-un generator de numere aleatoriu încorporat în hardware. În această fază de initializare, portofelul a afișat pe ecran o secvență numerotată de cuvinte, unul câte unul (vezi [Trezor afișând unul dintre cuvintele mnemonicice](#)).

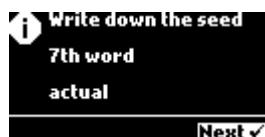


Figure 30. Trezor afișând unul dintre cuvintele mnemonicice

Notând acestă mnemonică, Gabriel a creat o copie de rezervă (vezi [Copia de rezervă de hârtie a lui Gabriel](#)) care poate fi folosită pentru recuperare în caz de pierdere sau deteriorare a dispozitivului Trezor. Această mnemonică poate fi utilizată pentru recuperarea unui nou Trezor sau în oricare dintre numeroasele portofele software sau hardware compatibile. Rețineți că succesiunea cuvintelor este importantă, astfel încât copiile mnemonicice de rezervă pe hârtie au spații numerotate pentru fiecare cuvânt. Gabriel trebuie să înregistreze cu atenție fiecare cuvânt în spațiul numerotat pentru a păstra secvența corectă.

Table 9. Copia de rezervă de hârtie a lui Gabriel

1.	army	7.	garbage
2.	van	8.	claim
3.	defense	9.	echo
4.	carry	10.	media
5.	jealous	11.	make
6.	true	12.	crunch

**NOTE**

O mnemonică de 12 cuvinte este afișată în [Copia de rezervă de hârtie a lui Gabriel](#), pentru simplitate. De fapt, majoritatea portofelelor hardware generează o mnemonică mai sigură fomată din 24 de cuvinte. Mnemonică este utilizată în acela și mod, indiferent de lungime.

Pentru prima implementare a magazinului său web, Gabriel folosește o singură adresă bitcoin, generată pe dispozitivul său Trezor. Această adresă unică este utilizată de către toți clienții pentru toate comenzi. După cum vom vedea, această abordare are unele dezavantaje și poate fi îmbunătățită cu un portofel HD.

## Detalii despre Tehnologia Portofelului

Să examinăm acum în detaliu fiecare dintre standardele importante ale industriei, care sunt utilizate de multe portofele bitcoin.

### Cuvinte Cod Mnemonic (BIP-39)

Cuvintele cod mnemonic sunt secvențe de cuvinte care reprezintă (codifică) un număr aleator folosit ca sămânță (seed) pentru a obține un portofel determinist. Secvența de cuvinte este suficientă pentru a recrea sămânța și de acolo pentru a recrea portofelul și toate cheile derivate. O aplicație portofel care implementează portofele deterministe cu cuvinte mnemonicice va arăta utilizatorului o secvență de 12 până la 24 de cuvinte atunci când acesta crează un portofel. Acea secvență de cuvinte este copia de rezervă a portofelului și poate fi folosită pentru a recupera și recrea toate cheile din aceași aplicație sau din alte aplicații portofel compatibile. Cuvintele mnemonicice facilitează crearea copiilor de rezervă pentru că sunt ușor de citit și de transcris corect, în comparație cu o secvență aleatoare de numere.

**TIP**

Cuvintele mnemonicice sunt deseori confundate cu "portofelele mentale". Nu sunt același lucru. Diferența principală este că un set de portofele mentale este format din cuvinte alese de utilizator, în timp ce cuvintele mnemonicice sunt create la întâmplare de portofel și prezentate utilizatorului. Această diferență importantă face ca cuvintele mnemonicice să fie mult mai sigure, deoarece oamenii sunt surse foarte slabe de aleatoriu.

Codurile mnemonicice sunt definite în BIP-39 (vezi [Propuneri de Îmbunătățire Bitcoin](#)). Rețineți că BIP-39 este o implementare a unui standard de cod mnemonic. Există un standard diferit, cu un set diferit de cuvinte, folosit de portofelul Electrum și care precedă BIP-39. BIP-39 a fost propus de compania din spatele portofelului hardware Trezor și este incompatibil cu implementarea Electrum. Cu toate acestea, BIP-39 a obținut acum un sprijin larg al industriei în zeci de implementări interoperabile și ar trebui să fie considerat standardul de facto al industriei.

BIP-39 definește crearea unui cod și a unei semințe mnemonicice, pe care le descriem aici în nouă etape. Pentru claritate, procesul este împărțit în două părți: etapele 1 până la 6 sunt prezentate în [Generarea de cuvinte mnemonicice](#) și etapele 7 până la 9 sunt prezentate în [De la mnemonică la sămânță \(seed\)](#).

## Generarea de cuvinte mnemonice

Cuvintele mnemonice sunt generate automat de portofel folosind procesul standardizat definit în BIP-39. Portofelul pornește de la o sursă de entropie, adaugă o sumă de control (checksum) și apoi mapează entropia la lista de cuvinte:

1. Creați o secvență aleatorie (entropie) de 128 până la 256 biți.
2. Creați o sumă de control a secvenței aleatorii luând primii (lungimea-entropiei/32) biți ai rezumatului (hash-ului) său SHA256.
3. Adăugați suma de control la sfârșitul secvenței aleatorii.
4. Împărțiți rezultatul în segmente de lungime de 11 biți.
5. Mapați fiecare valoare de 11 biți către un cuvânt din dicționarul predefinit de 2048 de cuvinte.
6. Codul mnemonic este succesiunea de cuvinte.

[Generarea entropiei și codificarea sub formă de cuvinte mnemonice](#) arată modul în care se utilizează entropia pentru a genera cuvinte mnemonice.

## Mnemonic Words 128-bit entropy/12-word example

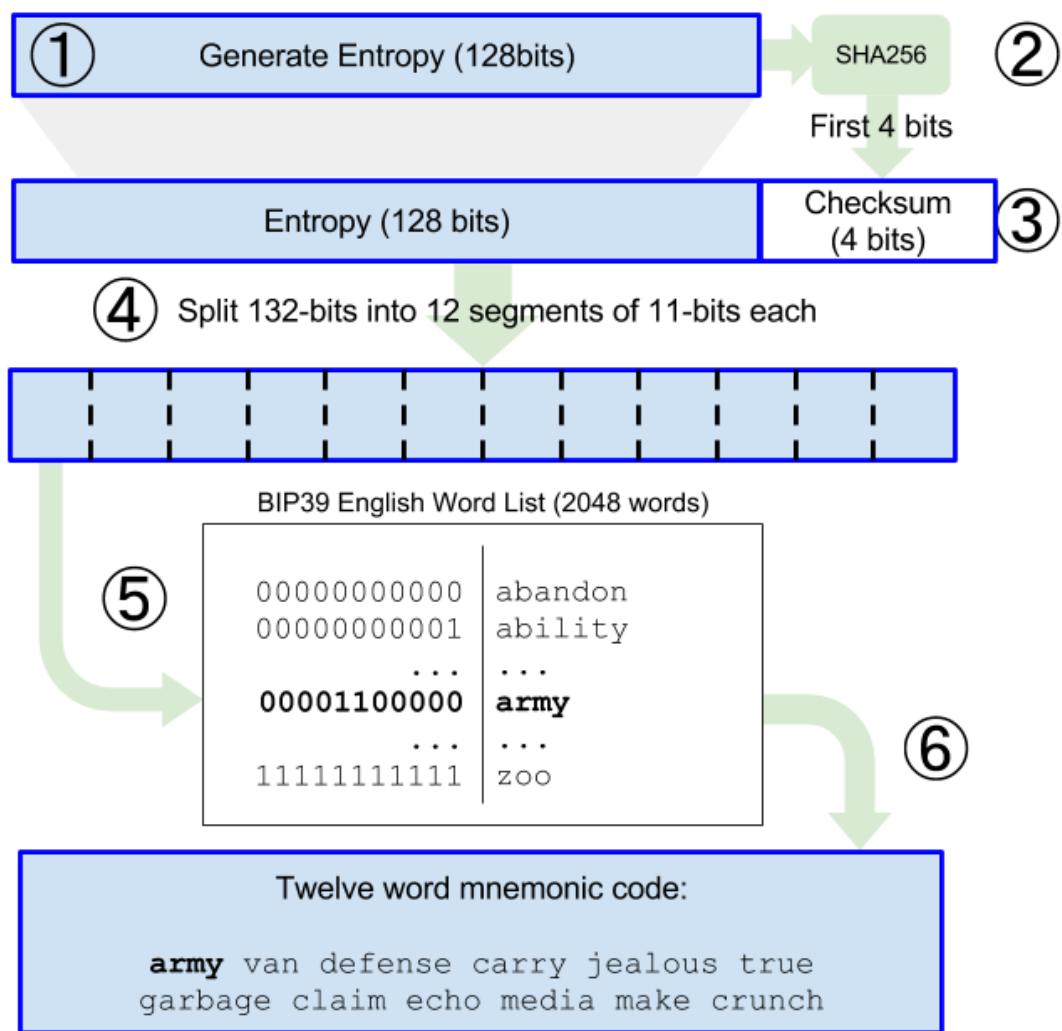


Figure 31. Generarea entropiei și codificarea sub formă de cuvinte mnemonice

**Coduri mnemonice: entropia și lungimea cuvintelor** arată relația dintre mărimea datelor de entropie și lungimea codurilor mnemonice formate din cuvinte.

Table 10. Coduri mnemonice: entropia și lungimea cuvintelor

Entropie (biți)	Sumă de control (biți)	Entropie + sumă de control (biți)	Lungime mnemonică (cuvinte)
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

## De la mnemonică la sămânță (seed)

Cuvintele mnemonicice reprezintă o entropie cu o lungime de 128 până la 256 biți. Entropia este apoi utilizată pentru a obține o sămânță mai lungă (512 biți) prin utilizarea funcției PBKDF2 de întindere (stretching) a cheilor. Sămânța produsă este apoi utilizată pentru a construi un portofel determinist și pentru a-i deduce cheile.

Funcția de întindere (stretching) a cheii are doi parametri: mnemonică și o *sare* (salt). Scopul unei sări într-o funcție de întindere a cheilor este de a face dificilă construirea unui tabel de căutare care să permită un atac prin forță brută. În standardul BIP-39, sareea are un alt scop - permite introducerea unei fraze care servește ca un factor suplimentar de securitate care protejează sămânța, așa cum vom descrie mai detaliat în [Fraza-de-acces optională în BIP-39](#).

Procesul descris în etapele 7 până la 9 continuă din procesul descris anterior în [Generarea de cuvinte mnemonicice](#):

7. Primul parametru al funcției PBKDF2 de întindere (stretching) a cheilor este *mnemonică* produsă la pasul 6.
8. Al doilea parametru pentru funcția PBKDF2 de întindere (stretching) a cheilor este o *sare*. Sarea (salt) este compusă din constanta **mnemonic** concatenată cu o frază-de-acces optională furnizată de utilizator.
9. PBKDF2 întinde parametrii (mnemonică și sareea) folosind 2048 de runde de rezumat cu algoritmul HMAC-SHA512, producând o valoare de 512 biți ca ieșire finală. Acea valoare de 512 biți este sămânța.

[De la mnemonică la sămânță](#) arată cum se folosește o mnemonică pentru a genera o sămânță.

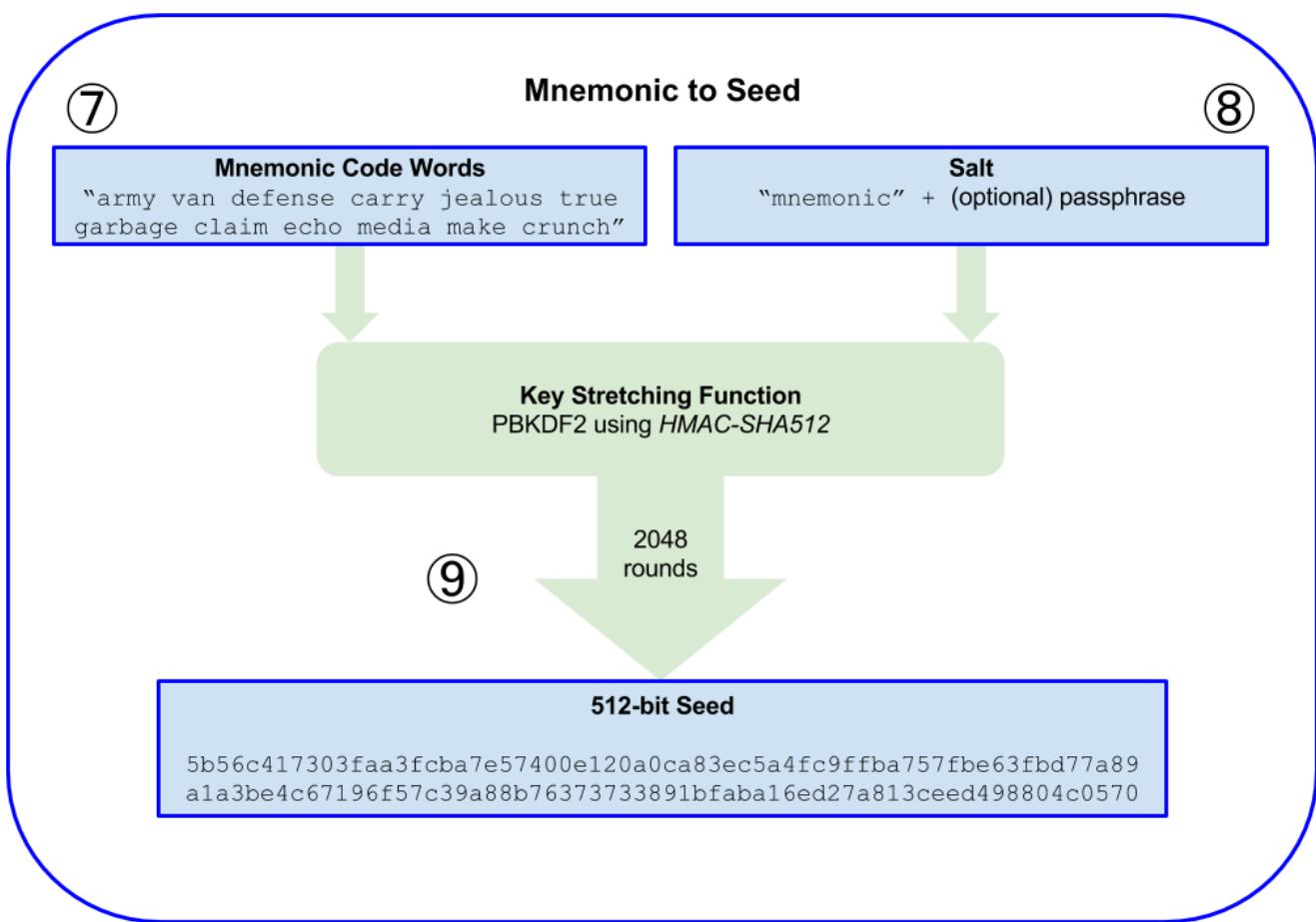


Figure 32. De la mnemonică la sămânță

**TIP** Funcția de întindere (stretching) a cheilor, cu runde de rezumări (hashing), este o protecție foarte eficientă împotriva atacurilor prin forță brută împotriva mnemonicei sau a frazei-de-acces. Este extrem de costisitor (ca efort de calcul) să încercați mai mult de câteva mii de fraze și combinații mnemonice, în timp ce numărul de semințe (seeds) derivate posibile este vast ( $2^{512}$ ).

Tabelele următoare arată câteva exemple de coduri mnemonice și semințele pe care le produc (cu sau fără o frază-de-acces).

Table 11. Cod mnemonic de entropie pe 128 biți, fără frază-de-acces, sămânța rezultată

Entropie (128 biți)	0c1e24e5917779d297e14d45f14e1a1a
Mnemonică (12 cuvinte)	army van defense carry jealous true garbage claim echo media make crunch
Frază-de-acces	(none)
Sămânță (512 biți)	5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fb77a89a1a3be4c67196f57c39a88b76373733891bfaba16ed27a813ceed498804c0570

Table 12. Cod mnemonic de entropie pe 128 biți, cu frază-de-acces, sămânța rezultată

Entropie (128 biți)	0c1e24e5917779d297e14d45f14e1a1a
---------------------	----------------------------------

Mnemonică (12 cuvinte)	army van defense carry jealous true garbage claim echo media make crunch
Frază-de-access	SuperDuperSecret
Sămânță (512 biți)	3b5df16df2157104cfdd22830162a5e170c0161653e 3afe6c88defeefb0818c793dbb28ab3ab091897d0 715861dc8a18358f80b79d49acf64142ae57037d1d 54

Table 13. Cod mnemonic de entropie pe 256 de biți, fără frază-de-acces, sămânța rezultată

Entropie (256 biți)	2041546864449caff939d32d574753fe684d3c947c3 346713dd8423e74abcf8c
Mnemonică (24 cuvinte)	cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige
Frază-de-access	(none)
Sămânță (512 biți)	3269bce2674acbd188d4f120072b13b088a0ecf87c6 e4cae41657a0bb78f5315b33b3a04356e53d062e5 5f1e0deaa082df8d487381379df848a6ad7e987984 04

### Fraza-de-access optională în BIP-39

Standardul BIP-39 permite utilizarea unei fraze-de-acces optionale în derivarea seminței (seed). Dacă nu se utilizează o frază-de-acces, mnemonică este întinsă (stretched) cu o sare (salt) constând din textul constant **mnemonic**, producând o sămânță specifică de 512 biți din orice mnemonică dată. Dacă se folosește o frază-de-acces, funcția de întindere produce o sămânță *diferită* din aceeași mnemonică. De fapt, având o singură mnemonică, fiecare frază-de-acces posibilă rezultă într-o sămânță *diferită*. În esență, nu există o frază-de-acces "greșită". Toate frazele-de-acces sunt valide și toate conduc la semințe *diferite*, formând un set vast de portofele posibile neinitializate. Setul de portofele posibile este atât de mare ( $2^{512}$ ) încât nu există nici o posibilitate practică de a folosi forță brută sau de a ghici accidental un portofel care este în uz.

**TIP**

Nu există fraze-de-acces "greșite" în BIP-39. Fiecare frază-de-acces conduce la un portofel, care, dacă nu a fost folosit anterior, va fi gol.

Fraza-de-access optională creează două caracteristici importante:

- Un al doilea factor (ceva memorat) care face ca o mnemonică să fie inutilă de la sine, protejând copile de rezervă mnemonice împotriva compromiterii de către un hoț.
- O formă de negare plauzibilă sau "portofel momeală", unde o frază-de-acces aleasă duce la un portofel cu o sumă mică de fonduri folosite pentru a distrage un atacator de la portofelul "real" care conține majoritatea fondurilor.

Cu toate acestea, este important să rețineți că utilizarea unei fraze-de-acces introduce, de asemenea, riscul de pierdere:

- Dacă proprietarul portofelului este incapacitat sau morț și nimeni altcineva nu cunoaște fraza-de-acces, sămânța este inutilă și toate fondurile stocate în portofel se pierd pentru totdeauna.
- În schimb, dacă proprietarul face o copie de rezervă a frazei-de-acces în același loc cu sămânța, atunci scopul unui al doilea factor își pierde rostul.

În timp ce frazele-de-acces sunt foarte utile, acestea ar trebui utilizate doar în combinație cu un proces planificat cu atenție pentru copii de rezervă și pentru recuperare, având în vedere posibilitatea de a supravețui mai mult decât proprietarul și permisând familiei sale să recupereze avereia în criptomonedă.

## **Lucrul cu coduri mnemonice**

BIP-39 este implementat ca o bibliotecă în mai multe limbaje de programare diferite:

### [\*\*python-mnemonic\*\*](#)

Implementarea referință a standardului de către echipa SatoshiLabs care a propus BIP-39, în Python

### [\*\*bitcoinjs/bip39\*\*](#)

O implementare a BIP-39, ca parte a bibliotecii populare bitcoinJS, în JavaScript

### [\*\*libbitcoin/mnemonic\*\*](#)

O implementare a BIP-39, ca parte a bibliotecii Libbitcoin, în C++

Există, de asemenea, un generator BIP-39 implementat într-o pagină web de sine stătătoare (standalone), extrem de utilă pentru testare și experimentare. [Un generator BIP-39 ca o pagină web de sine stătătoare](#) arată o pagină web statică care generează mnemonice, semințe și chei private extinse.

# Mnemonic

You can enter an existing BIP39 mnemonic, or generate a new random one. Typing your own twelve words will probably not work how you expect, since the words require a particular structure (the last word is a checksum)

For more info see the [BIP39 spec](#)

Generate a random  word mnemonic, or enter your own below.

<b>BIP39 Mnemonic</b>	army van defense carry jealous true garbage claim echo media make crunch
<b>BIP39 Passphrase (optional)</b>	
<b>BIP39 Seed</b>	5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fb77a89a1a3be4c6719 6f57c39a88b76373733891bfaba16ed27a813ceed498804c0570
<b>Coin</b>	Bitcoin
<b>BIP32 Root Key</b>	xprv9s21ZrQH143K3t4UZrNgeA3w861fwjYLaGwmPtQyPMmzshV2owVpfBSd2Q7YsHZ9j6 i6ddYjb5PLtUdMZn8LhvuCVhGcQntq5rn7JVMqnie

Figure 33. Un generator BIP-39 ca o pagină web de sine stătătoare

Pagina (<https://iancoleman.github.io/bip39/>) poate fi folosită offline într-un browser, sau accesată online.

## Crearea unui Portofel HD din Sămânță

Portofelele HD sunt create dintr-o singură *sămânță rădăcină* (root seed), care este un număr aleatoriu pe 128, 256 sau 512 biți. Cel mai frecvent, această sămânță este generată de o *mnemonică* aşa cum a fost detaliat în secțiunea anterioară.

Fiecare cheie din portofelul HD (determinist ierarhic) este derivată deterministic din această sămânță rădăcină (root seed), ceea ce face posibilă recrearea întregului portofel HD din această sămânță în orice portofel HD compatibil. Acest lucru vă ajută să creați o copie de rezervă, să restaurați, să exportați și să importați portofele HD care conțin mii sau chiar milioane de chei, prin simpla transferare a mnemoniciei din care se obține sămânța rădăcină.

Procesul de creare a cheilor principale (master keys) și a codului de lanț principal (master chain code) pentru un portofel HD este prezentat în [Crearea cheilor principale și a codului lanțului dintr-o sămânță rădăcină](#).

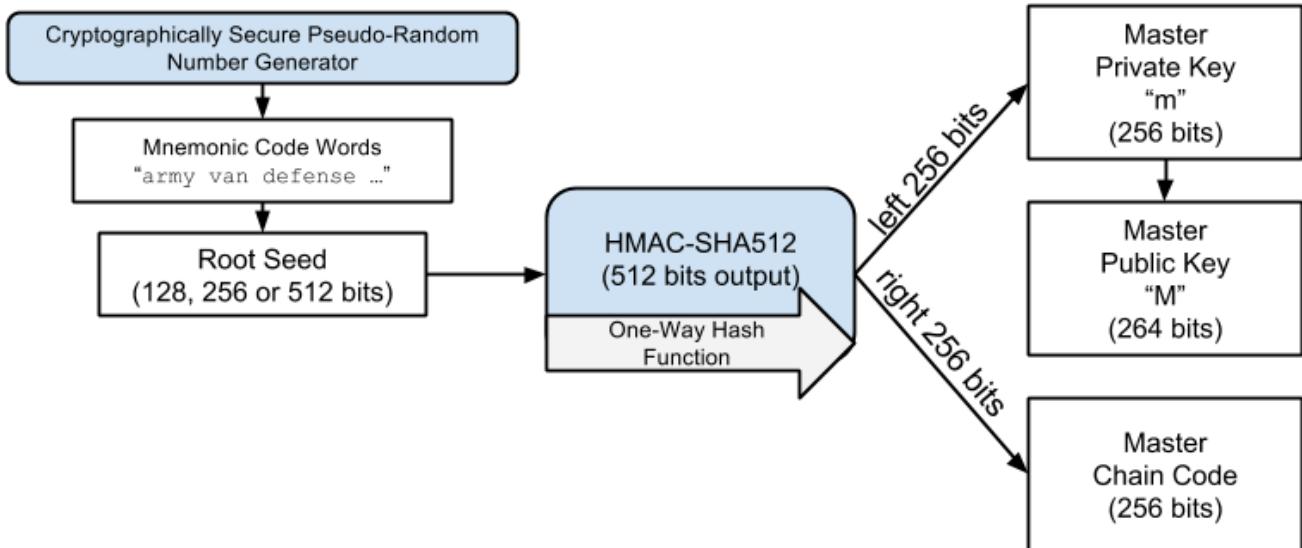


Figure 34. Crearea cheilor principale și a codului lanțului dintr-o sămânță rădăcină

Sămânța rădăcină (root seed) este folosită ca intrare în algoritmul HMAC-SHA512 și rezumatul (hash-ul) obținut este utilizat pentru a crea o *cheie privată principală* ( $m$ ) și un *cod de lanț principal* ( $c$ ).

Cheia privată principală (m) generează apoi o cheie publică principală (M) folosind procesul de înmulțire a curbei eliptice  $m * G$  pe care l-am văzut în [Chei Publice](#).

Codul de lanț (c) este utilizat pentru a introduce entropie în funcția care creează cheile copil din cheile părinte, așa cum vom vedea în secțiunea următoare.

## Derivarea cheii private copil

Portofelele HD folosesc o funcție de *derivare a cheilor copil* (child key derivation - CKD) pentru a deriva chei copil din cheile părinte.

Funcțiile de derivare a cheilor copil se bazează pe o funcție rezumat (hash) unidirecțională care combină:

- O cheie părinte privată sau publică (cheie comprimată ECDSA)
  - O sămânță numită cod de lanț (256 biți)
  - Un număr de indice (32 biți)

Codul lanțului (chain code) este utilizat pentru a introduce date aleatorii deterministe în proces, astfel încât cunoașterea indicelui și a unei chei copil nu este suficientă pentru a obține alte chei copil. Cunoașterea cheii unui copil nu face posibilă găsirea fraților săi, decât dacă aveți și codul lanțului. Sămânța de cod de lanț inițială (aflată la rădăcina arborelui) este derivată din sămânța originală, în timp ce codurile ulterioare de lanț pentru copii sunt deriveate din fiecare cod de lanț părinte.

Acstea trei elemente (cheie părinte, cod de lanț și indice) sunt combinate și rezumate (hashed) pentru a genera chei copii, după cum urmează.

Cheia publică părinte, codul lanțului și numărul de indice sunt combinate și rezumate (hashed) cu

algoritmul HMAC-SHA512 pentru a produce un rezumat (hash) de 512 biți. Acest rezumat de 512 biți este împărțit în două jumătăți de 256 biți. Cei 256 biți din jumătatea dreaptă a rezumatului devin codul lanțului pentru copil. Ceilalți 256 biți din jumătatea stângă a rezumatului sunt adăugați la cheia privată părinte pentru a produce cheia privată pentru copil. În [Extinderea unei chei private părinte pentru a crea o cheie privată copil](#), vedem acest lucru ilustrat cu indicele setat la 0 pentru a produce copilul "zero" (primul indice) al părintelui.

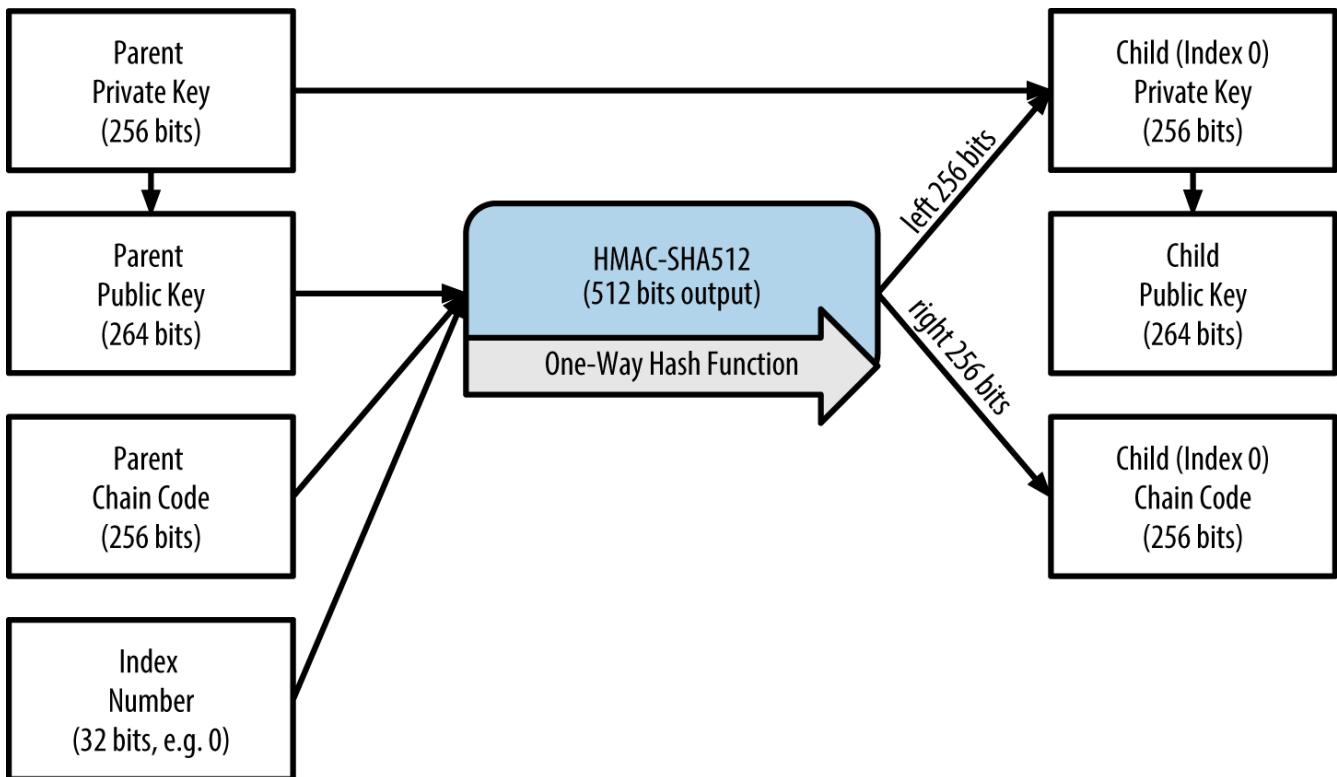


Figure 35. Extinderea unei chei private părinte pentru a crea o cheie privată copil

Modificarea indicelui ne permite să extindem părintele și să creăm ceilalți copii din secvență, de exemplu, Copil 0, Copil 1, Copil 2, etc. Fiecare cheie părinte poate avea  $2.147.483.647$  ( $2^{31}$ ) copii ( $2^{31}$  este jumătate din întregul interval  $2^{32}$  disponibil, deoarece cealaltă jumătate este rezervată unui tip special de derivare despre care vom vorbi mai târziu în acest capitol).

Repetând procesul cu un nivel mai jos, fiecare copil poate deveni la rândul său părinte și poate să-și creeze propriii copii, într-un număr infinit de generații.

### Utilizarea cheilor derivate din copil

Cheile private copii nu se disting de cheile nedeterministice (aleatorii). Deoarece funcția de derivare este o funcție unidirecțională, cheia copil nu poate fi utilizată pentru a găsi cheia părinte. De asemenea, cheia copilului nu poate fi folosită pentru a găsi vreun frate. Dacă aveți al  $n$ -ulea copil, nu puteți găsi frații săi, cum ar fi copilul  $n-1$  sau copilul  $n+1$ , sau alți copii care fac parte din secvență. Doar cheia părinților și codul de lanț pot deriva toți copiii. Fără codul lanțului pentru copii, nici cheia copil nu poate fi folosită pentru a obține vreun nepot. Aveți nevoie atât de cheia privată a copilului, cât și de codul lanțului copilului pentru a începe o nouă ramură și pentru a deriva nepoți.

Deci, la ce se poate folosi cheia privată a copilului de una singură? Poate fi folosită pentru a crea o cheie publică și o adresă bitcoin. Apoi, poate fi folosită pentru a semna tranzacții pentru a cheltui

orice a fost plătit la adresa respectivă.

**TIP**

Cheia privată copil, cheia publică corespunzătoare și adresa bitcoin sunt toate imposibil de deosebit față de chei și adrese create la întâmplare. Faptul că fac parte dintr-o secvență nu este vizibil în afara funcției de portofel HD care le-a creat. Odată create, acestea funcționează exact ca și cheile "normale".

## Chei extinse

Așa cum am văzut anterior, funcția de derivare a cheilor poate fi folosită pentru a crea copii la orice nivel al arborelui, bazat pe cele trei intrări: o cheie, un cod de lanț și indicele copilului dorit. Cele două ingrediente esențiale sunt cheia și codul lanțului, iar combinate sunt denumite o *cheie extinsă*. Termenul "cheie extinsă" ar putea fi, de asemenea, interpretat și ca "cheie extensibilă", deoarece o astfel de cheie poate fi folosită pentru a deriva copii.

Cheile extinse sunt stocate și reprezentate pur și simplu ca o concatenare a cheii de 256 biți și a codului de lanț de 256 biți într-o secvență de 512 biți. Există două tipuri de chei extinse. O cheie privată extinsă este combinația unei chei private și a unui cod de lanț și poate fi utilizată pentru a obține cheile private ale copiilor (și de la ele, cheile publice ale copiilor). O cheie publică extinsă este o cheie publică și un cod de lanț, care pot fi utilizate pentru a crea chei publice pentru copii (*doar publice*), așa cum este descris în [Generarea unei Chei Publice](#).

Gândiți-vă la o cheie extinsă ca la rădăcina unei ramuri din structura de arbore a portofelului HD. Cu rădăcina ramurii, puteți obține restul ramurii. Cheia privată extinsă poate crea o ramură completă, în timp ce cheia publică extinsă poate crea *doar* o ramură de chei publice.

**TIP**

O cheie extinsă constă dintr-o cheie privată sau publică și un cod de lanț. O cheie extinsă poate crea copii, generând propria ramură în structura arborelui. Distribuirea unei chei extinse oferă acces la întreaga ramură.

Cheile extinse sunt codificate folosind Base58Check, pentru a le exporta și importa cu ușurință între portofele compatibile BIP-32. Codificarea Base58Check pentru cheile extinse utilizează un număr special de versiune care are ca rezultat prefixele "xprv" și "xpub" atunci când sunt codificate în caractere Base58 pentru a le face ușor de recunoscut. Deoarece cheia extinsă este de 512 sau 513 biți, este, de asemenea, mult mai lungă decât alte șiruri codate Base58Check pe care le-am văzut anterior.

Iată un exemplu de cheie extinsă *privată*, codificată în Base58Check:

```
xprv9tyUQV64JT5qs3RSTJkXCWKMMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6Go
NMKUga5biW6Hx4tws2six3b9c
```

Iată cheia *publică* extinsă corespunzătoare, codificată în Base58Check:

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJnSD
MstwepyLXhRgPxdp14sk9tJPW9
```

## Derivarea cheii publice copil

Așa cum am menționat anterior, o caracteristică foarte utilă a portofelelor HD este capacitatea de a deriva cheile publice pentru copii din cheile publice părinte, *fără* a avea cheile private. Aceasta ne oferă două modalități de a obține o cheie publică a copilului: fie de la cheia privată a copilului, fie direct de la cheia publică a părintelui.

Prin urmare, o cheie publică extinsă poate fi utilizată pentru a deriva toate cheile *publice* (și doar cheile publice) din acea ramură a structurii portofelului HD.

Această scurtătură poate fi folosită pentru a crea implementări foarte sigure care folosesc exclusiv chei publice, unde un server sau o aplicație are o copie a unei chei publice extinse și nu are nici o cheie privată. Acest tip de implementare poate produce un număr infinit de chei publice și de adrese bitcoin, dar nu poate cheltui banii trimiși la respectivele adrese. Într timp, pe un alt server mai sigur, cheia privată extinsă poate deriva toate cheile private corespunzătoare pentru a semna tranzacții și a cheltui banii.

O aplicație obișnuită a acestei soluții este instalarea unei chei publice extinse pe un server web care servește o aplicație de comerț electronic. Serverul web poate utiliza funcția de derivare a cheii publice pentru a crea o nouă adresă bitcoin pentru fiecare tranzacție (de exemplu, pentru un coș de cumpărături pentru clienți). Serverul web nu va avea chei private care ar fi vulnerabile la furt. Fără portofele HD, singura modalitate de a face acest lucru este de a genera mii de adrese bitcoin pe un server securizat separat și apoi de a le preîncărca pe serverul de comerț electronic. Această abordare este greoaie și necesită o întreținere constantă pentru a vă asigura că serverul de comerț electronic nu "rămâne fără" chei.

O altă aplicație comună a acestei soluții este pentru portofelele de stocare la rece sau hardware. În acel scenariu, cheia privată extinsă poate fi stocată pe un portofel de hârtie sau pe un dispozitiv hardware (cum ar fi un portofel hardware Trezor), în timp ce cheia publică extinsă poate fi păstrată online. Utilizatorul poate crea adrese de "recepționare" după bunul plac, în timp ce cheile private sunt stocate în siguranță offline. Pentru a cheltui fondurile, utilizatorul poate utiliza cheia privată extinsă pe un client bitcoin care nu se conectează online sau poate semna tranzacții de pe portofelul hardware (de exemplu, Trezor). [Extinderea unei chei publice părinte pentru a crea o cheie publică copil](#) ilustrează mecanismul de extindere a unei chei publice părinte pentru a obține cheile publice ale copilului.

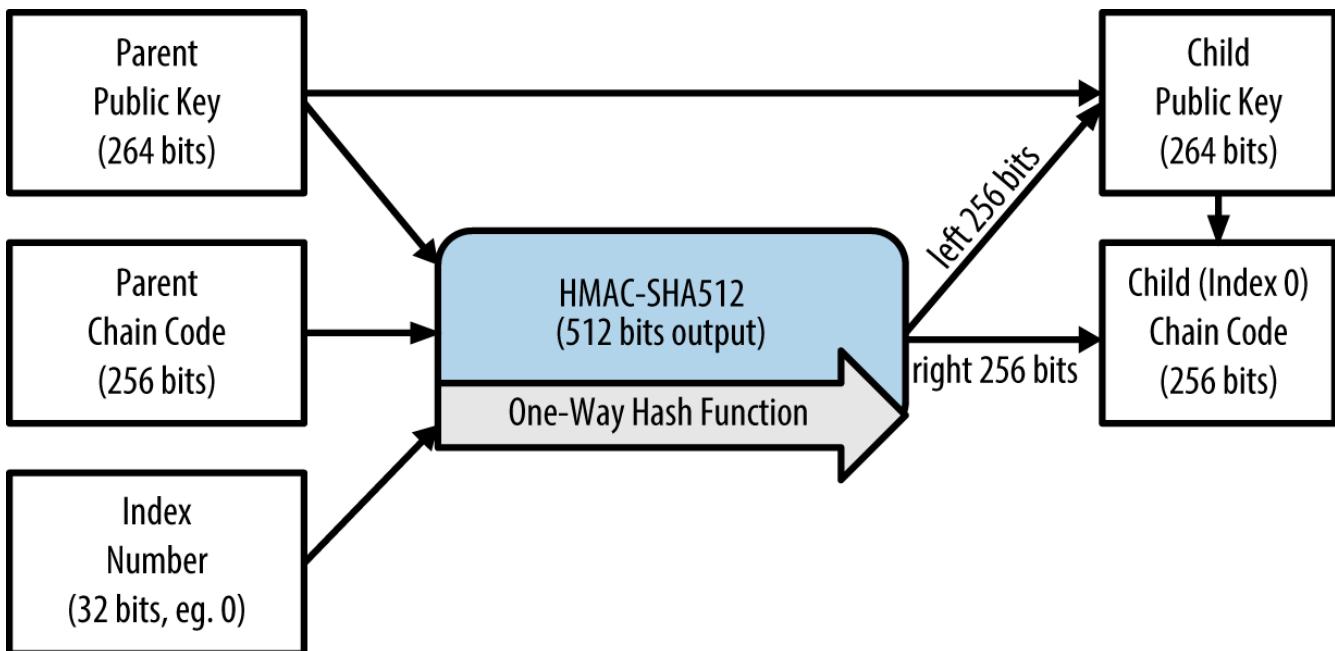


Figure 36. Extinderea unei chei publice părinte pentru a crea o cheie publică copil

## Utilizarea unei Chei Publice Extinse într-un Magazin Web

Să vedem cum sunt folosite portofelele HD continuând povestea noastră cu magazinul web al lui Gabriel.

Gabriel și-a înființat magazinul web din pasiune, bazat pe o simplă pagină Wordpress. Magazinul său era destul de simplu cu doar câteva pagini și un formular de comandă cu o singură adresă bitcoin.

Gabriel a folosit prima adresă bitcoin generată de dispozitivul său Trezor ca principală adresă bitcoin pentru magazinul său. În acest fel, toate plățile primite erau plătite la o adresă controlată de portofelul său hardware Trezor.

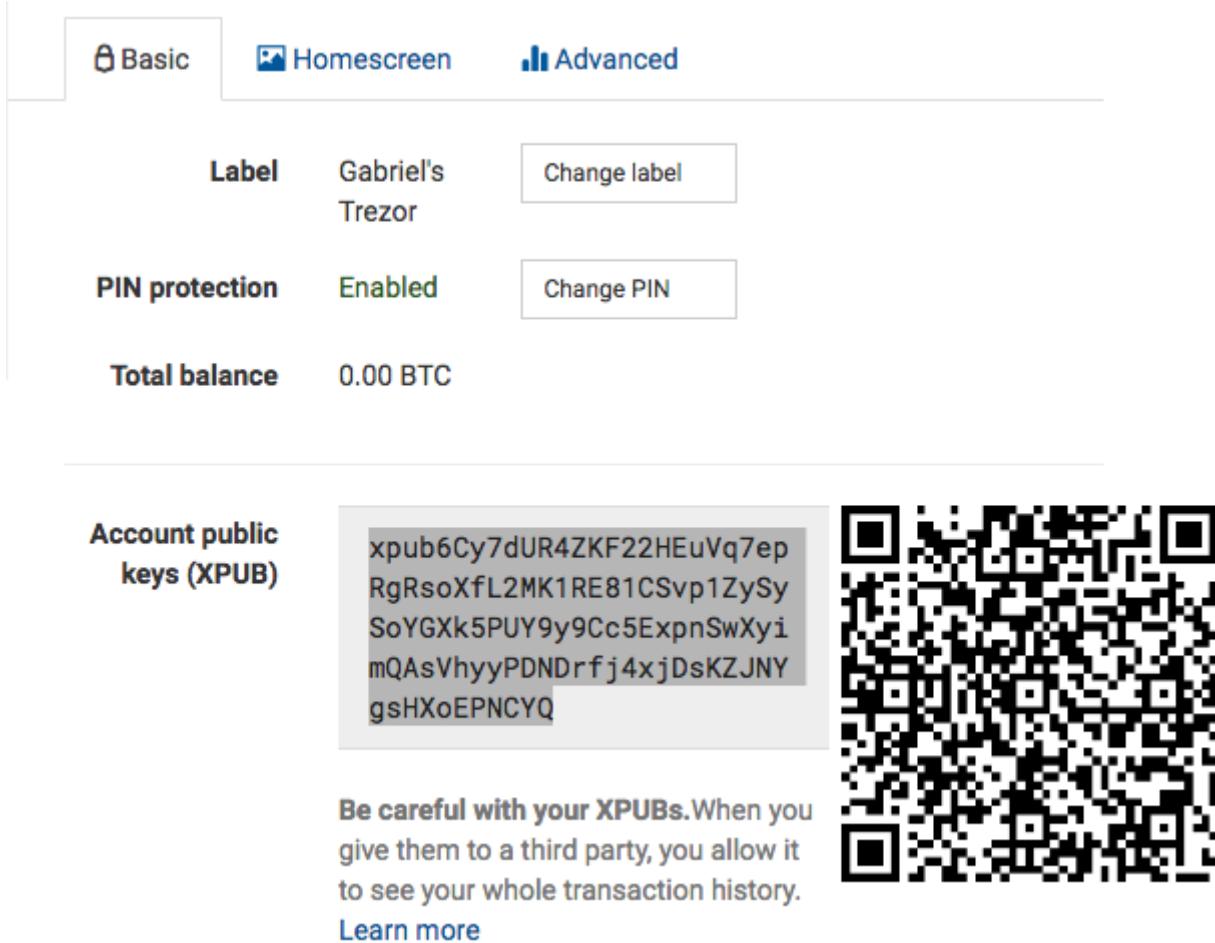
Clienții trimiteau o comandă folosind formularul și trimiteau plata pe adresa de bitcoin publicată de Gabriel, declanșând un e-mail cu detaliile comenzi pentru ca Gabriel să le proceseze. Cu doar câteva comenzi în fiecare săptămână, acest sistem a funcționat suficient de bine.

Cu toate acestea, micul magazin web a devenit destul de apreciat și a atras multe comenzi din partea comunității locale. Curând, Gabriel a fost copleșit. Cu toate comenziile care plăteau aceeași adresă, a devenit dificil să facă potrivirea corect dintre comenzi și tranzacții, mai ales atunci când mai multe comenzi pentru aceeași sumă erau primite la intervale scurte de timp.

Portofelul HD a lui Gabriel oferă o soluție mult mai bună prin posibilitatea de a deriva chei publice copii fără a cunoaște cheile private. Gabriel poate încărca o cheie publică extinsă (xpub) pe site-ul său web, care poate fi folosită pentru a obține o adresă unică pentru fiecare comandă a clienților. Gabriel poate cheltui fondurile folosind Trezor, dar xpub-ul încărcat pe site-ul web poate doar genera adrese și primi fonduri. Această caracteristică a portofelelor HD este o funcționalitate excelentă de securitate. Site-ul lui Gabriel nu conține chei private și, prin urmare, nu are nevoie de un nivel ridicat de securitate.

Pentru a exporta xpub-ul, Gabriel folosește software-ul web împreună cu portofelul hardware Trezor. Dispozitivul Trezor trebuie conectat pentru a exporta cheile publice. Rețineți că portofelele

hardware nu vor exporta niciodată cheile private - acestea rămân întotdeauna pe dispozitiv. [Exportarea unui xpub dintr-un portofel hardware Trezor](#) arată interfața web pe care Gabriel o folosește pentru a exporta xpub-ul.



The screenshot shows the Trezor web interface with the following details:

- Basic** tab is selected.
- Label**: Gabriel's Trezor (with a [Change label](#) button).
- PIN protection**: Enabled (with a [Change PIN](#) button).
- Total balance**: 0.00 BTC.
- Account public keys (XPUB)**: A text box displays the xpub key:  
xpub6Cy7dUR4ZKF22HEuVq7ep  
RgRs0XfL2MK1RE81CSvp1ZySy  
SoYGXk5PUY9y9Cc5ExpnSwXyi  
mQAsVhyyPDNDrfj4xjDsKZJNY  
gsHXoEPNCYQ
- Warning**: **Be careful with your XPUBs.** When you give them to a third party, you allow it to see your whole transaction history. [Learn more](#)
- QR code**: A large QR code representing the same xpub key.

Figure 37. Exportarea unui xpub dintr-un portofel hardware Trezor

Gabriel copiază xpub-ul pe magazinul lui online care utilizează plata cu bitcoin. El folosește *Mycelium Gear*, un plugin open source pentru magazine web pentru o varietate de platforme de găzduire web. Mycelium Gear folosește xpub pentru a genera o adresă unică pentru fiecare achiziție.

### Derivarea întărăită a cheii copil

Potibilitatea de a deriva o ramură de chei publice dintr-un xpub este foarte utilă, dar vine cu un potential risc. Accesul la un xpub nu oferă acces la cheile private ale copiilor. Cu toate acestea, deoarece xpub conține codul de lanț, dacă este cunoscută o cheie privată a copilului sau a fost cumva dezvăluită (leaked), ea poate fi folosită cu codul de lanț pentru a obține toate celelalte chei private pentru copii. O cheie privată cu un singur copil dezvăluit (leaked), împreună cu un cod de lanț părinte, dezvăluie toate cheile private ale tuturor copiilor. Mai rău, cheia privată pentru copii împreună cu un cod de lanț părinte pot fi folosite pentru a deduce cheia privată părinte.

Pentru a contracara acest risc, portofelele HD utilizează o funcție de derivare alternativă numită *derivare întărăită*, care "rupe" relația dintre cheia publică părinte și codul lanțului copil. Funcția de derivare întărăită utilizează cheia privată părinte pentru a deriva codul lanțului copil, în loc de cheia publică părinte. Aceasta creează un "firewall" în secvența părinte/copil, cu un cod de lanț

care nu poate fi utilizat pentru a compromite o cheie privată a unui părinte sau a unui frate. Funcția de derivare întărită pare aproape identică cu derivarea normală a cheii private pentru copii, cu excepția faptului că cheia privată părinte este utilizată ca intrare pentru funcția rezumat, în loc de cheia publică părinte, aşa cum se vede în diagrama din [Derivarea întărită a unei chei copil; omite cheia publică părinte](#).

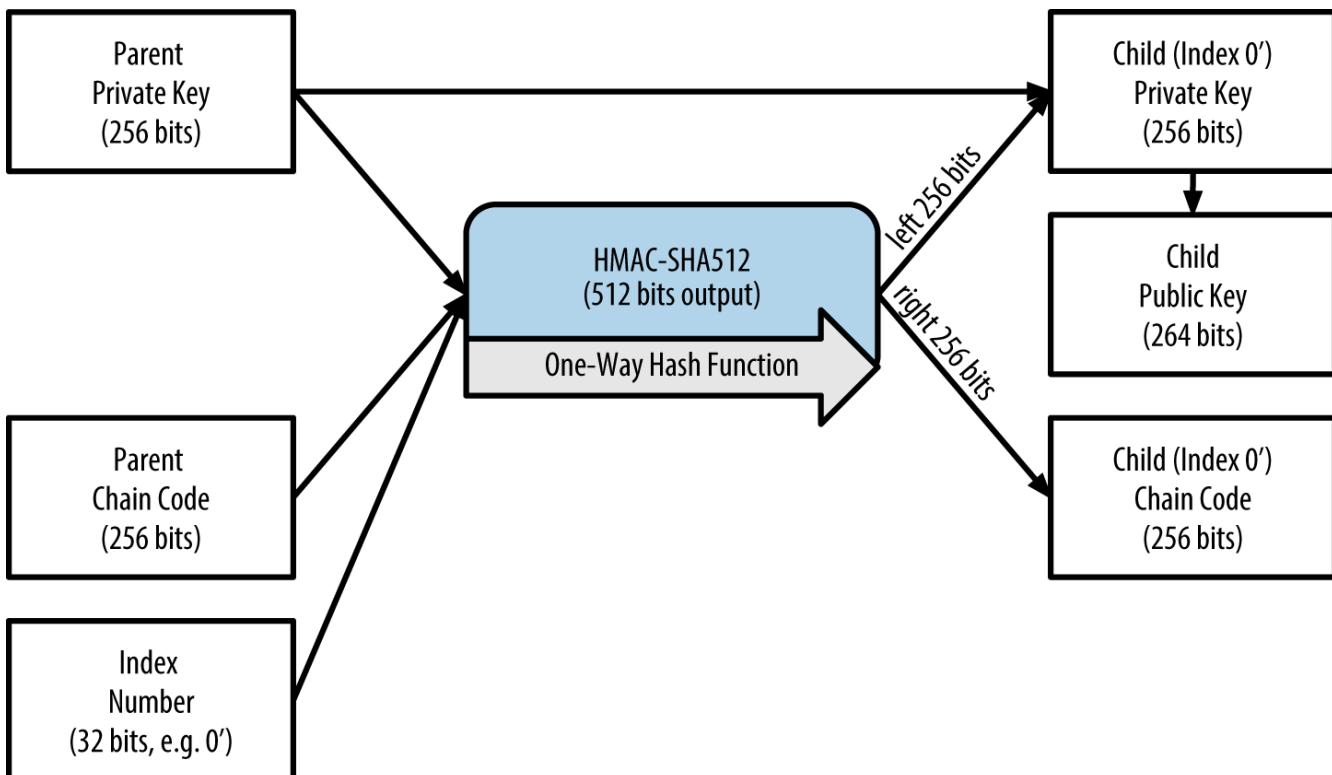


Figure 38. Derivarea întărită a unei chei copil; omite cheia publică părinte

Când se utilizează funcția de derivare privată întărită, cheia privată și codul lanțului copil rezultat sunt complet diferite de cea care ar rezulta din funcția normală de derivare. "Ramura" de chei rezultate poate fi utilizată pentru a produce chei publice extinse care nu sunt vulnerabile, deoarece codul de lanț pe care îl conțin nu poate fi exploatat pentru a dezvăluvi vreă cheie privată. Prin urmare, derivarea întărită este folosită pentru a crea un "gol" în arbore deasupra nivelului în care sunt utilizate cheile publice extinse.

În termeni simpli, dacă doriți să vă folosiți de comoditatea unui xpub pentru a obține ramuri de chei publice, fără să vă expuneți la riscul unui cod de lanț dezvăluit, ar trebui să îl derivați de la un părinte întărit, mai degrabă decât de la un părinte normal. Ca o bună practică, copiii de nivel 1 ai cheilor principale (master) sunt întotdeauna obținuți prin derivare întărită, pentru a preveni compromiterea cheilor principale.

### Numere de indice pentru derivare normală și întărită

Numărul de indice utilizat în funcția de derivare este un număr întreg pe 32 de biți. Pentru a distinge cu ușurință între cheile derivate prin funcția normală de derivare față de cheile obținute prin derivarea întărită, acest număr de indice este împărțit în două intervale. Numerele de indice între 0 și  $2^{31}-1$  (0x0 până la 0x7FFFFFFF) sunt utilizate *doar* pentru derivarea normală. Numerele de indice cuprinse între  $2^{31}$  și  $2^{32}-1$  (0x80000000 până la 0xFFFFFFFF) sunt utilizate *doar* pentru derivarea întărită. Prin urmare, dacă numărul indicelui este mai mic decât  $2^{31}$ , copilul este normal, iar dacă numărul indicelui este mai mare sau egal cu  $2^{31}$ , copilul este întărit.

Pentru a facilita citirea și afișarea numărului indicelui, acesta este afișat începând de la zero, dar cu simbolul prim pentru copiii întăriți. Prin urmare, prima cheie normală pentru copil este afișată ca 0, în timp ce primul copil întărit (indice 0x80000000) este afișat ca 0'. În secvență, a doua cheie întărită ar avea indicele 0x80000001 și va fi afișată ca 1', etc. Când vedeți un indice de portofel HD notat  $i'$ , asta înseamnă  $2^{31}+i$ .

### Identifierul cheii într-un portofel HD (calea)

Cheile dintr-un portofel HD sunt identificate folosind o convenție de denumire a "căii", cu fiecare nivel al arborelui separat printr-un caracter slash (/) (vezi [Exemple de cale pentru un portofel HD](#)). Cheile private derivate de la cheia privată principală (master) încep cu "m". Cheile publice derivate de la cheia publică principală (master) încep cu "M." Prin urmare, prima cheie privată copil a cheii private principale este m/0. Prima cheie publică copil este M/0. Al doilea nepot al primului copil este m/0/1 și așa mai departe.

"Strămoșii" unei chei se citesc de la dreapta la stânga, până când ajungeți la cheia principală de la care a fost derivată. De exemplu, identifierul m/x/y/z descrie cheia care este al z-ulea copil al cheii m/x/y, care este al y-ulea copil al cheii m/x, care este al x-ulea copil a lui m.

*Table 14. Exemple de cale pentru un portofel HD*

Calea HD	Descriere Cheie
m/0	Prima (0) cheie privată copil de la cheia privată principală (m)
m/0/0	Prima (0) cheie privată copil de la primul copil (m/0)
m/0'/0	Primul (0) copil normal de la primul copil întărit (m/0')
m/1/0	Prima (0) cheie privată copil de la al doilea copil (m/1)
M/23/17/0/0	Prima (0) cheie publică copil de la primul copil (M/23/17/0) de la al 18-lea copil (M/23/17) de la cel de-al 24-lea copil (M/23)

### Navigarea structurii arborescente a unui portofel HD

Structura arborescentă a portofelului HD oferă o flexibilitate extraordinară. Fiecare cheie părinte extinsă poate avea 4 miliarde de copii: 2 miliarde de copii normali și 2 miliarde de copii întăriți. Fiecare dintre acești copii poate avea alte 4 miliarde de copii și așa mai departe. Arborele poate fi cât de adânc doriți, cu un număr infinit de generații. Cu toată această flexibilitate, devine destul de dificil să navigați în acest arbore infinit. Este deosebit de dificil să transferați portofele HD între implementări, deoarece posibilitățile de organizare internă a ramurilor și sub-ramurilor sunt interminabile.

Două propuneri de îmbunătățire bitcoin (BIP) oferă o soluție la această complexitate prin crearea unor propuneri pentru structura arborilor portofelelor HD. BIP-43 propune utilizarea primului indice de copil întărit ca un identifier special care semnifică "scopul" structurii arborelui. Pe

baza BIP-43, un portofel HD ar trebui să utilizeze o singură ramură de nivel 1 a arborelui, numărul indicelui identificând structura și contextul restului arborelui, și astfel definindu-i scopul. De exemplu, un portofel HD care folosește numai ramura  $m/i'$  este destinat să semnifice un scop specific, iar acest scop este identificat cu numărul de indice "i".

Extinzând această specificație, BIP-44 propune o structură multicont ca număr "de scop"  $44'$  în conformitate cu BIP-43. Toate portofelele HD care urmează structura BIP-44 sunt identificate prin faptul că au folosit doar o ramură a arborelui:  $m/44'.$

BIP-44 specifică structura ca fiind formată din cinci niveluri predefinite ale arborelui:

```
m / scop' / tip_monedă' / cont' / rest / adresă_indice
```

"Scopul" de pe primul nivel este întotdeauna setat la  $44'$ . Al doilea nivel "tip\_monedă" specifică tipul de cripto monedă, permășând portofele HD multimonedă în care fiecare monedă are propriul său sub-arbore sub al doilea nivel. Există trei valute definite deocamdată: Bitcoin este  $m/44'/0$ , Bitcoin Testnet este  $m/44'/1$ , iar Litecoin este  $m/44'/2$ .

Al treilea nivel al arborelui este "cont", care permite utilizatorilor să își subdivizeze portofelele în subconturi logice separate, în scopuri contabile sau organizaționale. De exemplu, un portofel HD poate conține două "conturi" bitcoin:  $m/44'/0'/0$  și  $m/44'/0'/1$ . Fiecare cont este rădăcina propriului său sub-arbore.

La al patrulea nivel, "rest", un portofel HD are doi sub-arbore, unul pentru crearea adreselor de primire și unul pentru crearea adreselor de rest. Rețineți că, în timp ce nivelurile anterioare au utilizat o derivare întărită, acest nivel folosește derivarea normală. Acest lucru permite ca acest nivel al arborelui să explice chei publice extinse pentru utilizare într-un mediu nesigur. Adresele utilizabile sunt deriveate de portofelul HD ca fiind copii de nivelul al patrulea, ceea ce face ca al cincilea nivel al arborelui să fie "adresa\_indice". De exemplu, a treia adresă de primire pentru plăți bitcoin în contul principal ar fi  $M/44'/0'/0/2$ .[Exemple de structuri BIP-44 de portofel HD](#) arată alte câteva exemple.

Table 15. Exemple de structuri BIP-44 de portofel HD

Calea HD	Descriere Cheie
$M/44'/0'/0/2$	Cea de-a treia cheie publică de primire pentru contul bitcoin primar
$M/44'/0'/'/1/14$	Cea de-a cincisprezecea cheie publică pentru rest pentru al patrulea cont bitcoin
$m/44'/2'/0'/0/1$	A doua cheie privată din contul principal Litecoin, pentru semnarea tranzacțiilor

## Tranzacții

# Introducere

Tranzacțiile sunt cea mai importantă parte a sistemului bitcoin. Totul în bitcoin este conceput pentru a se asigura că tranzacțiile pot fi create, propagate în rețea, validate și, în sfârșit, adăugate în registrul global al tranzacțiilor (lanț-de-blocuri). Tranzacțiile sunt structuri de date care codifică transferul valorii între participanții la sistemul bitcoin. Fiecare tranzacție este o intrare publică în lanțul-de-blocuri (blockchain) bitcoin, registrul global de evidență contabilă.

În acest capitol vom examina toate formele de tranzacții, ce conțin, cum le creăm, cum sunt verificate și cum devin parte a înregistrării permanente a tuturor tranzacțiilor. Când folosim termenul "portofel" din acest capitol, ne referim la software-ul care construiește tranzacții, nu doar la baza de date de chei.

## Tranzacțiile în Detaliu

În [Cum Funcționează Bitcoin](#) am analizat, folosind un explorator de blocuri, tranzacția pe care Alice a folosit-o pentru a plăti cafeaua la Cafeneaua lui Bob([Tranzacția lui Alice către Cafeneaua lui Bob](#)).

Aplicația de explorator de blocuri (block explorer) arată o tranzacție de la "adresa" lui Alice la "adresa" lui Bob. Aceasta este o privire simplificată a ceea ce este conținut într-o tranzacție. De fapt, după cum vom vedea în acest capitol, o mare parte din informațiile prezentate sunt construite de exploratorul de blocuri și nu se află de fapt în tranzacție.

### Transaction View information about a bitcoin transaction

<a href="#">0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbda57286c345c2f2</a>	
<a href="#">1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK</a> (0.1 BTC - Output)	 <a href="#">1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA</a> - <a href="#">(Unspent)</a> 0.015 BTC <a href="#">1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK</a> - <a href="#">(Unspent)</a> 0.0845 BTC
	<a href="#">97 Confirmations</a> <span style="background-color: green; color: white; padding: 2px 5px;">0.0995 BTC</span>
<strong>Summary</strong>	<strong>Inputs and Outputs</strong>
Size <span style="float: right;">258 (bytes)</span>	Total Input <span style="float: right;">0.1 BTC</span>
Received Time <span style="float: right;">2013-12-27 23:03:05</span>	Total Output <span style="float: right;">0.0995 BTC</span>
Included In Blocks <span style="float: right;">277316 (2013-12-27 23:11:54 +9 minutes)</span>	Fees <span style="float: right;">0.0005 BTC</span>
	Estimated BTC Transacted <span style="float: right;">0.015 BTC</span>

Figure 39. Tranzacția lui Alice către Cafeneaua lui Bob

## Tranzacțiile - În Spatele Scenei

În spatele scenei, o tranzacție propriu zisă arată foarte diferit de o tranzacție oferită de un

explorator de blocuri obișnuit. De fapt, majoritatea conceptelor de nivel înalt pe care le vedem în diferitele interfețe cu utilizatorul ale aplicațiilor bitcoin *nu există de fapt* în sistemul bitcoin.

Putem folosi interfața liniei de comandă a Bitcoin Core (*getrawtransaction* și *decoderawtransaction*) pentru a prelua tranzacția "brută" a lui Alice, a o decoda și a vedea ce conține. Rezultatul arată astfel:

*Tranzacția lui Alice decodificată*

```
{  
  "version": 1,  
  "locktime": 0,  
  "vin": [  
    {  
      "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",  
      "vout": 0,  
      "scriptSig" :  
        "3045022100884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f039  
        ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL]  
        0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d172787e  
        c3457eee41c04f4938de5cc17b4a10fa336a8d752adf",  
        "sequence": 4294967295  
    }  
  ],  
  "vout": [  
    {  
      "value": 0.01500000,  
      "scriptPubKey": "OP_DUP OP_HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7  
      OP_EQUALVERIFY OP_CHECKSIG"  
    },  
    {  
      "value": 0.08450000,  
      "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8  
      OP_EQUALVERIFY OP_CHECKSIG",  
    }  
  ]  
}
```

Este posibil să observați câteva lucruri la această tranzacție, mai ales lucrurile care lipsesc! Unde este adresa lui Alice? Unde este adresa lui Bob? Unde este intrarea 0.1 "trimisă" de Alice? În bitcoin, nu există monede, nici expeditori, nici destinatari, nici solduri, nici conturi și nici adrese. Toate aceste lucruri sunt construite la un nivel superior pentru beneficiul utilizatorului, pentru a face lucrurile mai ușor de înțeles.

Este posibil să observați, de asemenea, o mulțime de câmpuri ciudate și indescifrabile și siruri de caractere hexazecimale. Nu vă faceți griji, în acest capitol vă vom explica în detaliu fiecare câmp prezentat aici.

# Ieșirile și Intrările Tranzacției

Piatra de căpătâi a unei tranzacții bitcoin este o *ieșire de tranzacție* (transaction output). Ieșirile tranzacției sunt bucăți indivizibile de monedă bitcoin, înregistrate pe lanțul-de-blocuri (blockchain) și recunoscute de întreaga rețea ca fiind valide. Nodurile complete bitcoin urmăresc toate ieșirile disponibile și cheltuibile, cunoscute sub denumirea de "ieșiri de tranzacție necheltuite" (unspent transaction outputs) sau *UTXO*. Colecția tuturor UTXO-urilor este cunoscută sub numele de *setul UTXO* și în prezent conține milioane de UTXO-uri. Setul UTXO crește pe măsură ce un UTXO nou este creat și se micșorează atunci când un UTXO este consumat. Fiecare tranzacție reprezintă o modificare (tranzacție de stare) în setul UTXO.

Când spunem că portofelul unui utilizator a "primit" bitcoin, la ce ne referim este că portofelul a detectat un UTXO care poate fi cheltuit cu una dintre cheile controlate de acel portofel. Astfel, "soldul" bitcoin al unui utilizator este suma tuturor UTXO pe care portofelul utilizatorului le poate cheltui și care poate fi dispersat între sute de tranzacții și sute de blocuri. Conceptul de balanță este creat de aplicația portofel. Portofelul calculează soldul utilizatorului scanând lanțul-de-blocuri (blockchain-ul) și agregând valoarea fiecărui UTXO pe care portofelul îl poate cheltui cu cheile pe care le controlează. Majoritatea portofelelor mențin o bază de date sau folosesc un serviciu de baze de date pentru a stoca un set de referință rapidă a tuturor UTXO-urilor pe care le pot cheltui cu cheile pe care le controlează.

O ieșire a unei tranzacții poate avea o valoare arbitrară (întreagă) exprimată ca multiplu de satoshi. La fel cum dolarii pot fi împărțiți în două zecimale ca centi, bitcoin poate fi împărțit în opt zecimale sub formă de satoshi. Deși o ieșire poate avea orice valoare arbitrară, odată creată, aceasta este indivizibilă. Aceasta este o caracteristică importantă a ieșirilor care trebuie subliniată: ieșirile sunt unități de valoare *discrete și indivizibile*, exprimate în satoshi întregi. O ieșire neutilizată poate fi consumată numai în totalitate printr-o tranzacție.

Dacă o UTXO este mai mare decât valoarea dorită a unei tranzacții, ea trebuie să fie consumată în întregime și restul trebuie să fie generat în tranzacție. Cu alte cuvinte, dacă aveți o UTXO în valoare de 20 bitcoin și doriți să plătiți doar 1 bitcoin, tranzacția dumneavoastră trebuie să consume întregul UTXO de 20 de bitcoin și să producă două ieșiri: una plătind 1 bitcoin destinatarului dorit și alta plătind 19 bitcoin ca rest înapoi în portofel. Ca urmare a naturii indivizibile a ieșirilor de tranzacție, majoritatea tranzacțiilor bitcoin vor trebui să genereze rest.

Imaginați-vă că un client cumpără o băutură de 1,50 USD, uitându-se în portofel pentru a găsi o combinație de monede și bancnote pentru a acoperi costul de 1,50 USD. Clientul va alege suma exactă dacă este disponibilă, de exemplu, o bancnotă de un dolar și două monede de 25 de centi sau o combinație de 6 monede de 25 de centi sau, dacă este necesar, o unitate mai mare, cum ar fi o bancnotă de 5 dolari. Dacă înmânează prea mulți bani, să zicem 5 dolari, clientul se va aștepta să primească rest în valoare de 3,50 USD, pe care îl va pune înapoi în portofel și va fi disponibil pentru tranzacțiile viitoare.

În mod similar, o tranzacție bitcoin trebuie creată din UTXO-urile unui utilizator, indiferent de valoarea pe care o are utilizatorul. Utilizatorii nu pot să ia un UTXO în jumătate așa cum nu pot să ia o bancnotă în jumătate și să o folosească apoi. Aplicația portofel a utilizatorului va selecta, de obicei, dintre UTXO-urile disponibile astfel încât va compune o sumă mai mare sau egală cu valoarea tranzacției dorite.

La fel ca în viața reală, aplicația bitcoin poate folosi mai multe strategii pentru a compune suma de cumpărare: combinarea mai multor unități mai mici, găsirea unei sume exacte sau utilizarea unei singure unități mai mari decât valoarea tranzacției și primirea restului. Toate aceste asamblări complexe de UTXO-uri cheltuibile sunt realizate automat de portofelul utilizatorului și sunt invizibile pentru utilizatori. Este relevant numai dacă construți tranzacții brute din UTXO-uri în mod programatic.

O tranzacție consumă ieșiri de tranzacție necheltuite anterior și creează noi ieșiri de tranzacție care pot fi consumate de o tranzacție viitoare. În acest fel, bucăți de valoare bitcoin înaintează de la proprietar la proprietar într-un lanț de tranzacții care consumă și creează UTXO.

Excepția de la lanțul de ieșiri și de intrări este un tip special de tranzacție numit tranzacție *cu monedă de bază* (coinbase), care este prima tranzacție din fiecare bloc. Această tranzacție este plasată acolo de minerul "câștigător" și creează bitcoin nou plătit aceluui miner drept recompensă pentru minerit. Această tranzacție specială cu monedă de bază (coinbase) nu consumă UTXO; în schimb, are un tip special de intrare numit coinbase. Așa este creată rezerva monetară bitcoin în timpul procesului de minerit, după cum vom vedea în [Minerit și Consens](#).

**TIP** Ce a fost mai întâi? Intrările sau ieșirile, găina sau oul? Strict vorbind, ieșirile au fost mai întâi pentru că tranzacțiile coinbase, care generează bitcoin nou, nu au intrări și creează ieșiri din nimic.

## Ieșirile Tranzacției

Fiecare tranzacție bitcoin creează ieșiri, care sunt înregistrate în registrul bitcoin. Aproape toate aceste ieșiri, cu o excepție (vezi [Ieșire de Înregistrare a Datelor \(RETURN\)](#)) creează bucăți cheltuibile de bitcoin numite UTXO, care sunt apoi recunoscute de întreaga rețea și disponibile pentru proprietar să le cheltuiască într-o tranzacție viitoare.

UTXO-urile sunt monitorizate în setul UTXO de fiecare client nod-complet bitcoin. Tranzacțiile noi consumă (cheltuiesc) una sau mai multe dintre aceste ieșiri din setul UTXO.

Ieșirile tranzacției sunt compuse din două părți:

- O cantitate de bitcoin, exprimată în *satoshi*, cea mai mică unitate bitcoin
- Un puzzle criptografic care determină condițiile necesare pentru a cheltui ieșirea

Puzzle-ul criptografic este cunoscut și ca un *script de blocare*, un *script martor* sau un *scriptPubKey*.

Limbajul de script al tranzacției, utilizat în scriptul de blocare menționat anterior, este discutat în detaliu în [Scripturi de Tranzacție și Limbaj de Scriptare](#).

Acum, să ne uităm la tranzacția lui Alice (prezentată anterior în [Tranzacțiile - În Spatele Scenei](#)) și să vedem dacă putem identifica ieșirile. În codarea JSON, ieșirile sunt într-o listă numită *vout*:

```

"vout": [
  {
    "value": 0.01500000,
    "scriptPubKey": "OP_DUP OP_HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
OP_EQUALVERIFY
OP_CHECKSIG"
  },
  {
    "value": 0.08450000,
    "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8
OP_EQUALVERIFY OP_CHECKSIG",
  }
]

```

După cum puteți vedea, tranzacția conține două ieșiri. Fiecare ieșire este definită de o valoare și un puzzle criptografic. În codificarea afișată de Bitcoin Core, valoarea este afișată în bitcoin, dar în tranzacția în sine este înregistrată ca un număr întreg exprimat în satoshi. A doua parte a fiecărei ieșiri este puzzle-ul criptografic care stabilește condițiile pentru cheltuire. Bitcoin Core arată acest lucru ca *scriptPubKey* și ne arată o reprezentare care poate fi citită de către un om.

Subiectul de blocare și deblocare a UTXO-urilor va fi discutat mai târziu, în [Construcția Scriptului \(Blocare + Deblocare\)](#). Limbajul de script folosit pentru scriptul din *scriptPubKey* este discutat în [Scripturi de Tranzacție și Limbaj de Scriptare](#). Dar înainte de a aprofunda aceste subiecte, trebuie să înțelegem structura generală a intrărilor și ieșirilor unei tranzacții.

### Serializarea tranzacțiilor - ieșiri

Când tranzacțiile sunt transmise prin rețea sau schimbate între aplicații, acestea sunt *serialized*. Serializarea este procesul de transformare a reprezentării interne a unei structuri de date într-un format care poate fi transmis octet cu octet, cunoscut și sub numele de flux de octeți. Serializarea este cel mai frecvent utilizată pentru codificarea structurilor de date pentru transmisie prin rețea sau pentru stocarea într-un fișier. Formatul de serializare al unei ieșiri a tranzacției este prezentat în [Serializare ieșire tranzacție](#).

Table 16. Serializare ieșire tranzacție

Dimensiune	Câmp	Descriere
8 octeți (little-endian)	Sumă	Valoarea Bitcoin în satoshi ( $10^{-8}$ bitcoin)
1–9 octeți (VarInt)	Dimensiunea scriptului de blocare	Lungimea scriptului în octeți, va urma
Variabilă	Script de Blocare	Un script care definește condițiile necesare pentru a cheltui ieșirea

Majoritatea bibliotecilor și framework-urilor bitcoin nu stochează tranzacțiile intern ca fluxuri de octeți, deoarece acest lucru ar necesita o parcurgere complexă de fiecare dată când aveți nevoie să accesați un singur câmp. Pentru comoditate și lizibilitate, bibliotecile bitcoin stochează tranzacțiile

intern în structuri de date (de obicei structuri orientate pe obiect).

Procesul de conversie din reprezentarea flux-de-octeți a unei tranzacții în structura de date folosită intern de o bibliotecă se numește *deserializare* sau *parcurgerea tranzacției*. Procesul de conversie înapoi într-un flux-de-octeți pentru a fi transmis prin rețea, pentru rezumare (hashing) sau pentru stocarea pe disc se numește *serializare*. Majoritatea bibliotecilor bitcoin au funcții integrate pentru serializarea și deserializarea tranzacțiilor.

Vedeți dacă puteți decodifica manual tranzacția lui Alice din forma hexazecimală serializată, găsind unele dintre elementele pe care le-am văzut anterior. Secțiunea care conține cele două ieșiri este evidențiată în [Tranzacția lui Alice, serializată și prezentată în notare hexazecimală](#) pentru a vă ajuta:

*Example 18. Tranzacția lui Alice, serializată și prezentată în notare hexazecimală*

```
0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd73
4d2804fe65fa35779000000008b483045022100884d142d86652a3f47
ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f039
ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813
01410484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade84
16ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc1
7b4a10fa336a8d752adfffffffff0260e31600000000001976a914ab6
8025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef8000000000000
1976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac 00000000
```

Iată câteva indicii:

- Există două ieșiri în secțiunea evidențiată, fiecare serializată așa cum se arată în [Serializare ieșire tranzacție](#).
- Valoarea de 0,015 bitcoin este 1.500.000 de satoshi. Adică *16 e3 60* în hexazecimal.
- În tranzacția serializată, valoarea *16 e3 60* este codificată în little-endian (cel mai puțin semnificativ-octet-primul), deci arată ca *60 e3 16*.
- Lungimea *scriptPubKey* este de 25 octeți, ceea ce este *19* în hexazecimal.

## Intrările Tranzacției

Intrările tranzacției identifică (prin referință) care UTXO va fi consumat și oferă dovada proprietății printr-un script de deblocare.

Pentru a construi o tranzacție, un portofel selectează dintre UTXO-urile pe care le controlează, UTXO-uri cu suficientă valoare pentru a efectua plata solicitată. Uneori este suficient o UTXO, alte ori este nevoie de mai mult de una. Pentru fiecare UTXO care va fi consumată pentru a efectua această plată, portofelul creează o intrare care referențiază către UTXO și o deblochează cu un script de deblocare.

Să analizăm mai detaliat componentele unei intrări. Prima parte a unei intrări este un indicator către o UTXO prin referire la rezumatul (hash-ul) tranzacției și un index de ieșire, care identifică UTXO-ul specific în tranzacția respectivă. A doua parte este un script de deblocare, pe care

portofelul îl construiește pentru a satisface condițiile de cheltuieli stabilite în UTXO. Cel mai adesea, scriptul de deblocare este o semnătură digitală și o cheie publică care dovedește proprietatea asupra bitcoin-ului. Cu toate acestea, nu toate scripturile de deblocare conțin semnături. A treia parte este un număr de secvență, despre care vom discuta mai târziu.

Luați în considerare exemplul nostru din [Tranzacțiile - În Spatele Scenei](#). Intrările tranzacției sunt un o listă numit *vin*:

*Intrările tranzacției în tranzacția lui Alice*

```
"vin": [
  {
    "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
    "vout": 0,
    "scriptSig" :
      "3045022100884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f039
      ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL]
      0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d172787e
      c3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
    "sequence": 4294967295
  }
]
```

După cum vedeti, există o singură intrare în listă (deoarece un UTXO conținea suficientă valoare pentru a efectua această plată). Intrarea conține patru elemente:

- Un ID de tranzacție, care face referire la tranzacția care conține UTXO ce urmează să fie cheltuit
- Un indice de ieșire (*vout*), care identifică la care UTXO se face referire din tranzacția respectivă (primul este zero)
- Un *scriptSig*, care satisface condițiile plasate pe UTXO, deblocându-l pentru a fi cheltuit
- Un număr de secvență (care va fi discutat mai târziu)

În tranzacția lui Alice, intrarea indică ID-ul tranzacției:

```
7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18
```

și indexul de ieșire 0 (adică, primul UTXO creat de acea tranzacție). Scriptul de deblocare este construit de către portofelul lui Alice. Portofelul extrage mai întâi UTXO-ul referit, îi examinează scriptul de blocare, apoi îl utilizează pentru a crea scriptul de deblocare necesar.

Analizând doar intrarea, este posibil să fi observat că nu știm nimic despre acest UTXO, decât o referire la tranzacția care îl conține. Nu știm valoarea acesteia (suma în satoshi) și nu știm scriptul de blocare care stabilește condițiile pentru cheltuirea acesteia. Pentru a găsi aceste informații, trebuie să obținem UTXO-ul referit prin găsirea tranzacției în care a fost inclus. Observați că, deoarece valoarea intrării nu este specificată explicit, trebuie să folosim și UTXO-ul referit pentru a calcula comisioanele care vor fi plătite în această tranzacție (vezi [Comisioanele de tranzacție](#)).

Nu este doar portofelul lui Alice care trebuie să obțină UTXO-ul la care se face referire în intrări. Odată ce această tranzacție este transmisă în rețea, fiecare nod de validare va trebui, de asemenea, să recupereze UTXO-ul la care se face referire în intrările tranzacției pentru a valida tranzacția.

Tranzacțiile de unele singure par incomplete, deoarece le lipsește contextul. Ele referă UTXO-uri în intrările lor, dar fără a prelua acele UTXO-uri nu putem cunoaște valoarea intrărilor sau condițiile de blocare ale acestora. Când scrieți software bitcoin, oricând decodați o tranzacție cu intenția de a o valida, de a calcula comisioanele sau de a verifica scriptul de deblocare, codul dumneavoastră va trebui mai întâi să obțină UTXO-ul referit din lanțul-de-blocuri (blockchain) pentru a construi contextul insinuat, dar care nu este prezent în referințele UTXO ale intrărilor. De exemplu, pentru a calcula suma plătită pentru comisioane, trebuie să cunoașteți suma valorilor intrărilor și ieșirilor. Dar fără a prelua UTXO-urile la care se face referire în intrări, nu le cunoașteți valoarea. Așadar, o operație aparent simplă, precum calcularea comisioanelor într-o singură tranzacție implică de fapt mai multe etape și date din mai multe tranzacții.

Putem folosi aceeași secvență de comenzi din Bitcoin Core ca atunci când am obținut tranzacția lui Alice (*getrawtransaction* și *decoderawtransaction*). Cu aceaste comenzi putem obține UTXO-ul la care se face referire în intrarea precedentă și să aruncăm o privire:

*UTXO-ul lui Alice din tranzacția anterioară, la care se face referire în intrare*

```
"vout": [
  {
    "value": 0.10000000,
    "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeea53a8f3cc025a8
OP_EQUALVERIFY OP_CHECKSIG"
  }
]
```

Vedem că acest UTXO are o valoare de 0,1 BTC și că are un script de blocare (*scriptPubKey*) care conține "OP\_DUP OP\_HASH160 ...".

**TIP**

Pentru a înțelege pe deplin tranzacția lui Alice, a trebuit să obținem tranzacția(iile) anterioară(e) la care am făcut referință. O funcție care obține tranzacțiile anterioare și ieșirile necheltuite ale tranzacțiilor este foarte frecventă și există în aproape fiecare bibliotecă și API bitcoin.

### Serializarea tranzacțiilor - intrări

Când tranzacțiile sunt serializate pentru a fi transmise în rețea, intrările lor sunt codificate într-un flux de octeți, așa cum se arată în [Serializare intrare tranzacție](#).

Table 17. Serializare intrare tranzacție

Dimensiune	Câmp	Descriere
32 octeți	Rezumat (Hash) Tranzacție	Indicator la tranzacția care conține UTXO-ul care trebuie cheltuit

Dimensiune	Câmp	Descriere
4 octeți	Indicele Ieșire	Numărul de index al UTXO care trebuie cheltuit; primul este 0
1–9 octeți (VarInt)	Dimensiunea Scriptului de Deblocare	Lungimea Scriptului de Deblocare în octeți, va urma
Variabilă	Script de Deblocare	Un script care îndeplinește condițiile scriptului de blocare UTXO
4 octeți	Număr de Secvență	Folosit pentru timpul de blocare (locktime) sau dezactivat (0xFFFFFFFF)

Ca și în cazul ieșirilor, să vedem dacă putem găsi intrările din tranzacția lui Alice în format serializat. În primul rând, intrările decodate:

```
"vin": [
  {
    "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
    "vout": 0,
    "scriptSig" :
      "3045022100884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f039
      ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL]
      0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d172787e
      c3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
    "sequence": 4294967295
  }
],
```

Acum, să vedem dacă putem identifica aceste câmpuri în codificarea hexa în [Tranzacția lui Alice, serializată și prezentată în notare hexazecimală](#):

*Example 19. Tranzacția lui Alice, serializată și prezentată în notare hexazecimală*

```
0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd73
4d2804fe65fa35779000000008b483045022100884d142d86652a3f47
ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f039
ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813
01410484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade84
16ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc1
7b4a10fa336a8d752adffffffff0260e31600000000001976a914ab6
8025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef8000000000000
1976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac00000 000
```

Sugestii:

- ID-ul tranzacției este serializat în ordine inversată a octetilor, deci începe cu (hex) 18 și se

- Indexul ieșirii este un grup de 4 biți de zerouri, ușor de identificat
- Lungimea *scriptSig* este de 139 octeți, sau *8b* în hexa
- Numărul de secvență este setat la *FFFFFFFF*, din nou ușor de identificat

## Comisioanele de tranzacție

Majoritatea tranzacțiilor includ comisioane de tranzacție, care recompensează minerii bitcoin pentru securizarea rețelei. Comisioanele servesc, de asemenea, ca un mecanism de securitate, făcând imposibil din punct de vedere economic ca atacatorii să inunde rețeaua cu tranzacții. Mineritul, comisioanele și recompensele încasate de mineri sunt discutate mai detaliat în [Minerit și Consens](#).

Această secțiune examinează modul în care comisioanele de tranzacție sunt incluse într-o tranzacție obișnuită. Majoritatea portofelelor calculează și includ automat comisioanele de tranzacție. Cu toate acestea, dacă construți tranzacții programatic sau utilizați o interfață din linia de comandă, trebuie să vă contabilizați și să includeți aceste comisioane manual.

Comisioanele de tranzacție sunt un stimulent pentru includerea (mineritul) unei tranzacții în următorul bloc și, de asemenea, ca descurajare împotriva abuzurilor asupra sistemului, impunând un cost mic pentru fiecare tranzacție. Comisioanele de tranzacție sunt colectate de minerul care minerește blocul ce înregistrează tranzacția în lanțul-de-blocuri (blockchain).

Comisioanele de tranzacție sunt calculate în funcție de mărimea tranzacției în kilo-octeți, nu de valoarea tranzacției în bitcoin. În general, comisioanele de tranzacție sunt stabilite în funcție de forțele pieței din cadrul rețelei bitcoin. Minerii acordă prioritate tranzacțiilor pe baza mai multor criterii diferite, inclusiv comisioanele și chiar pot procesa tranzacții gratuit în anumite circumstanțe. Comisioanele de tranzacție afectează prioritatea procesării, ceea ce înseamnă că o tranzacție cu comisioane suficiente este probabil să fie inclusă în următorul bloc minat, în timp ce o tranzacție cu comisioane insuficiente sau fără comisioane ar putea fi întârziată, procesată după câteva blocuri sau neprocesată deloc. Comisioanele de tranzacție nu sunt obligatorii, iar tranzacțiile fără comisioane pot fi procesate în cele din urmă; cu toate acestea, includerea comisioanelor de tranzacție încurajează procesarea prioritată.

De-a lungul timpului, modul în care se calculează comisioanele de tranzacție și efectul pe care îl au asupra priorității tranzacțiilor a evoluat. La început, comisioanele de tranzacție au fost fixe și constante în toată rețeaua. Treptat, structura comisioanelor s-a relaxat și poate fi influențată de forțele pieței, pe baza capacitatei rețelei și a volumului tranzacțiilor. Încă de la începutul anului 2016, limitele de capacitate ale bitcoin au creat concurență între tranzacții, rezultând în comisioane mai mari și făcând de domeniu trecutului tranzacțiile gratuite. Tranzacțiile cu comision zero sau foarte mic sunt rareori minate și uneori nici măcar nu vor fi propagate în rețea.

În Bitcoin Core, politicile comisionului de releu (relay) sunt stabilite prin opțiunea [minrelaytxfee](#). Valoarea implicită actuală [minrelaytxfee](#) este 0,00001 bitcoin sau o sutime de milibitcoin pe kilo-octet. Prin urmare, în mod implicit, tranzacțiile cu un comision mai mic de 0,00001 bitcoin sunt tratate ca fiind gratuite și sunt transmise doar dacă există spațiu în mempool; în caz contrar, sunt abandonate. Nodurile Bitcoin pot înlocui politica de releu a comisioanelor implicate prin ajustarea valorii [minrelaytxfee](#).

Orice serviciu bitcoin care creează tranzacții, inclusiv portofele, burse de schimb, aplicații de retail etc., trebuie să implementeze comisioane dinamice. Comisioanele dinamice pot fi implementate printr-un serviciu terț de estimare a comisioanelor sau cu un algoritm de estimare al comisioanelor încorporat. Dacă nu sunteți sigur, începeți cu un serviciu extern și, pe măsură ce aveți experiență, puteți să proiectați și să implementați propriul algoritm dacă doriți să eliminați dependența față de terți.

Algoritmii de estimare a tarifelor calculează comisionul corespunzător, pe baza capacitatei și a comisioanelor oferite de tranzacțiile "concurrente". Acești algoritmi variază de la simplist (comision mediu sau median din ultimul bloc) la sofisticat (analiză statistică). Aceștia estimează comisionul necesar (în satoshi per octet) care va oferi unei tranzacții o probabilitate ridicată de a fi selectată și inclusă într-un anumit număr de blocuri. Majoritatea serviciilor oferă utilizatorilor opțiunea de a alege comisioane cu prioritate mare, medie sau mică. Prioritate mare înseamnă că utilizatorii plătesc comisioane mai mari, dar tranzacția va fi probabil inclusă în următorul bloc. Prioritate medie și scăzută înseamnă că utilizatorii plătesc comisioane de tranzacție mai mici, dar tranzacțiile pot dura mult mai mult pentru a fi confirmate.

Multe aplicații portofel utilizează servicii terțe pentru calcularea comisioanelor. Un serviciu popular este <https://bitcoinfees.earn.com/>, care oferă un API și un grafic vizual care arată comisionul în satoshi/octet pentru diferite priorități.

**TIP** Comisioanele statice nu mai sunt viabile în rețeaua bitcoin. Portofelele care stabilesc comisioane statice vor oferi o experiență slabă a utilizatorului, deoarece tranzacțiile vor fi adesea "blocate" și vor rămâne neconfirmate. Utilizatorii care nu înțeleg tranzacțiile și tarifele bitcoin sunt buimăciți de tranzacțiile "blocate", deoarece cred că și-au pierdut banii.

Diagrama din [Serviciul de estimare a comisioanelor bitcoinfees.earn.com](https://bitcoinfees.earn.com) prezintă estimarea în timp real a comisioanelor în segmente de 10 satoshi/octet și timpul de confirmare preconizat (în minute și număr de blocuri) pentru tranzacțiile cu comisioane din fiecare interval. Pentru fiecare interval de comisioane (de exemplu, 61–70 satoshi/octet), două bare orizontale indică numărul de tranzacții neconfirmate (1405) și numărul total de tranzacții din ultimele 24 de ore (102.975), cu comisioanele în acest interval. Pe baza graficului, comisionul recomandat pentru prioritate ridicată la acel moment a fost de 80 satoshi/octet, un comision care ar fi făcut ca tranzacția să fie minată chiar în următorul bloc (întârziere bloc zero). În perspectivă, dimensiunea medie a tranzacției este de 226 de octeți, deci comisionul recomandat pentru o dimensiune a tranzacției ar fi de 18,080 satoshi (0,00018080 BTC).

Datele de estimare a comisioanelor pot fi obținute printr-un simplu apel HTTP REST, la API-ul <https://bitcoinfees.earn.com/api/v1/fees/recommended>. De exemplu, din linia de comandă folosind comanda `curl`:

*Utilizarea API-ului de estimare a comisioanelor*

```
$ curl https://bitcoinfees.earn.com/api/v1/fees/recommended
```

```
{"fastestFee":80,"halfHourFee":80,"hourFee":60}
```

API-ul returnează un obiect JSON cu estimarea curentă a comisioanelor pentru confirmarea cea mai rapidă (`fastestFee`), confirmarea până în trei blocuri (`halfHourFee`) și până în șase blocuri (`hourFee`), în satoshi pe octet.

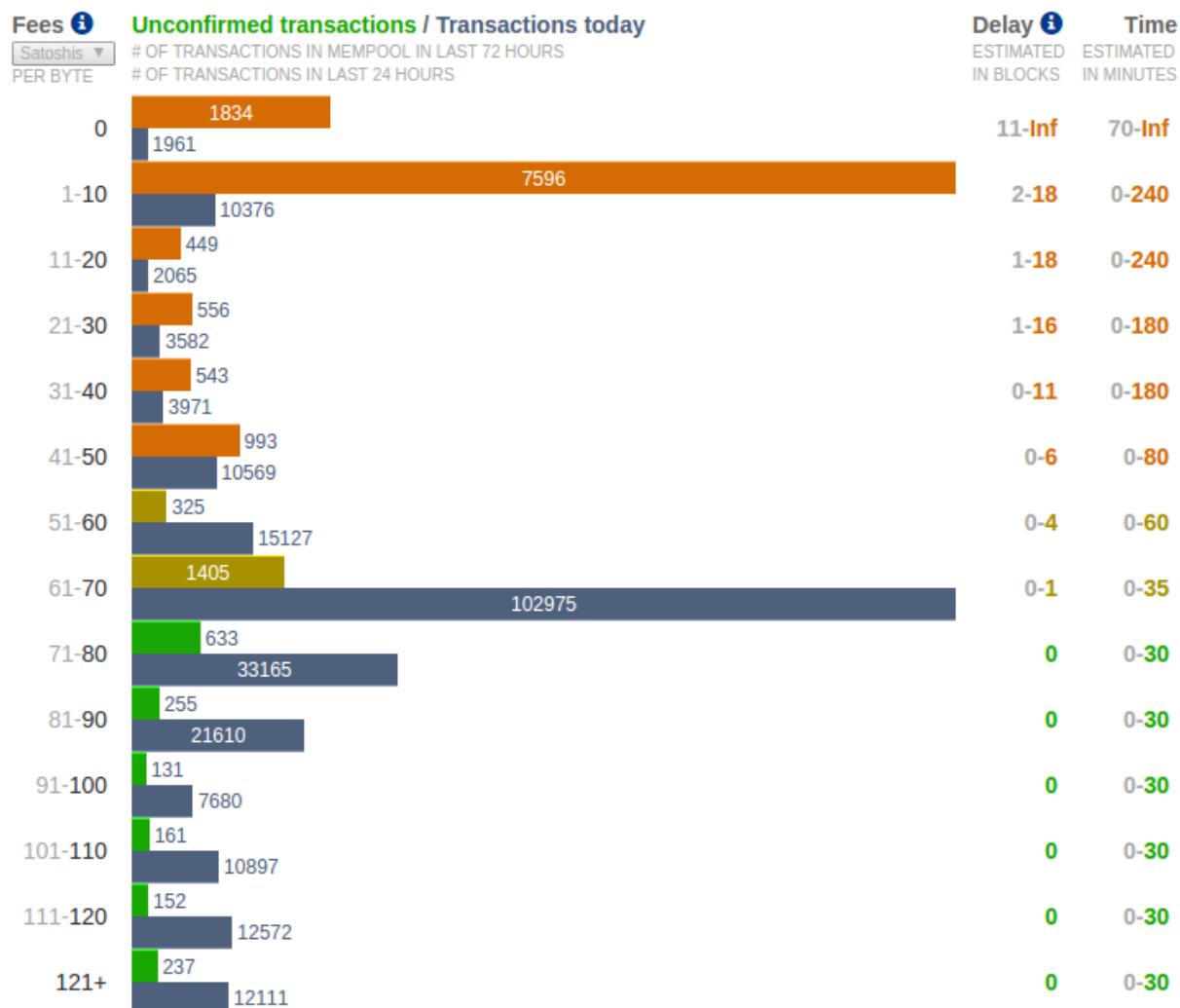


Figure 40. Serviciul de estimare a comisioanelor bitcoinfees.earn.com

## Adăugarea comisioanelor la tranzacții

Structura de date a tranzacțiilor nu are un câmp pentru comisioane. În schimb, comisioanele sunt considerate implicit ca fiind diferența între suma intrărilor și suma ieșirilor. Orice sumă în exces care rămâne după ce toate ieșirile au fost deduse din toate intrările este comisionul primit de mineri:

*Comisioanele de tranzacție sunt considerate implicit ca fiind excesul de la intrări minus ieșiri:*

$$\text{Comisioane} = \text{Sumă}(\text{Intrări}) - \text{Sumă}(\text{Ieșiri})$$

Acesta este un element oarecum confuz al tranzacțiilor și un punct important de înțeles, deoarece, dacă construiești tranzacțiile proprii, trebuie să vă asigurați că nu includeți, din neatenție, un comision foarte mare, subutilizând intrările. Asta înseamnă că trebuie să țineți cont de toate intrările, dacă este necesar prin crearea unui rest, sau veți ajunge să oferiți minerilor un bacșis foarte mare!

De exemplu, dacă consumați o UTXO de 20 de bitcoin pentru a efectua o plată de 1 bitcoin, trebuie să includeți o ieșire de 19 bitcoin în portofel. În caz contrar, "restul" de 19 bitcoin va fi contabilizat ca un comision de tranzacție și va fi încasat de minerul care va mina tranzacția dumneavoastră într-un bloc. Deși veți primi prioritate la procesare și veți face un miner foarte fericit, probabil că nu este ceea ce ați intenționat.

#### WARNING

Dacă uitați să adăugați o ieșire pentru rest într-o tranzacție construită manual, veți plăti restul ca un comision de tranzacție. "Păstrează restul!" s-ar putea să nu fie ceea ce intenționați.

Să vedem cum funcționează în practică, analizând din nou achiziția de cafea a lui Alice. Alice vrea să cheltuiască 0,015 bitcoin pentru a plăti cafeaua. Pentru a se asigura că această tranzacție este procesată prompt, va dori să includă o taxă de tranzacție, să zicem 0,001. Astă înseamnă că costul total al tranzacției va fi de 0,016. Prin urmare, portofelul ei trebuie să creeze un set de UTXO-uri care adaugă 0,016 bitcoin sau mai mult și, dacă este necesar, să creeze rest. Să spunem că portofelul ei are o UTXO de 0,2-bitcoin disponibil. Prin urmare, va trebui să consume acest UTXO, să creeze o ieșire la Cafeneaua lui Bob pentru 0,015 și o a doua ieșire cu 0,184 bitcoin ca rest propriului portofel, lăsând 0,001 bitcoin nealocat, ca un comision implicit pentru tranzacție.

Acum, să analizăm un scenariu diferit. Eugenia, directorul nostru de organizație de caritate pentru copii din Filipine, a finalizat o strângere de fonduri pentru achiziționarea de cărți școlare pentru copii. A primit câteva mii de donații mici de la oameni din întreaga lume, în valoare totală de 50 de bitcoin, astfel că portofelul ei este plin de plăți foarte mici (UTXO). Acum vrea să cumpere sute de cărți școlare de la o editură locală, plătind în bitcoin.

Deoarece aplicația portofel a Eugeniei încearcă să construiască o singură tranzacție de plată mai mare, ea trebuie să aleagă din setul UTXO disponibil, care este compus din mai multe sume mai mici. Aceasta înseamnă că tranzacția rezultată va alege din mai mult de o sută de UTXO cu valoare mică ca intrări și o singură ieșire, plătind editorul de carte. O tranzacție cu multe intrări va fi mai mare decât un kilo-octet, poate că mai mulți kilo-octeți. În consecință, va necesita un comision mult mai mare decât o tranzacție de dimensiune medie.

Aplicația portofel a Eugeniei va calcula comisionul corespunzător, măsurând dimensiunea tranzacției și înmulțind-o cu comisionul per kilo-octet. Multe portofele vor plăti comisioane pentru tranzacții mai mari, pentru a se asigura că tranzacția este procesată prompt. Comisionul mai mare nu se datorează faptului că Eugenia cheltuiește mai mulți bani, ci pentru că tranzacția ei este mai complexă și are dimensiuni mai mari - comisionul este independent de valoarea bitcoin a tranzacției.

## Scripturi de Tranzacție și Limbaj de Scriptare

Limbajul de scriptare pentru tranzacții bitcoin, numit *Script*, este un limbaj în notare poloneză inversă, asemănător cu Forth, bazat pe stivă de execuție. Dacă asta vi se pare o bolboroseală, probabil că nu ați studiat limbajele de programare din anii 1960, dar este în regulă - vom explica totul în acest capitol. Atât scriptul de blocare plasat pe un UTXO, cât și scriptul de deblocare sunt scrise în acest limbaj de scriptare. Când o tranzacție este validată, scriptul de deblocare din fiecare intrare este executat alături de scriptul de blocare corespunzător, pentru a vedea dacă satisfac condiția de cheltuire.

Script este un limbaj foarte simplu, care a fost conceput pentru a avea un domeniu limitat și să fie executabil pe o gamă largă de hardware, chiar la fel de simplu ca un dispozitiv încorporat. Necesită o prelucrare minimă și nu poate face multe dintre lucrurile extravagante pe care le pot face limbajele de programare moderne. Pentru utilizarea sa în validarea banilor programabili, aceasta este o caracteristică de securitate deliberată.

Astăzi, cele mai multe tranzacții procesate prin rețeaua bitcoin au forma "Plată către adresa bitcoin a lui Bob" și se bazează pe un script numit Plată-Către-Rezumat-Cheie-Publică (Pay-to-Public-Key-Hash). Cu toate acestea, tranzacțiile bitcoin nu sunt limitate la scripturi de forma "Plată către adresa de bitcoin a lui Bob". De fapt, scripturile de blocare pot fi scrise pentru a exprima o mare varietate de condiții complexe. Pentru a înțelege aceste scripturi mai complexe, trebuie mai întâi să înțelegem elementele de bază ale scripturilor de tranzacții și ale limbajului de scriptare.

În această secțiune, vom demonstra componentele de bază ale limbajului de scriptare folosit pentru tranzacții bitcoin și vom arăta cum poate fi utilizat pentru a exprima condiții simple de cheltuire și cum pot fi îndeplinite aceste condiții prin deblocarea scripturilor.

**TIP** Validarea tranzacțiilor Bitcoin nu se bazează pe un model static, ci se realizează prin executarea unui limbaj de scriptare. Acest limbaj permite exprimarea unei varietăți aproape infinite de condiții. Astfel, bitcoin capătă puterea de "bani programabili".

## Turing Incomplet

Limbajul de scriptare pentru tranzacții bitcoin conține mulți operatori, dar este limitat în mod deliberat într-un mod important - nu există bucle sau capabilități complexe de control al fluxului, altele decât controlul condițional al fluxului. Acest lucru asigură că limbajul nu este *Turing Complet*, ceea ce înseamnă că scripturile au o complexitate limitată și tempi de execuție previzibili. Scriptul nu este un limbaj cu scop general. Aceste limitări asigură faptul că limbajul nu poate fi folosit pentru a crea o buclă infinită sau o altă formă de "bombă logică" care ar putea fi încorporată într-o tranzacție într-un mod care provoacă un atac de tipul denial-of-service împotriva rețelei bitcoin. Nu uitați, fiecare tranzacție este validată de fiecare nod complet din rețeaua bitcoin. Un limbaj limitat împiedică utilizarea mecanismului de validare a tranzacțiilor ca o vulnerabilitate.

## Verificare fără Stare

Limbajul de scriptare pentru tranzacții bitcoin este fără stare (stateless), prin faptul că nu există nicio stare înainte de executarea scriptului sau stare salvată după executarea scriptului. Prin urmare, toate informațiile necesare pentru a executa un script sunt conținute în script. Un script va fi executat în mod previzibil în același mod pe orice sistem. Dacă sistemul dumneavoastră verifică un script, puteți fi sigur că orice alt sistem din rețeaua bitcoin va verifica și el scriptul, în sensul că o tranzacție validă este validă pentru toată lumea și toată lumea știe acest lucru. Această predictibilitate a rezultatelor este un beneficiu esențial al sistemului bitcoin.

## Construcția Scriptului (Blocare + Deblocare)

Motorul de validare a tranzacțiilor bitcoin se bazează pe două tipuri de scripturi pentru validarea tranzacțiilor: un script de blocare și un script de deblocare.

Un script de blocare este o condiție de cheltuire plasată pe o ieșire: specifică condițiile care trebuie îndeplinite pentru a cheltui ieșirea în viitor. Istoric, scriptul de blocare a fost numit *scriptPubKey*, deoarece de obicei conținea o cheie publică sau o adresă bitcoin (rezumat al cheii publice). În această carte ne referim la acesta ca la un "script de blocare" pentru a cuprinde gama mult mai largă de posibilități ale acestei tehnologii de scriptare. În majoritatea aplicațiilor bitcoin, la ceea ce ne referim ca un script de blocare va apărea în codul sursă ca *scriptPubKey*. Veți vedea, de asemenea, scriptul de blocare menționat drept *script martor* (vezi [Martor Segregat](#)) sau mai general ca un *puzzle criptografic*. Acești termeni înseamnă același lucru, la diferite niveluri de abstractizare.

Un script de deblocare este un script care "rezolvă" sau îndeplinește condițiile plasate pe o ieșire de un script de blocare și permite cheltuirea ieșirii. Scripturile de deblocare fac parte din fiecare intrare a tranzacției. De cele mai multe ori, acestea conțin o semnătură digitală produsă de portofelul utilizatorului folosind cheia sa privată. Istoric, scriptul de deblocare a fost numit *scriptSig*, deoarece de obicei conținea o semnătură digitală. În majoritatea aplicațiilor bitcoin, codul sursă se referă la scriptul de deblocare ca *scriptSig*. Veți vedea, de asemenea, scriptul de deblocare menționat drept *martor* (vezi [Martor Segregat](#)). În această carte, ne referim la acesta ca la un "script de deblocare" pentru a cuprinde gama mult mai largă de cerințe pentru scripturile de blocare, deoarece nu toate scripturile de deblocare trebuie să conțină semnături.

Fiecare nod de validare bitcoin va valida tranzacțiile executând scripturile de blocare și de blocare împreună. Fiecare intrare conține un script de deblocare și se referă la o UTXO existentă anterior. Programul de validare va copia scriptul de deblocare, va prelua UTXO-ul la care face referire intrarea și va copia scriptul de blocare din UTXO. Scripturile de deblocare și blocare sunt apoi executate în succesiune. Intrarea este validă dacă scriptul de deblocare satisface condițiile scriptului de blocare (vezi [Executarea separată a scripturilor de deblocare și blocare](#)). Toate intrările sunt validate independent, ca parte a validării generale a tranzacției.

Rețineți că UTXO-ul este permanent înregistrat în lanțul-de-blocuri (blockchain) și, prin urmare, este invariabil și nu este afectat de încercările eşuate de a-l cheltui atunci când este referențiat într-o nouă tranzacție. Doar o tranzacție validă care îndeplinește corect condițiile ieșirii va face ca ieșirea să fie considerată "cheltuită" și eliminată din setul de ieșiri de tranzacție necheltuite (setul UTXO).

[Combinarea scriptSig și scriptPubKey pentru a evalua un script de tranzacție](#) este un exemplu de scripturi de deblocare și de blocare pentru cel mai obișnuit tip de tranzacție bitcoin (o plată către un rezumat de cheie publică), care arată scriptul combinat rezultat din concatenarea scripturilor de deblocare și de blocare înainte de validarea scriptului.

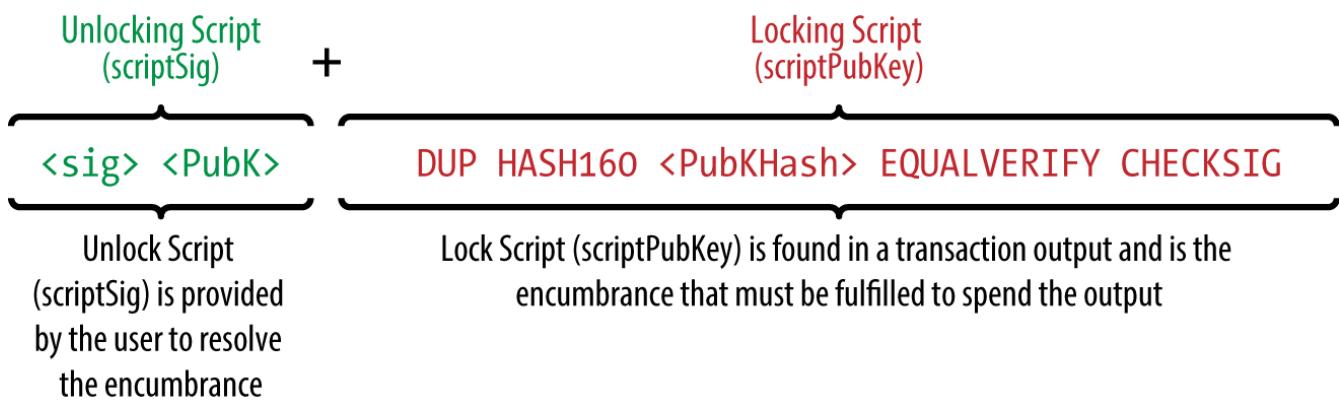


Figure 41. Combinarea scriptSig și scriptPubKey pentru a evalua un script de tranzacție

### Stiva de execuție a scriptului

Limbajul de scriptare al bitcoin este un limbaj bazat pe stivă, deoarece folosește o structură de date numită *stivă* (stack). O stivă este o structură de date foarte simplă care poate fi vizualizată ca o stivă de carduri. O stivă permite două operații: *push* și *pop*. Push adaugă un element în vârful stivei. Pop elimină elementul din vârful stivei. Operațiunile pe o stivă pot acționa numai asupra elementului cel mai de sus al stivei. O structură de date de tip stivă este de asemenea numită coadă Ultimul-Intrat-Primul-Ieșit (Last-In-First-Out) sau "LIFO".

Limbajul de scriptare execută scriptul procesând fiecare element de la stânga la dreapta. Numerele (constante) sunt împinse pe stivă. Operatorii împing (push) sau scot (pop) unul sau mai mulți parametri din stivă, acționează asupra lor și pot împinge (push) un rezultat înapoi pe stivă. De exemplu, **OP\_ADD** va scoate (pop) două elemente din stivă, le va aduna și va împinge (push) suma rezultată pe stivă.

Operatorii condiționali evaluează o condiție, producând un rezultat boolean de TRUE (adevărat) sau FALSE (fals). De exemplu, **OP\_EQUAL** scoate două elemente din stivă și împinge TRUE (reprezentat de numărul 1) dacă sunt egale sau FALSE (reprezentat de zero) dacă nu sunt egale. Scripturile de tranzacții bitcoin conțin de obicei un operator condițional, astfel încât acestea pot produce rezultatul TRUE care semnifică o tranzacție validă.

### Un script simplu

Acum să aplicăm ceea ce am învățat despre scripturi și stive în câteva exemple simple.

În [Scriptul de validare bitcoin facând calcule simple](#), scriptul **2 3 OP\_ADD 5 OP\_EQUAL** demonstrează operatorul de adăugare aritmetică **OP\_ADD**, adunând două numere și punând rezultatul pe stivă, urmat de operatorul condițional **OP\_EQUAL**, care verifică dacă suma rezultată este egală cu **5**. Pentru concizie, prefixul **OP\_** este omis în exemplul pas cu pas. Pentru mai multe detalii despre operatorii și funcțiile de script disponibile, consultați [Operatori, Constante și Simboluri ale Limbajului de Scriptare pentru Tranzacții](#).

Deși majoritatea scripturilor de blocare se referă la un rezumat (hash) de cheie publică (în esență, o adresă bitcoin), necesitând astfel o dovdă de proprietate pentru a cheltui fondurile, scriptul nu trebuie să fie atât de complex. Orice combinație de scripturi de blocare și deblocare care rezultă într-o valoare TRUE este validă. Aritmetica simplă pe care am folosit-o ca exemplu de limbaj de scriptare este, de asemenea, un script de blocare valid care poate fi folosit pentru a bloca o ieșire a tranzacției.

Utilizați o parte din scriptul de exemplu aritmetic ca script de blocare:

```
3 OP_ADD 5 OP_EQUAL
```

care poate fi satisfăcut de o tranzacție care conține o intrare cu scriptul de deblocare:

```
2
```

Programul de validare combină scripturile de blocare și deblocare, iar scriptul rezultat este:

```
2 3 OP_ADD 5 OP_EQUAL
```

După cum am văzut în exemplul pas cu pas din [Scriptul de validare bitcoin facând calcule simple](#), când acest script este executat, rezultatul este [OP\\_TRUE](#), ceea ce face tranzacția validă. Nu numai că este un script valid de blocare a ieșirii tranzacțiilor, dar UTXO-ul rezultat ar putea fi cheltuit de către oricine are abilități aritmetice pentru a ști că numărul 2 satisface scriptul.

**TIP**

Tranzacțiile sunt valide dacă rezultatul stivei este [TRUE](#) (notat ca [{0x01}](#)), orice altă valoare diferită de zero sau dacă stiva este goală după executarea scriptului. Tranzacțiile sunt invalide dacă valoarea stivei este [FALSE](#) (o valoare goală de lungime zero, notată ca [{}](#)) sau dacă execuția scriptului este oprită explicit de către un operator, cum ar fi [OP\\_VERIFY](#), [OP\\_RETURN](#) sau un terminator condițional, cum ar fi [OP\\_ENDIF](#). Vezi [Operatori, Constante și Simboluri ale Limbajului de Scriptare pentru Tranzacții](#) pentru detalii.

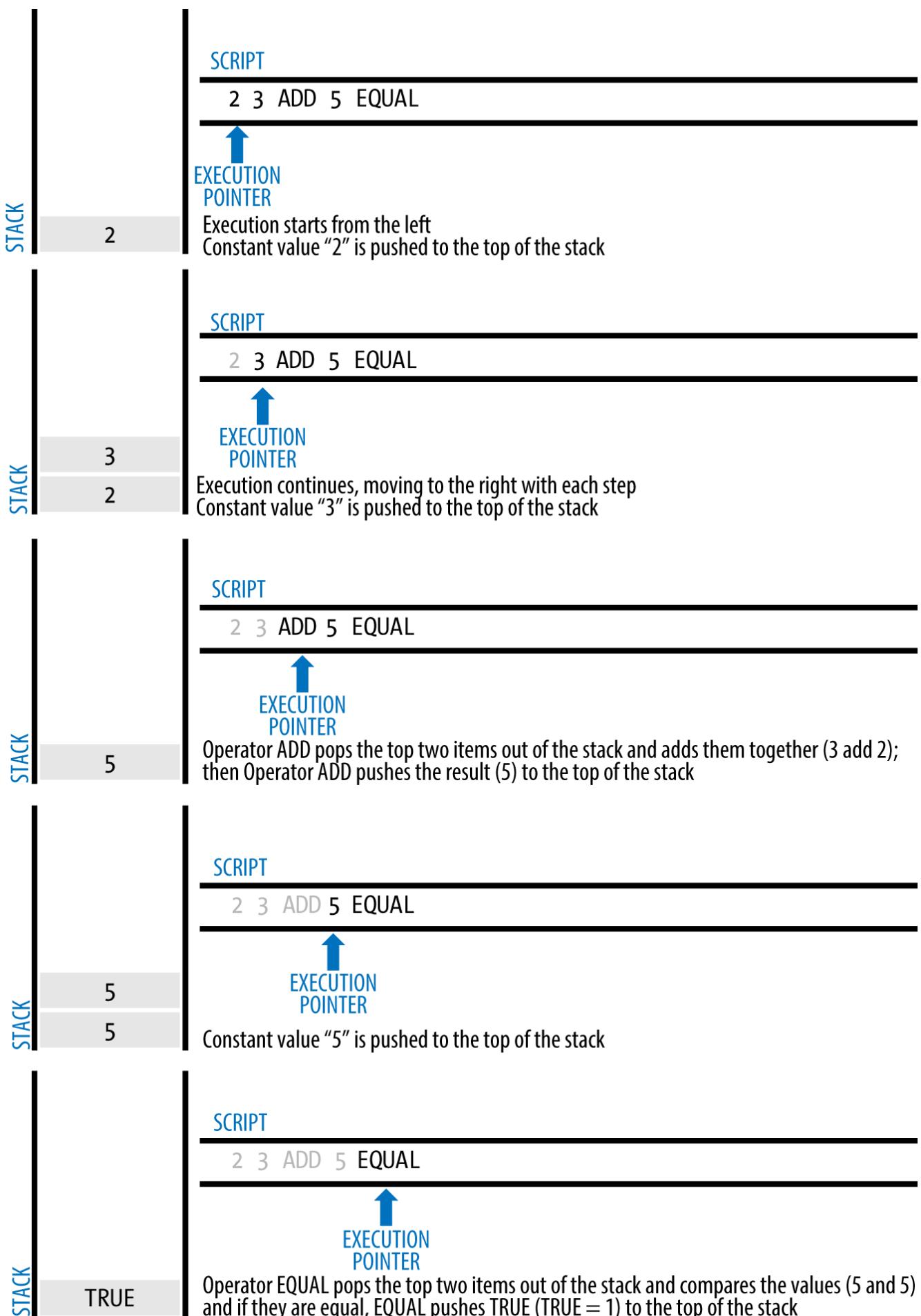


Figure 42. Scriptul de validare bitcoin facând calcule simple

Următorul este un script ceva mai complex, care calculează  $2 + 7 - 3 + 1$ . Observați că atunci când scriptul conține mai mulți operatori la rând, stiva permite ca rezultatele unui operator să fie folosite de următorul operator:

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

Încercați să validați singur scriptul precedent folosind un creion și hârtie. Când execuția scriptului se încheie, ar trebui să rămâneți cu valoarea **TRUE** pe stivă.

### Executarea separată a scripturilor de deblocare și blocare

În clientul original bitcoin, scripturile de deblocare și de blocare erau concatenate și executate în succesiune. Din motive de securitate, acest lucru a fost schimbat în 2010, din cauza unei vulnerabilități care permitea unui script de deblocare malformat să împingă datele pe stivă și să corupă scriptul de blocare. În implementarea curentă, scripturile sunt executate separat iar stiva este transferată între cele două execuții, aşa cum este descris în continuare.

Întâi, scriptul de deblocare este executat, folosind motorul de execuție al stivei. Dacă scriptul de deblocare este executat fără erori (de ex., nu au rămas operatori "suspendați"), stiva principală este copiată și scriptul de blocare este executat. Dacă rezultatul executării scriptului de blocare cu datele de stivă copiate din scriptul de deblocare este "TRUE", scriptul de deblocare a reușit să rezolve condițiile impuse de scriptul de blocare și, prin urmare, intrarea este o autorizație validă pentru a cheltui UTXO-ul. Dacă orice rezultat, în afară de "TRUE", rămâne după executarea scriptului combinat, intrarea nu este validă, deoarece nu a reușit să satisfacă condițiile de cheltuire prezente pe UTXO.

### Plată-către-Rezumat-Cheie-Publică (P2PKH)

Marea majoritate a tranzacțiilor procesate în rețeaua bitcoin cheltuiește ieșiri blocate cu un script Plată-către-Rezumat-Cheie-Publică (Pay-to-Public-Key-Hash) sau "P2PKH". Aceste ieșiri conțin un script de blocare care blochează ieșirea unui rezumat (hash) de cheie publică, cunoscut și ca adresă bitcoin. O ieșire blocată de un script P2PKH poate fi deblocată (cheltuită) prin prezentarea unei chei publice și a unei semnături digitale create de cheia privată corespunzătoare (vezi [Semnături Digitale \(ECDSA\)](#)).

De exemplu, să ne uităm din nou la plata lui Alice către Cafeneaua lui Bob. Alice a efectuat o plată de 0,015 bitcoin la adresa bitcoin a cafenelei. Această ieșire a tranzacției ar avea un script de blocare de forma:

```
OP_DUP OP_HASH160 <Cafe Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG
```

**Cafe Public Key Hash** (Rezumatul Cheii Publice al Cafenelei) este echivalent cu adresa bitcoin a cafenelei, fără codificarea Base58Check. Majoritatea aplicațiilor vor afișa *rezumatul cheii publice* în codificare hexazecimală și nu adresa familiară bitcoin în formatul Base58Check care începe cu un "1".

Scriptul de blocare precedent poate fi rezolvat cu un script de deblocare de forma:

<Cafe Signature> <Cafe Public Key>

Cele două scripturi împreună vor forma următorul script de validare combinat:

<Cafe Signature> <Cafe Public Key> OP\_DUP OP\_HASH160  
<Cafe Public Key Hash> OP\_EQUALVERIFY OP\_CHECKSIG

Când este executat, acest script combinat va fi evaluat la TRUE dacă și numai dacă, scriptul de deblocare se potrivește cu condițiile stabilite de scriptul de blocare. Cu alte cuvinte, rezultatul va fi TRUE dacă scriptul de deblocare are o semnătură validă din cheia privată a cafenelei, care corespunde cu rezumatul (hash-ul) cheii publice setat ca sarcină.

Imaginiile următoare arată (în două părți) o execuție pas cu pas a scriptului combinat, care va demonstra că este o tranzacție validă.

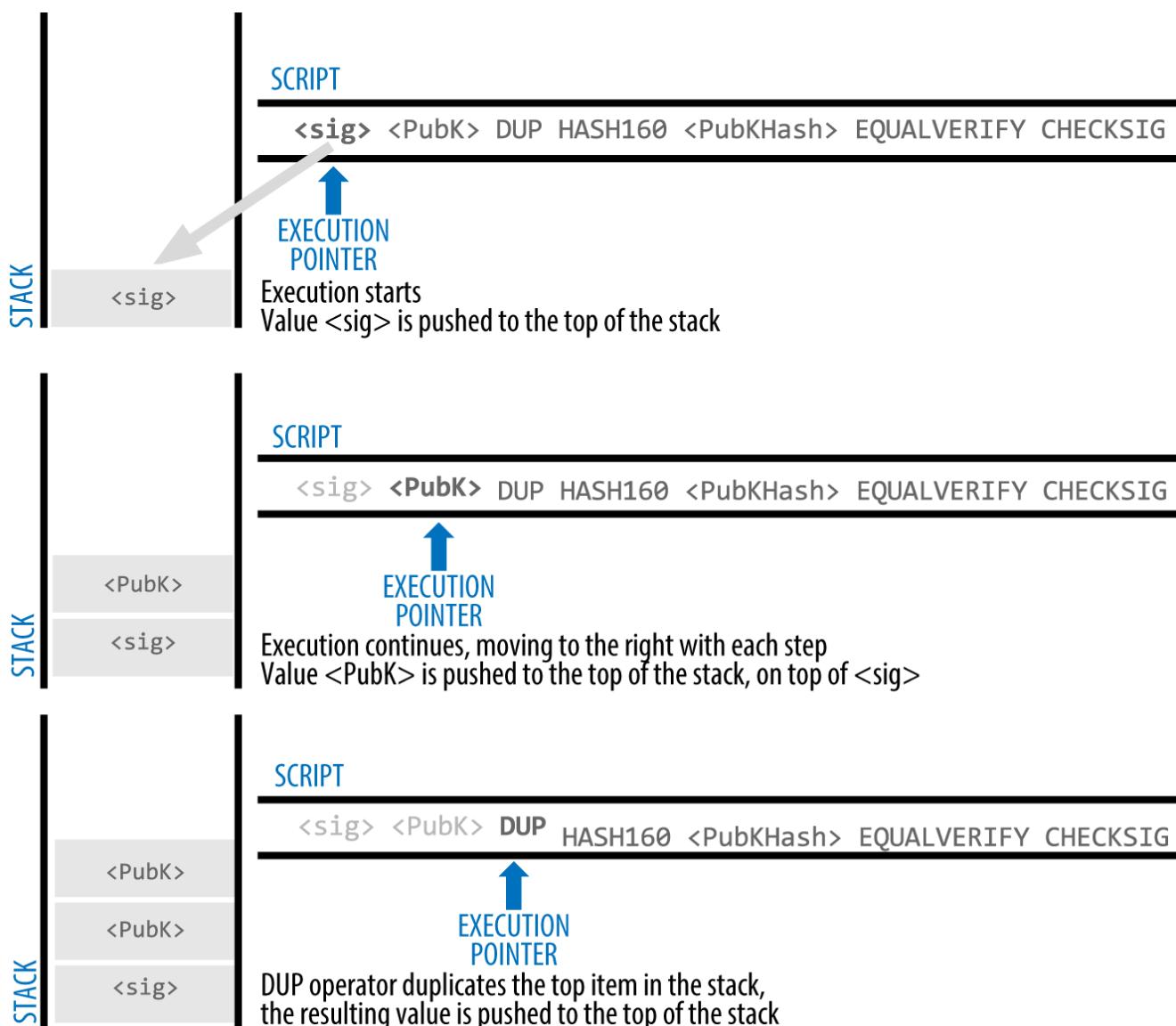


Figure 43. Evaluarea unui script pentru o tranzacție P2PKH (partea 1 din 2)

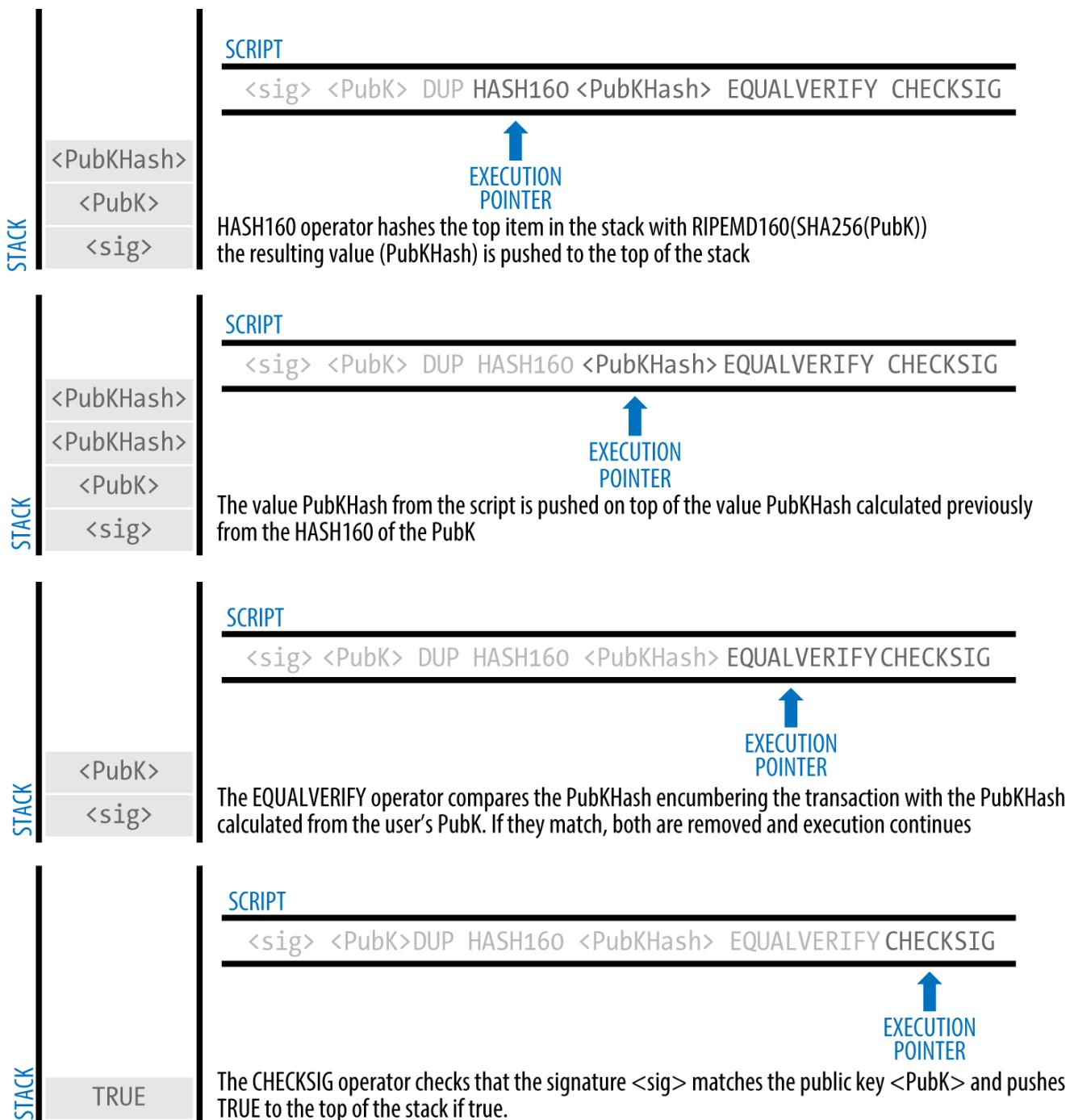


Figure 44. Evaluarea unui script pentru o tranzacție P2PKH (partea 2 din 2)

## Semnături Digitale (ECDSA)

Până în prezent, nu am aprofundat niciun detaliu legat de "semnături digitale". În această secțiune analizăm modul în care funcționează semnăturile digitale și cum pot prezenta dovada dreptului de proprietate asupra unei chei private, fără să dezvăluie acea cheie privată.

Algoritmul pentru semnătură digitală utilizat în bitcoin este Algoritmul cu Semnătură Digitală Curbă Eliptică (Elliptic Curve Digital Signature Algorithm), sau *ECDSA*. ECDSA este algoritmul utilizat pentru semnăturile digitale bazate pe perechi de cheie privată/publică ale curbei eliptice, așa cum este descris în [Criptografia Curbei Eliptice Explicată](#). ECDSA este folosit de funcțiile de scriptare `OP_CHECKSIG`, `OP_CHECKSIGVERIFY`, `OP_CHECKMULTISIG` și `OP_CHECKMULTISIGVERIFY`. De fiecare dată când vedeați una din funcțiile precedente într-un script de blocare, scriptul de deblocare

trebuie să conțină o semnătură ECDSA.

Semnătura digitală îndeplinește trei scopuri în bitcoin. În primul rând, semnătura dovedește că proprietarul cheii private, care este implicit proprietarul fondurilor, a autorizat cheltuirea fondurilor respective. În al doilea rând, dovada autorizării este *incontestabilă* (non-repudiere). În al treilea rând, semnătura dovedește că tranzacția (sau anumite părți ale tranzacției) nu au fost modificate și nu pot fi modificate de nimeni după ce a fost semnată.

Rețineți că fiecare intrare a tranzacției este semnată independent. Acest lucru este esențial, întrucât nici semnăturile, nici intrările nu trebuie să aparțină sau să fie aplicate de aceiași "proprietari". De fapt, o schemă de tranzacții specifică numită "CoinJoin" folosește acest fapt pentru a crea tranzacții multilaterale pentru confidențialitate.

#### NOTE

Fiecare intrare a tranzacției și orice semnătură pe care o poate conține este *complet* independentă față de orice altă intrare sau semnătură. Mai multe entități pot colabora pentru construirea tranzacțiilor prin semnarea unei singure intrări fiecare.

### Definiția Wikipedia pentru "Semnătură digitală"

Semnătura digitală este o schemă matematică pentru a demonstra autenticitatea unui mesaj sau document digital. O semnătură digitală validă oferă unui destinatar motive să credă că mesajul a fost creat de un expeditor cunoscut (autentificare), că expeditorul nu poate nega că a trimis mesajul (non-repudiere) și că mesajul nu a fost modificat în tranzit (integritate).

Sursă: [https://en.wikipedia.org/wiki/Digital\\_signature](https://en.wikipedia.org/wiki/Digital_signature)

## Cum Funcționează Semnăturile Digitale

Semnătura digitală este o *schemă matematică* care constă din două părți. Prima parte este un algoritm pentru crearea unei semnături, folosind o cheie privată (cheia de semnare), dintr-un mesaj (tranzacția). A doua parte este un algoritm care permite oricui să verifice semnătura, având mesajul și o cheie publică.

### Crearea unei semnături digitale

În implementarea bitcoin a algoritmului ECDSA, "mesajul" semnat este tranzacția sau, mai exact, un rezumat (hash) al unui subset specific de date din tranzacție (vezi [Tipuri de Rezumat \(Hash\) pentru Semnătură \(SIGHASH\)](#)). Cheia de semnare este cheia privată a utilizatorului. Rezultatul este semnătura:

$$(Sig = F_{sig}(F_{hash}(m), dA))$$

unde:

- $dA$  este cheia privată de semnare
- $m$  este tranzacția (sau părți ale acesteia)

- $F_{hash}$  este funcția de rezumare (hashing)
- $F_{sig}$  este algoritmul de semnare
- $Sig$  este semnătura rezultată

Mai multe detalii despre calculul ECDSA găsiți în [Calculul ECDSA](#).

Funcția  $F_{sig}$  produce o semnătură  $Sig$  care este compusă din două valori, denumite în mod obișnuit  $R$  și  $S$ :

$Sig = (R, S)$

Acum că au fost calculate cele două valori  $R$  și  $S$ , acestea sunt serializate într-un flux de octeți folosind o schemă de codificare standard internațională numită *Reguli de Codificare Distincte* (Distinguished Encoding Rules), sau *DER*.

### Serializarea semnăturilor (DER)

Să ne uităm din nou la tranzacția pe care Alice a creat-o. În intrarea tranzacției există un script de deblocare care conține următoarea semnătură codată DER creată de portofelul lui Alice:

```
3045022100884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb02204b9f039f
f08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e381301
```

Semnătura respectivă este un flux de octeți serializat al valorilor  $R$  și  $S$  produs de portofelul lui Alice pentru a dovedi că deține cheia privată autorizată să cheltuiască acea ieșire. Formatul de serializare constă din nouă elemente, după cum urmează:

- **0x30** - indicând începutul unei secvențe DER
- **0x45** - lungimea secvenței (69 octeți)
- **0x02** - urmează o valoare întreagă
- **0x21** - lungimea numărului întreg (33 octeți)
- **R** - `00884d142d86652a3f47ba4746ec719bbfb040a570b1deccbb6498c75c4ae24cb`
- **0x02** - urmează un alt număr întreg
- **0x20** - lungimea numărului întreg (32 octeți)
- **S** - `4b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813`
- Un sufix (**0x01**) care indică tipul de rezumat (hash) utilizat (**SIGHASH\_ALL**)

Vedeți dacă puteți decoda semnătura serializată (codată DER) a lui Alice folosind această listă. Numerele importante sunt  $R$  și  $S$ ; restul datelor fac parte din schema de codare DER.

### Verificarea Semnăturii

Pentru a verifica semnătura, trebuie să aveți semnătura ( $R$  și  $S$ ), tranzacția serializată și cheia publică (care corespunde cheii private utilizate pentru a crea semnătura). În esență, verificarea

unei semnături înseamnă "Doar proprietarul cheii private care a generat această cheie publică ar fi putut produce această semnătură pentru această tranzacție."

Algoritmul de verificare a semnături primește mesajul (un rezumat al tranzacției sau părți ale acesteia), cheia publică a semnatarului și semnătura (valorile  $R$  și  $S$ ) și returnează TRUE dacă semnătura este validă pentru acest mesaj și pentru această cheie publică.

## Tipuri de Rezumat (Hash) pentru Semnătură (SIGHASH)

Semnăturile digitale sunt aplicate mesajelor, care în cazul bitcoin sunt tranzacțiile în sine. Semnătura implică un *angajament* din partea semnatarului asupra datelor specifice ale tranzacției. În cea mai simplă formă, semnătura se aplică întregii tranzacții, implicând astfel toate intrările, ieșirile și celelalte câmpuri ale tranzacției. Cu toate acestea, o semnătură poate să se aplice doar la un subset de date dintr-o tranzacție, ceea ce este util pentru o serie de scenarii aşa cum vom vedea în această secțiune.

Semnăturile bitcoin au o modalitate de a indica ce parte a datelor unei tranzacții este inclusă în rezumatul (hash-ul) semnat de cheia privată folosind un indicator **SIGHASH**. Indicatorul **SIGHASH** este un singur octet care este anexat la semnătura. Fiecare semnătură are un indicator **SIGHASH**, iar indicatorul poate fi diferit de la intrare la intrare. O tranzacție cu trei intrări semnate poate avea trei semnături cu indicatori diferenți **SIGHASH**, fiecare semnătură semnând părți diferite ale tranzacției.

Nu uitați, fiecare intrare poate conține o semnătură în scriptul său de deblocare. Drept urmare, o tranzacție care conține mai multe intrări poate avea semnături cu diferenți indicatori **SIGHASH** care includ diferenți părți ale tranzacției în fiecare dintre intrări. Rețineți, de asemenea, că tranzacțiile bitcoin pot conține intrări de la diferenți "proprietari", care pot semna o singură intrare într-o tranzacție parțial construită (și invalidă), colaborând cu alții pentru a aduna toate semnăturile necesare pentru a realiza o tranzacție validă. Multe dintre tipurile de indicatori **SIGHASH** au sens numai dacă vă gândiți la mai mulți participanți care colaborează în afara rețelei bitcoin și actualizează o tranzacție parțial semnată.

Există trei indicatori **SIGHASH**: **ALL**, **NONE** și **SINGLE**, aşa cum se vede în [Tipuri de SIGHASH și semnificațiile lor](#).

Table 18. Tipuri de SIGHASH și semnificațiile lor

Indicator <b>SIGHASH</b>	Valoare	Descriere
<b>ALL</b>	0x01	Semnătura se aplică tuturor intrărilor și ieșirilor
<b>NONE</b>	0x02	Semnătura se aplică tuturor intrărilor și nici unei ieșiri
<b>SINGLE</b>	0x03	Semnătura se aplică tuturor intrărilor, dar numai ieșirii cu același număr de index cu intrarea semnată

În plus, există un indicator modifier **SIGHASH\_ANYONECANPAY**, care poate fi combinat cu fiecare din

indicatorii precedenți. Când **ANYONECANPAY** este setat, o singură intrare este semnată, lăsând restul (și numărul lor de secvență) deschise pentru modificare. **ANYONECANPAY** are valoarea **0x80** și se aplică folosind operatorul OR pe biți, rezultând indicatoarele combinate, așa cum se vede în [Tipurile SIGHASH cu modificatori și semnificațiile acestora](#).

Table 19. Tipurile SIGHASH cu modificatori și semnificațiile acestora

Indicator SIGHASH	Valoare	Descriere
ALL   ANYONECANPAY	0x81	Semnatura se aplică unei intrări și tuturor ieșirilor
NONE   ANYONECANPAY	0x82	Semnatura se aplică unei intrări și niciuna dintre ieșiri
SINGLE   ANYONECANPAY	0x83	Semnatura se aplică unei intrări și ieșirii cu același număr de index

ACESTE COMBINAȚII DE INDICATORI SUNT REZUMATE ÎN [REZUMATUL DIFERITELOR COMBINAȚII DE SIGHASH](#).

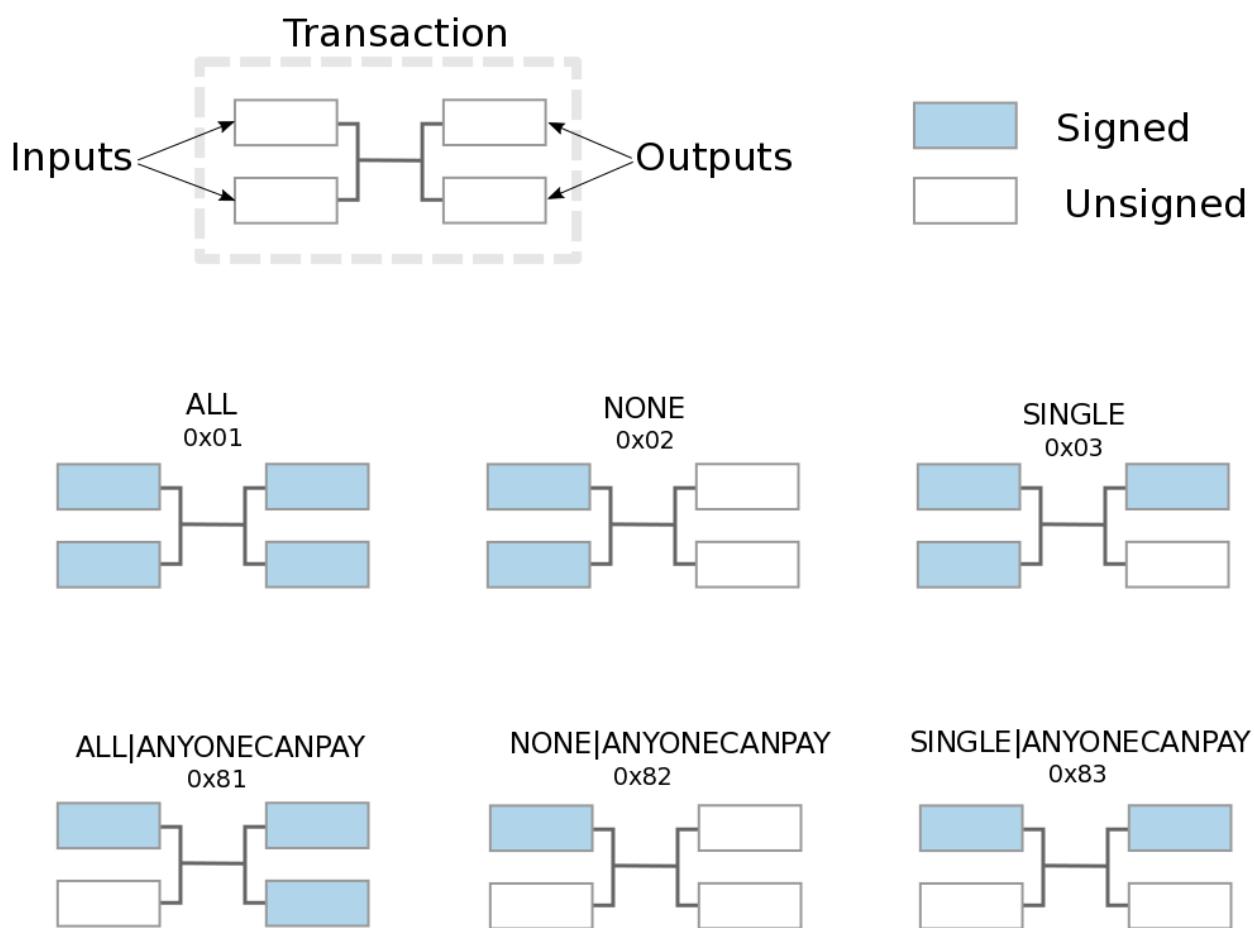


Figure 45. Rezumatul diferitelor combinații de SIGHASH

Procedeul prin care indicatorii **SIGHASH** sunt aplicăți în timpul semnării și verificării este că se face o copie a tranzacției și anumite câmpuri din interior sunt trunchiate (setate la lungimea zero și golite). Tranzacția rezultată este serializată. Indicatorul **SIGHASH** este adăugat la sfârșitul tranzacției serializate și rezultatul este rezumat (hashed). Rezumatul (hash-ul) în sine este "mesajul" care este semnat. În funcție de care indicator **SIGHASH** este folosit, diferite părți ale tranzacției sunt

trunchiate. Rezumatul rezultat depinde de diferite subseturi ale datelor din tranzacție. Prin includerea **SIGHASH** ca ultimul pas înainte de rezumare (hashing), semnătura include și tipul **SIGHASH**, deci nu poate fi modificată (de exemplu, de către un miner).

**NOTE** Toate tipurile **SIGHASH** semnează câmpul **nLocktime** al tranzacției (vezi [Timp de Blocare a Tranzacției \(nLocktime\)](#)). În plus, tipul **SIGHASH** este atașat la tranzacție înainte de a fi semnat, astfel încât nu poate fi modificat o dată ce a fost semnat.

În exemplul tranzacției lui Alice (consultați lista din [Serializarea semnăturilor \(DER\)](#)), am văzut că ultima parte a semnăturii codate DER a fost **01**, care este indicatorul **SIGHASH\_ALL**. Aceasta blochează datele tranzacției, astfel încât semnătura lui Alice include starea tuturor intrărilor și ieșirilor. Aceasta este cea mai comună formă de semnătură.

Să ne uităm la unele dintre celelalte tipuri **SIGHASH** și cum pot fi utilizate în practică:

#### **ALL | ANYONECANPAY**

Această construcție poate fi folosită pentru a realiza o tranzacție de tip "crowdfunding". Cineva care încearcă să strângă fonduri poate construi o tranzacție cu o singură ieșire. Unica ieșire plătește suma "țintă" către cei care se ocupă de strângerea fondurilor. O astfel de tranzacție nu este în mod evident validă, întrucât nu are intrări. Cu toate acestea, alții o pot modifica acum adăugând un aport propriu, ca donație. Ei își semnează propria intrare cu **ALL | ANYONECANPAY**. Dacă nu sunt adunate suficiente intrări pentru a atinge valoarea ieșirii, tranzacția nu este validă. Fiecare donație este un "gaj", care nu poate fi colectat de către cei care strâng fondurile până când nu se colectează întreaga sumă țintită.

#### **NONE**

Această construcție poate fi utilizată pentru a crea o "verificare la purtător" sau "cec în alb" al unei sume specificate. Se creează un angajament pentru intrare, dar permite schimbarea scriptului de blocare a ieșirii. Oricine își poate introduce propria adresă bitcoin în scriptul de blocare a ieșirilor și poate răscumpăra tranzacția. Cu toate acestea, valoarea de ieșire în sine este blocată prin semnătură.

#### **NONE | ANYONECANPAY**

Această construcție poate fi folosită pentru a construi un "colector de praf (măruntiș)". Utilizatorii care au UTXO-uri minusculă în portofele nu le pot cheltui fără costuri care depășesc valoarea prafului. Cu acest tip de semnătura, praful UTXO poate fi donat pentru ca oricine dorește, să îl poată agenda și să îl cheltuiască oricând dorește.

Există câteva propuneri de modificare sau extindere a sistemului **SIGHASH**. O astfel de propunere este *Bitmask Sighash Modes* de Glenn Willen de la Blockstream, ca parte a proiectului Elements. Aceasta își propune să creeze o înlocuire flexibilă pentru tipurile **SIGHASH** care să permită o "mască pe biți pentru intrări și ieșiri redactabilă arbitrar de către mineri" care poate exprima "scheme contractuale de pre-angajament mai complexe, cum ar fi ofertele semnate cu rest într-o bursă distribuită de active".

**NOTE**

Nu veți vedea indicatoarele **SIGHASH** prezentate ca opțiune în aplicația portofel a utilizatorului. Cu câteva excepții, portofelele construiesc scripturi P2PKH și semnează cu indicatorul **SIGHASH\_ALL**. Pentru a utiliza un alt indicator **SIGHASH**, ar trebui să scrieți software pentru a construi și semna tranzacții. Mai important, indicatoarele **SIGHASH** pot fi utilizate de aplicațiile-dedicate bitcoin care să permită utilizări noi.

## Calculul ECDSA

După cum am menționat anterior, semnăturile sunt create de o funcție matematică  $F_{sig}$  care produce o semnătură compusă din două valori  $R$  și  $S$ . În această secțiune analizăm funcția  $F_{sig}$  mai în detaliu.

Algoritmul de semnătură generează mai întâi o pereche de chei private/publice *efemere* (temporare). Această pereche de chei temporare este utilizată la calculul valorilor  $R$  și  $S$ , după o transformare care implică semnarea cheii private și rezumatului (hash-ul) tranzacției.

Perechea de chei temporare se bazează pe un număr aleatoriu  $k$ , care este utilizat ca și cheie privată temporară. Din  $k$ , generăm cheia publică temporară corespunzătoare  $P$  (calculată ca  $P = k*G$ , în același mod sunt derivate cheile publice bitcoin; vezi [Chei Publice](#)). Valoarea  $R$  a semnăturii digitale este coordonata x a cheii publice efemere  $P$ .

De acolo, algoritmul calculează valoarea  $S$  a semnăturii, astfel încât:

$$S = k^{-1} (Hash(m) + dA * R) \bmod n$$

unde:

- $k$  este cheia privată efemera
- $R$  este coordonata x a cheii publice efemere
- $dA$  este cheia privată de semnare
- $m$  sunt datele tranzacției
- $n$  este ordinea primă a curbei eliptice

Verificarea este inversa funcției de generare a semnăturilor, folosind valorile  $R$ ,  $S$  și cheia publică pentru a calcula o valoare  $P$ , care este un punct de pe curba eliptică (cheia publică efemera folosită la crearea semnăturilor):

$$P = S^{-1} * Hash(m) * G + S^{-1} * R * Qa$$

unde:

- $R$  și  $S$  sunt valorile semnăturii
- $Qa$  este cheia publică a lui Alice
- $m$  sunt datele tranzacției care a fost semnată
- $G$  este punctul generator de curbă eliptică

Dacă coordonata  $x$  a punctului calculat  $P$  este egală cu  $R$ , atunci verificatorul poate concluziona că semnătura este validă.

Rețineți că la verificarea semnăturii, cheia privată nu este nici cunoscută, nici dezvăluită.

**TIP**

ECDSA este în mod necesar o piesă de matematică destul de complicată; o explicație completă este dincolo de scopul acestei cărți. Câteva ghiduri excelente online vă ajută să o parcurgeți pas cu pas: căutați "ECDSA explained" sau încercați acesta: <http://bit.ly/2r0HhGB>.

## Importanța Aleatorului în Semnături

După cum am văzut în [Calculul ECDSA](#), algoritmul de generare a semnăturii folosește o cheie aleatorie  $k$ , ca bază pentru o pereche de chei private/publice efemere. Valoarea lui  $k$  nu este importantă, atât timp cât este aleatorie. Dacă aceeași valoare  $k$  este utilizată pentru a produce două semnături pe mesaje diferite (tranzacții), atunci *cheia privată* folosită pentru semnat poate fi calculată de oricine. Reutilizarea aceleiași valori pentru  $k$  într-un algoritm de semnătură duce la expunerea cheii private!

**WARNING**

Dacă aceeași valoare  $k$  este folosită în algoritmul de semnare pentru două tranzacții diferite, cheia privată poate fi calculată și expusă lumii!

Aceasta nu este doar o posibilitate teoretică. Am văzut că această problemă duce la expunerea cheilor private în câteva implementări diferite ale algoritmilor de semnare a tranzacțiilor în bitcoin. Oamenii au avut fonduri furate din cauza reutilizării inadvertente a unei valori  $k$ . Cel mai frecvent motiv pentru refolosirea unei valori  $k$  este un generator de numere aleatoare inițializat necorespunzător.

Pentru a evita această vulnerabilitate, cele mai bune practici din industrie recomandă să nu se genereze  $k$  cu un generator cu număr aleator care folosește ca sămânță (seed) entropie, ci să folosească în schimb un proces determinist-aleator care folosește ca sămânță (seed) datele tranzacției în sine. Acest lucru asigură că fiecare tranzacție produce un  $k$  diferit. Algoritmul standard al industriei pentru inițializarea deterministă a lui  $k$  este definit în [RFC 6979](#), publicat de Internet Engineering Task Force.

Dacă implementați un algoritm pentru a semna tranzacții în bitcoin, trebuie să utilizați RFC 6979 sau un algoritm similar-deterministic aleatoriu pentru a vă asigura că generați un  $k$  diferit pentru fiecare tranzacție.

## Adrese Bitcoin, Solduri și alte Abstractizări

Am început acest capitol cu descoperirea că tranzacțiile arată foarte diferit "în culise" decât modul în care sunt prezentate în portofele, de exploratorii de lanț-de-blocuri (blockchain explorers), și de alte aplicații orientate către utilizator. Multe dintre concepții simpliste și familiare din capitolele anterioare, cum ar fi adresele și soldurile bitcoin, par să fie absente din structura tranzacției. Am văzut că tranzacțiile nu conțin adrese bitcoin, în sine, ci operează prin scripturi care blochează și deblochează valori discrete ale bitcoin. Soldurile nu sunt prezente nicăieri în acest sistem și, cu toate acestea, fiecare aplicație portofel afișează în mod evident soldul portofelului utilizatorului.

Acum că am explorat ceea ce este de fapt inclus într-o tranzacție bitcoin, putem examina modul în care abstracțiile de nivel înalt sunt derivate din componentele aparent primitive ale tranzacției.

Să ne uităm din nou la modul în care tranzacția lui Alice a fost prezentată pe un explorator de blocuri popular ([Tranzacția lui Alice către Cafeneaua lui Bob](#)).

## Transaction View information about a bitcoin transaction

<a href="#">0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2</a>	
<a href="#">1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK</a> (0.1 BTC - <a href="#">Output</a> )	 <a href="#">1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA</a> - <a href="#">(Unspent)</a> 0.015 BTC <a href="#">1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK</a> - <a href="#">(Unspent)</a> 0.0845 BTC
	<a href="#">97 Confirmations</a> <span style="background-color: green; border: 1px solid black; padding: 2px 5px; color: white;">0.0995 BTC</span>
<strong>Summary</strong>	<strong>Inputs and Outputs</strong>
Size <span>258 (bytes)</span>	Total Input <span>0.1 BTC</span>
Received Time <span>2013-12-27 23:03:05</span>	Total Output <span>0.0995 BTC</span>
Included In Blocks <span>277316 (2013-12-27 23:11:54 +9 minutes)</span>	Fees <span>0.0005 BTC</span>
	Estimated BTC Transacted <span>0.015 BTC</span>

Figure 46. Tranzacția lui Alice către Cafeneaua lui Bob

În partea stângă a tranzacției, exploratorul lanțului-de-blocuri arată adresa bitcoin a lui Alice ca "expeditor". De fapt, aceste informații nu se regăsesc în tranzacția în sine. Când exploratorul lanțului-de-blocuri face referire la tranzacție, acesta face referire și la tranzacția anterioară asociată cu intrarea și a extras prima ieșire din tranzacția mai veche. În cadrul acestei ieșiri este un script de blocare care blochează UTXO-ul la rezumatul (hash-ul) cheii publice a lui Alice (un script P2PKH). Exploratorul lanțului-de-blocuri a extras rezumatul cheii publice și l-a codat folosind codificarea Base58Check pentru a produce și afișa adresa bitcoin care reprezintă acea cheie publică.

În mod similar, în partea dreaptă, exploratorul lanțului-de-blocuri arată cele două ieșiri; prima la adresa bitcoin a lui Bob și a doua la adresa bitcoin a lui Alice (ca rest). Încă o dată, pentru a crea aceste adrese bitcoin, exploratorul lanțului-de-blocuri a extras scriptul de blocare din fiecare ieșire, l-a recunoscut ca un script P2PKH și a extras rezumatul cheii publice din interior. În cele din urmă, exploratorul lanțului-de-blocuri a codificat acel rezumat al cheii publice cu Base58Check pentru a produce și afișa adresele bitcoin.

Dacă ar fi să dați clic pe adresa bitcoin a lui Bob, exploratorul lanțului-de-blocuri v-ar arăta imaginea din [Balanța adresei bitcoin a lui Bob](#).

## Bitcoin Address

Addresses are identifiers which you use to send bitcoins to another person.

Summary		Transactions	
Address	1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA	No. Transactions	25
Hash 160	ab68025513c3dbd2f7b92a94e0581f5d50f654e7	Total Received	0.17579525 BTC
Tools	<a href="#">Taint Analysis</a> - <a href="#">Related Tags</a> - <a href="#">Unspent Outputs</a>	Final Balance	0.17579525 BTC

Figure 47. Balanța adresei bitcoin a lui Bob

Exploratorul lanțului-de-blocuri afișează balanța adresei bitcoin a lui Bob. Dar nicăieri în sistemul bitcoin nu există un concept de "balanță". Mai degrabă, valorile afișate aici sunt construite de exploratorul lanțului-de-blocuri după cum urmează.

Pentru a construi suma "Total Primit" (Total Received), exploratorul lanțului-de-blocuri va decoda mai întâi codificarea Base58Check a adresei bitcoin pentru a prelua rezumatul (hash-ul) de 160 de biți a cheii publice a lui Bob. Apoi, exploratorul lanțului-de-blocuri va inspecta baza de date a tranzacțiilor, căutând ieșiri cu scripturi de blocare P2PKH care conțin rezumatul (hash-ul) cheii publice a lui Bob. Prin însumarea valorii tuturor rezultatelor, exploratorul lanțului-de-blocuri poate produce valoarea totală primită.

Construirea balanței curente (Final Balance) necesită ceva mai multă muncă. Exploratorul lanțului-de-blocuri păstrează o bază de date separată a ieșirilor care nu sunt cheltuite în prezent, setul UTXO. Pentru a menține această bază de date, exploratorul blockchain trebuie să monitorizeze rețeaua bitcoin, să adauge UTXO-uri nou create și să eliminate UTXO-uri cheltuite, în timp real, în timp ce apar în tranzacții neconfirmate. Aceasta este un proces complicat, care depinde de ținerea evidenței tranzacțiilor pe măsură ce acestea se propagă, precum și de a menține un consens cu rețeaua bitcoin pentru a se asigura că urmează lanțul corect. Uneori, exploratorul lanțului-de-blocuriiese din sincronizare și perspectiva sa asupra setului UTXO este incompletă sau incorectă.

Din setul UTXO, exploratorul lanțului-de-blocuri însumează valoarea tuturor ieșirilor necheltuite care fac referire la rezumatul (hash-ul) cheii publice a lui Bob și produce numărul "Balanță Finală" (Final Balance) afișat utilizatorului.

Pentru a produce această imagine, cu aceste două "balanțe", exploratorul lanțului-de-blocuri trebuie să indexeze și să caute prin zeci, sute sau chiar sute de mii de tranzacții.

În concluzie, informațiile prezentate utilizatorilor prin intermediul aplicațiilor portofel, exploratorilor lanțului-de-blocuri și alte interfețe cu utilizatorul sunt adesea compuse din abstractizări de nivel înalt care sunt obținute prin căutarea multor tranzacții diferite, inspecția conținutului lor și manipularea datelor acestora. Prin prezentarea acestei concepții simpliste asupra tranzacțiilor bitcoin care seamănă cu cecuri bancare de la un expeditor la un destinatar, aceste aplicații trebuie să rezume o mulțime de detalii adiacente. Ele se concentrează mai ales pe tipurile comune de tranzacții: P2PKH cu semnături SIGHASH\_ALL pe fiecare intrare. Astfel, în timp ce aplicațiile bitcoin pot prezenta mai mult de 80% din toate tranzacțiile într-o manieră ușor de citit, ele se împiedică uneori de tranzacții care se abat de la normă. Tranzacțiile care conțin scripturi de blocare mai complexe sau diferite indicatoare SIGHASH, sau multe intrări și ieșiri, demonstrează simplitatea și slăbiciunea acestor abstractizări.

În fiecare zi, sute de tranzacții care nu conțin ieșiri P2PKH sunt confirmate pe lanțul-de-blocuri. Exploratorii lanțului-de-blocuri le prezintă adesea cu mesaje de avertizare spunând că nu pot decoda o adresă.

După cum vom vedea în capitolul următor, acestea nu sunt neapărat tranzacții ciudate. Sunt tranzacții care conțin scripturi de blocare mai complexe decât P2PKH-ul obișnuit. Vom învăța cum să decodăm și să înțelegem scripturi mai complexe și aplicațiile pe care le susțin.

# Tranzacții Avansate și Scripturi

## Introducere

În capitolul anterior, am introdus elementele de bază ale tranzacțiilor bitcoin și am analizat cel mai comun tip de script de tranzacții, scriptul P2PKH. În acest capitol vom analiza scripturile mai avansate și cum le putem folosi pentru a construi tranzacții cu condiții complexe.

În primul rând, vom analiza scripturile *multisemnătură* (multisignature). În continuare, vom examina al doilea cel mai întâlnit script de tranzacție, Plată-către-Rezumat-Script (Pay-to-Script-Hash), care deschide un întreg univers de scripturi complexe. Apoi, vom examina noi operatori de script care adaugă o dimensiune legată de timp la bitcoin, prin *timpi de blocare* (timelocks). În cele din urmă, vom analiza *Martor Segregat* (Segregated Witness), o schimbare arhitecturală a structurii tranzacțiilor.

## Multisemnătură

Scripturile cu mai multe semnături stabilesc o condiție în care N chei publice sunt înregistrate în script și cel puțin M dintre acestea trebuie să furnizeze semnături pentru deblocarea fondurilor. Aceasta este cunoscută și sub denumirea de schemă M-din-N, unde N este numărul total de chei și M este pragul de semnături necesar pentru validare. De exemplu, o multisemnătură 2 din 3 este una în care sunt identificate trei chei publice ca potențiali semnatari și cel puțin două dintre acestea trebuie utilizate pentru a crea semnături pentru o tranzacție validă pentru a cheltui fondurile.

În acest moment, scripturile multisemnătură *standard* sunt limitate la cel mult 3 chei publice, ceea ce înseamnă că puteți face orice, de la 1-din-1 la 3-din-3 multisemnături sau orice combinație din acel interval. Limitarea la 3 chei ar putea fi ridicată la momentul publicării acestei cărți, așa că verificați funcția `IsStandard()` pentru a vedea ce este acceptat în prezent de rețea. Rețineți că limita a 3 chei se aplică numai scripturilor multisemnătură standard (cunoscute și sub denumirea de "simple"), nu și scripturilor multisemnătură "învelite" (wrapped) într-un script Plată-către-Rezumat-Script (P2SH). Scripturile multisemnătură P2SH sunt limitate la 15 chei, permitând o multisemnătură de până la 15 din 15. Vom învăța despre P2SH în [Plată-către-Rezumat-Script \(Pay-to-Script-Hash - P2SH\)](#).

Forma generală a unui script de blocare care setează o condiție multisemnătură M-din-N este:

```
M <Public Key 1> <Public Key 2> ... <Public Key N> N CHECKMULTISIG
```

unde  $N$  este numărul total de chei publice listate și  $M$  este pragul semnăturilor necesare pentru a cheltui ieșirile.

Un script de blocare care setează o condiție multisemnătură 2-din-3 arată astfel:

```
2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

Scriptul de blocare precedent poate fi satisfăcut cu un script de deblocare care conține perechi de semnături și chei publice:

```
<Signature B> <Signature C>
```

sau orice combinație de două semnături create de cheile private corespunzătoare celor trei chei publice enumerate.

Cele două scripturi împreună ar forma un script de validare combinat:

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3  
CHECKMULTISIG
```

Când este executat, acest script combinat va fi evaluat la TRUE (adevărat) dacă și numai dacă, scriptul de deblocare corespunde condițiilor stabilite de scriptul de blocare. În acest caz, condiția este dacă scriptul de deblocare are o semnătură validă de la cele două chei private care corespund la două dintre cele trei chei publice setate ca restricții.

## Un defect în execuția CHECKMULTISIG

Există un defect în execuția **CHECKMULTISIG** care necesită o soluție ușoară. Când **CHECKMULTISIG** se execută, ar trebui să consume  $M+N+2$  elemente de pe stivă ca parametri. Cu toate acestea, din cauza erorii, **CHECKMULTISIG** va scoate o valoare suplimentară sau o valoare mai mult decât se așteaptă.

Să analizăm mai în detaliu acest lucru folosind exemplul de validare anterior:

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3  
CHECKMULTISIG
```

În primul rând, **CHECKMULTISIG** scoate elementul din vârf, care este  $N$  (în acest exemplu "3"). Apoi scoate  $N$  elemente, care sunt cheile publice care pot fi semnate. În acest exemplu, cheile publice A, B și C. Apoi, scoate un element, care este  $M$ , cvorumul (câte semnături sunt necesare). Aici  $M = 2$ . În acest moment, **CHECKMULTISIG** ar trebui să scoată ultimele  $M$  elemente, care sunt semnăturile și să vadă dacă sunt valide. Cu toate acestea, din păcate, o eroare de implementare face ca **CHECKMULTISIG** să scoată mai mult cu un element (în total  $M+1$ ) decât ar trebui. Elementul suplimentar nu este luat în considerare atunci când sunt verificate semnăturile, astfel încât nu are efect direct asupra **CHECKMULTISIG**. Cu toate acestea, o valoare suplimentară trebuie să fie prezentă deoarece, dacă nu este prezentă, când **CHECKMULTISIG** încearcă să scoată de pe o stivă goală, va provoca o eroare de

stivă și o eroare a scriptului (marcarea tranzacției ca fiind invalidă). Deoarece elementul suplimentar este ignorat, acesta poate fi orice, dar se folosește în mod obișnuit **0**.

Deoarece această eroare a devenit parte a regulilor de consens, acum ea trebuie replicată pentru totdeauna. Prin urmare, validarea corectă a scriptului ar arăta astfel:

```
0 <Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3  
CHECKMULTISIG
```

Astfel, scriptul de deblocare folosit de fapt de multisemnătură nu este:

```
<Signature B> <Signature C>
```

dar în schimb este:

```
0 <Signature B> <Signature C>
```

De acum înainte, dacă vedeți un script de deblocare multisemnătură, trebuie să vă așteptați să vedeți un **0** suplimentar la început, al cărui singur scop este să repare o eroare care a devenit accidental o regulă de consens.

## Plată-către-Rezumat-Script (Pay-to-Script-Hash - P2SH)

Plata-către-Rezumat-Script (P2SH) a fost introdus în 2012 ca un nou tip puternic de tranzacție care simplifică mult utilizarea scripturilor de tranzacții complexe. Pentru a explica nevoia de P2SH, să analizăm un exemplu practic.

În [Introducere](#) l-am prezentat pe Mohammed, un importator de produse electronice cu sediul în Dubai. Compania lui Mohammed folosește intens funcționalitatea multisemnătură bitcoin pentru conturile sale corporative. Scripturile multisemnătură sunt una dintre cele mai frecvente utilizări ale capacitaților avansate de scriptare bitcoin și sunt o funcționalitate foarte puternică. Compania lui Mohammed folosește un script multisemnatură pentru toate plățile clienților, cunoscute în termeni contabili drept "conturi de încasări". Cu schema multisemnătură, orice plăți efectuate de clienți sunt blocate astfel încât să necesite cel puțin două semnături pentru deblocare, de la Mohammed și unul dintre partenerii săi sau de la avocatul său care are o cheie de rezervă. O schemă multisemnătură de genul acesta oferă control de guvernantă corporativă și protejează împotriva furtului, delapidării sau pierderilor.

Scriptul rezultat este destul de lung și arată astfel:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3 Public  
Key> <Attorney Public Key> 5 CHECKMULTISIG
```

Deși scripturile cu mai multe semnături reprezintă o funcționalitate puternică, ele sunt greoi de utilizat. Având în vedere scriptul precedent, Mohammed ar trebui să comunice acest script fiecărui

client înainte de plată. Fiecare client ar trebui să utilizeze un portofel special bitcoin cu posibilitatea de a crea scripturi de tranzacții personalizate, iar fiecare client ar trebui să înțeleagă cum să creeze o tranzacție folosind scripturi personalizate. Mai mult, tranzacția rezultată ar fi de aproximativ cinci ori mai mare decât o simplă tranzacție de plată, deoarece acest script conține chei publice foarte lungi. Încărcătura acestei tranzacții foarte mari ar fi suportată de client sub formă de comisioane. În cele din urmă, un script de tranzacție mare ca acesta va fi transportat în setul UTXO în memoria RAM din fiecare nod complet, până când va fi cheltuit. Toate aceste probleme fac dificilă utilizarea în practică a scripturilor complexe de blocare.

P2SH a fost dezvoltat pentru a rezolva aceste dificultăți practice și pentru a face utilizarea unor scripturi complexe la fel de ușară ca plata către o adresă bitcoin. Cu plățile P2SH, scriptul de blocare complex este înlocuit cu amprenta sa digitală, un rezumat (hash) criptografic. Când o tranzacție care încearcă să cheltuiască UTXO-ul este prezentată mai târziu, trebuie să conțină scriptul care se potrivește cu rezumatul (hash-ul), pe lângă scriptul de deblocare. În termeni simpli, P2SH înseamnă ”plată către un script care se potrivește cu acest rezumat (hash), un script care va fi prezentat ulterior, atunci când va fi cheltuită această ieșire”.

În tranzacțiile P2SH, scriptul de blocare care este înlocuit cu un rezumat (hash) este denumit *script de răscumpărare* (redeem script) deoarece este prezentat sistemului la momentul răscumpărării și nu ca un script de blocare. [Script complex fără P2SH](#) arată scriptul fără P2SH și [Script complex ca P2SH](#) arată același script codat cu P2SH.

Table 20. Script complex fără P2SH

Script de Blocare	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG
Scriptul de Deblocare	Sig1 Sig2

Table 21. Script complex ca P2SH

Script de Răscumpărare (Redeem Script)	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG
Script de Blocare	HASH160 <20-byte hash of redeem script> EQUAL
Scriptul de Deblocare	Sig1 Sig2 <redeem script>

După cum puteți vedea din tabele, cu P2SH scriptul complex care detaliază condițiile pentru cheltuirea ieșirii (scriptul de răscumpărare) nu este prezentat în scriptul de blocare. În schimb, doar un rezumat (hash) al acestuia este în scriptul de blocare, iar scriptul de răscumpărare în sine este prezentat mai târziu, ca parte a scriptului de deblocare atunci când ieșirea este cheltuită. Aceasta schimbă povara comisioanelor și complexității de la expeditor la destinatarul (cel care cheltuieste) tranzacției.

Să ne uităm la compania lui Mohammed, scriptul multisemnătură complex și scripturile P2SH rezultate.

În primul rând, scriptul multisemnătură pe care compania lui Mohammed îl folosește pentru toate plățile primite de la clienți:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3 Public Key> <Attorney Public Key> 5 CHECKMULTISIG
```

Dacă locațiile de înlocuire sunt substituite cu chei publice reale (afișate aici ca numere pe 520 biți începând cu 04), puteți vedea că acest script devine foarte lung:

```
2
04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395D7EEC6
984D83F1F50C900A24DD47F569FD4193AF5DE762C58704A2192968D8655D6A935BEAF2CA23E3FB87A3495E
7AF308EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD2423657738C0C2B1D5CE65C97D78D0E34224858008E8B
49047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812F393DA7D4420D7E1A9
162F0279CFC10F1E8E8F3020DECDBC3C0DD389D99779650421D65CBD7149B255382ED7F78E946580657EE6
FDA162A187543A9D85BAAA93A4AB3A8F044DADA618D087227440645ABE8A35DA8C5B73997AD343BE5C2AFD
94A5043752580AFA1ECED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF51
8C022DD618DA774D207D137AAB59E0B000EB7ED238F4D800 5 CHECKMULTISIG
```

Întregul script poate fi reprezentat în schimb de un rezumat criptografic pe 20 de octeți, aplicând mai întâi algoritmul de rezumare SHA256 și apoi aplicând algoritmul RIPEMD160 pe rezumat.

Folosim *libbitcoin-explorer (bx)* de la linia de comandă pentru a produce rezumatul scriptului, după cum urmează:

```
echo \
2 \
[04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395D7EEC
6984D83F1F50C900A24DD47F569FD4193AF5DE762C587] \
[04A2192968D8655D6A935BEAF2CA23E3FB87A3495E7AF308EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD24
23657738C0C2B1D5CE65C97D78D0E34224858008E8B49] \
[047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812F393DA7D4420D7E1A91
62F0279CFC10F1E8E8F3020DECDBC3C0DD389D9977965] \
[0421D65CBD7149B255382ED7F78E946580657EE6FDA162A187543A9D85BAAA93A4AB3A8F044DADA618D08
7227440645ABE8A35DA8C5B73997AD343BE5C2AFD94A5] \
[043752580AFA1ECED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF518C0
22DD618DA774D207D137AAB59E0B000EB7ED238F4D800] \
5 CHECKMULTISIG \
| bx script-encode | bx sha256 | bx ripemd160
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

Seria de comenzi de mai sus mai întâi codifică scriptul de răscumpărare multisemnătură a lui Mohammed ca script bitcoin codificat hexa. Următoarea comandă *bx* calculează rezumatul SHA256. Următoarea comandă *bx* rezumă din nou cu RIPEMD160, producând rezumatul final al scriptului:

Rezumatul de 20 de octeți al scriptului de răscumpărare al lui Mohammed este:

```
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

O tranzacție P2SH blochează ieșirea la acest rezumat în locul scriptului de răscumpărare mai lung, folosind scriptul de blocare:

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

care, după cum vedeti, este mult mai scurt. În loc de "plătiți către acest script cu 5 chei multisemnătură", tranzacția echivalentă P2SH este "plătiți unui script cu acest rezumat". Un client care efectuează o plată către compania lui Mohammed trebuie să includă doar scriptul de blocare mult mai scurt în plata sa. Când Mohammed și partenerii săi vor să cheltuiască acest UTXO, trebuie să prezinte scriptul original de răscumpărare (cel al cărui rezumat a blocat UTXO-ul) și semnăturile necesare pentru deblocarea acestuia, astfel:

```
<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG>
```

Cele două scripturi sunt combinate în două etape. În primul rând, scriptul de răscumpărare este verificat cu scriptul de blocare pentru a se asigura că rezumatul se potrivește:

```
<2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG> HASH160 <redeem scriptHash> EQUAL
```

Dacă rezumatul scriptului de răscumpărare se potrivește, scriptul de deblocare este executat de unul singur, pentru a debloca scriptul de răscumpărare:

```
<Sig1> <Sig2> 2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG
```

Aproape toate scripturile descrise în acest capitol pot fi implementate doar ca scripturi P2SH. Acestea nu pot fi utilizate direct în scriptul de blocare al unui UTXO.

## Adrese P2SH

O altă parte importantă a funcționalității P2SH este capacitatea de a codifica un rezumat de script ca adresă, așa cum este definit în BIP-13. Adresele P2SH sunt codificări Base58Check ale rezumatului pe 20 de octeți a unui script, la fel cum adresele bitcoin sunt codificările Base58Check ale rezumatului pe 20 de octeți ale unei chei publice. Adresele P2SH folosesc prefixul de versiune "5", care are ca rezultat adrese codificate Base58Check care încep cu un "3".

De exemplu, scriptul complex al lui Mohammed, rezumat și codificat Base58Check ca adresă P2SH, devine 39RF6JqABIHdYHkfChV6USGMe6Ns66Gzw. Putem confirma asta folosind comanda *bx*:

```
echo \
'54c557e07dde5bb6cb791c7a540e0a4796f5e97e' \
| bx address-encode -v 5
39RF6JqABIHdYHkfChV6USGMe6Ns66Gzw
```

Acum, Mohammed poate oferi această "adresă" clienților săi și ei pot utiliza aproape orice portofel

bitcoin pentru a face o plată simplă, ca și cum ar fi o adresă bitcoin. Prefixul 3 le oferă un indiciu că acesta este un tip special de adresă, una corespunzătoare unui script în loc de o cheie publică, dar altfel funcționează exact în același mod ca o plată către o adresă bitcoin.

Adresele P2SH ascund toată complexitatea, astfel încât persoana care efectuează o plată să nu vadă scriptul.

## Beneficiile P2SH

Funcția P2SH oferă următoarele avantaje în comparație cu utilizarea directă a scripturilor complexe în blocarea ieșirilor:

- Scripturile complexe sunt înlocuite cu amprente mai scurte în ieșirea tranzacției, ceea ce face ca tranzacția să fie mai mică.
- Scripturile pot fi codificate ca adresă, astfel că expeditorul și portofelul expeditorului nu au nevoie de cunoștințe complexe pentru a implementa P2SH.
- P2SH transferă sarcina construirii scriptului către destinatar, nu expeditor.
- P2SH mută sarcina în stocarea datelor pentru scriptul lung de la ieșire (care, pe lângă că este stocat în lanțul-de-blocuri este ținut și în setul UTXO) la intrare (stocat doar în lanțul-de-blocuri).
- P2SH mută sarcina în stocarea datelor pentru scriptul lung de la momentul curent (plata) la o perioadă viitoare (când este cheltuită ieșirea).
- P2SH transferă costul de tranzacție pentru un script lung de la expeditor la destinatar, care trebuie să includă scriptul lung de răscumpărare pentru a cheltui ieșirea.

## Script de Răscumpărare și Validarea Lui

Înainte de versiunea 0.9.2 a clientului Bitcoin Core, Plata-către-Rezumat-Script (Pay-to-Script-Hash) era limitată la tipurile standard de scripturi de tranzacții bitcoin, prin funcția `IsStandard()`. Aceasta înseamnă că scriptul de răscumpărare prezentat în tranzacția de cheltuire nu poate fi decât unul dintre tipurile standard: P2PK, P2PKH sau multisemnătură.

Începând cu versiunea 0.9.2 a clientului Bitcoin Core, tranzacțiile P2SH pot conține orice script valid, făcând standardul P2SH mult mai flexibil și permitând experimentarea cu multe tipuri de tranzacții noi și complexe.

Nu puteți pune un P2SH în interiorul unui script de răscumpărare P2SH, deoarece specificația P2SH nu este recursivă. De asemenea, deși este posibil din punct de vedere tehnic să includeți **RETURN** (vezi [Ieșire de Înregistrare a Datelor \(RETURN\)](#)) într-un script de răscumpărare, deoarece nimic din reguli nu vă împiedică să faceți acest lucru, nu are niciun folos practic, deoarece executarea **RETURN** în timpul validării va face ca tranzacția să fie marcată ca invalidă.

Rețineți că, deoarece scriptul de răscumpărare nu este prezentat rețelei până când nu încercați să cheltuiți o ieșire P2SH, dacă blocați o ieșire cu rezumatul unui script de răscumpărare invalid, acesta va fi procesat oricum. UTXO-ul va fi blocat cu succes. Cu toate acestea, nu îl veți putea cheltui, deoarece tranzacția de cheltuire, care include scriptul de răscumpărare, nu va fi acceptată, deoarece este un script invalid. Acest lucru creează un risc, deoarece puteți bloca bitcoin într-un

P2SH care nu poate fi cheltuit ulterior. Rețeaua va accepta scriptul de blocare P2SH chiar dacă corespunde unui script de răscumpărare invalid, deoarece rezumatul scriptului nu oferă nici un indiciu despre scriptul pe care îl reprezintă.

**WARNING**

Scripturile de blocare P2SH conțin rezumatul unui script de răscumpărare, care nu oferă indicii cu privire la conținutul scriptului de răscumpărare. Tranzacția P2SH va fi considerată validă și acceptată chiar dacă scriptul de răscumpărare nu este valid. S-ar putea să blocați accidental bitcoin în așa fel încât să nu poată fi cheltuit ulterior.

## Ieșire de Înregistrare a Datelor (RETURN)

Registrul bitcoin distribuit și de marcare a timpului (timestamped ledger), lanțul-de-blocuri, are potențiale utilizări care depășesc plățile. Mulți programatori au încercat să utilizeze limbajul de scriptare pentru tranzacții pentru a profita de securitatea și rezistența sistemului în scopul creării de aplicații precum servicii notariale digitale, certificate de stoc și contracte inteligente (smart contracts). Încercările timpurii de a utiliza limbajul de scriptare bitcoin în aceste scopuri au implicat crearea de ieșiri de tranzacții care au înregistrat date pe lanțul-de-blocuri; de exemplu, să înregistreze o amprentă digitală a unui fișier în așa fel încât oricine să poată stabili doveda existenței aceluiași fișier la o dată specifică prin referire la tranzacția respectivă.

Utilizarea lanțului-de-blocuri bitcoin pentru a stoca datele care nu au legătură cu plățile bitcoin este un subiect controversat. Mulți programatori consideră că o astfel de utilizare este abuzivă și vor să o descurajeze. Alții o privesc ca o demonstrație a capacitateilor puternice ale tehnologiei lanț-de-blocuri și vor să încurajeze o astfel de experimentare. Cei care se opun includerii datelor care nu țin de plăți susțin că acestea provoacă "congestionarea lanțului-de-blocuri", care îi încarcă pe cei care rulează noduri bitcoin complete cu costurile de stocare pe disc a datelor pe care lanțul-de-blocuri nu a fost intenționat să le susțină. Mai mult, astfel de tranzacții creează UTXO-uri care nu pot fi cheltuite, folosind adresa bitcoin de destinație ca un câmp de 20 de octeți care poate avea orice valoare. Deoarece adresa este folosită pentru date, aceasta nu corespunde unei chei private, iar UTXO-ul rezultat nu poate fi cheltuită; este o plată falsă. Prin urmare, aceste tranzacții care nu pot fi cheltuite niciodată nu sunt eliminate niciodată din setul UTXO și determină creșterea permanentă a dimensiunii bazei de date UTXO, sau "congestionarea".

În versiunea 0.9 a clientului Bitcoin Core, s-a ajuns la un compromis odată cu introducerea operatorului **RETURN**. **RETURN** permite programatorilor să adauge 80 de octeți de date care nu sunt asociați unei plăți la o ieșire a tranzacției. Cu toate acestea, spre deosebire de utilizarea de UTXO-uri "false", operatorul **RETURN** creează o ieșire *demonstrabil necheltuibilă* în mod explicit, care nu trebuie să fie stocată în setul UTXO. Ieșirile **RETURN** sunt înregistrate în lanțul-de-blocuri, astfel încât consumă spațiu pe disc și contribuie la creșterea dimensiunii lanțului-de-blocuri, dar nu sunt stocate în setul UTXO și, prin urmare, nu crește memoria necesară pentru setul de UTXO și nu încarcă nodurile complete cu costuri de RAM mai mari.

Scripturile **RETURN** arată astfel:

```
RETURN <data>
```

Portiunea de date este limitată la 80 de octeți și cel mai adesea reprezintă un rezumat, cum ar fi cel produs de algoritmul SHA256 (32 octeți). Multe aplicații pun un prefix în fața datelor pentru a ajuta la identificarea aplicației. De exemplu, serviciul de notarizare digitală [Proof of Existence](#) folosește prefixul pe 8 octeți *DOCPROOF*, care este codat ASCII în hexazecimal ca *44 4f 43 50 52 4f 4f 46*.

Rețineți că nu există niciun "script de deblocare" care să corespundă unui **RETURN** care ar putea fi folosit pentru "cheltuirea" unei ieșiri **RETURN**. Întregul scop al **RETURN** este că nu puteți cheltui banii blocați în acea ieșire și, prin urmare, nu este necesar să fie reținută în setul UTXO ca potențial cheltuibilă - **RETURN** este *demonstrabil necheltuibilă*. **RETURN** este de obicei o ieșire cu o sumă de zero bitcoin, deoarece orice bitcoin atribuit unei astfel de ieșiri se pierde efectiv pentru totdeauna. Dacă un **RETURN** este referit ca o intrare într-o tranzacție, motorul de validare a scripturilor va opri execuția scriptului de validare și va marca tranzacția ca invalidă. Execuția lui **RETURN** determină, în esență, ca scriptul să returneze un **FALSE** și să se oprească. Astfel, dacă faceți o referință accidentală la o ieșire **RETURN** ca o intrare într-o tranzacție, acea tranzacție este invalidă.

O tranzacție standard (una conformă cu verificarea [IsStandard\(\)](#)) poate avea doar o ieșire **RETURN**. Cu toate acestea, o singură ieșire **RETURN** poate fi combinată într-o tranzacție cu ieșiri de orice alt tip.

Două opțiuni pentru linia de comandă au fost adăugate în Bitcoin Core începând cu versiunea 0.10. Opțiunea *datacarrier* controlează difuzarea și mineritul tranzacțiilor **RETURN**, cu setarea implicită pe "1" pentru a le permite. Opțiunea *datacarriersize* primește un argument numeric specificând dimensiunea maximă în octeți a scriptului **RETURN**, în mod implicit 83 de octeți, ceea ce permite maxim 80 de octeți pentru valoarea **RETURN** plus un octet pentru codul operatorului **RETURN** și doi octeți pentru codul operatorului **PUSHDATA**.

**NOTE** **RETURN** a fost propus inițial cu o limită de 80 de octeți, dar limita a fost redusă la 40 de octeți când a fost lansată funcționalitatea. În februarie 2015, în versiunea 0.10 a Bitcoin Core, limita a fost ridicată din nou la 80 de octeți. Nodurile pot alege să nu difuzeze sau să mineze scripturi **RETURN**, sau doar să difuzeze și să mineze scripturi **RETURN** care conțin mai puțin de 80 de octeți de date.

## Timpi de Blocare (Timelocks)

Timpii de blocare (timelocks) sunt restricții pentru tranzacții sau pentru ieșiri care permit cheltuirea numai după un moment în timp. Bitcoin a avut de la început o funcționalitate temporală la nivel de tranzacție. Este implementată de câmpul *nLocktime* într-o tranzacție. Două noi funcționalități pentru timpi de blocare au fost introduse la sfârșitul anului 2015 și la jumătatea anului 2016, care oferă timpi de blocare (timelocks) la nivel de UTXO. Acestea sunt **CHECKLOCKTIMEVERIFY** și **CHECKSEQUENCEVERIFY**.

Timpii de blocare (timelocks) sunt utili pentru postdatarea tranzacțiilor și blocarea fondurilor la o anumită dată în viitor. și mai important, timpii de blocare extind scripturile bitcoin în dimensiunea timpului, deschizând ușa pentru contracte inteligente complexe cu mai mulți pași (multistep).

## Timp de Blocare a Tranzacției (nLocktime)

De la început, bitcoin a avut o funcționalitate de timp la nivel de tranzacție. Timpul de blocare (timelock) al tranzacției este o setare la nivel de tranzacție (un câmp din structura datelor de tranzacție) care definește cel mai devreme moment în care o tranzacție este validă și poate fi transmisă în rețea sau adăugată la lanțul-de-blocuri. Timpul de blocare este cunoscut și sub denumirea de **nLocktime** de la numele variabilei utilizate în codul Bitcoin Core. Este setat la zero în majoritatea tranzacțiilor pentru a indica propagarea și execuția imediată. Dacă valoarea **nLocktime** este între zero și sub 500 de milioane, aceasta este interpretată ca înălțimea blocului (block height), ceea ce înseamnă că tranzacția nu este validă și nu este transmisă sau inclusă în lanțul-de-blocuri înainte de înălțimea specificată a blocului. Dacă este mai mare sau egală cu 500 de milioane, se interpretează ca o marcă de timp (timestamp) Unix Epoch (secunde de la 1 ianuarie 1970), iar tranzacția nu este valabilă înainte de data specificată. Tranzacțiile cu **nLocktime** care specifică un bloc sau dată în viitor trebuie să fie păstrate de sistemul inițiator și transmise rețelei bitcoin numai după ce devin valide. Dacă o tranzacție este transmisă în rețea înainte de valoarea specificată pentru **nLocktime**, tranzacția va fi respinsă de primul nod ca fiind invalidă și nu va fi transmisă către alte noduri. Utilizarea **nLocktime** este echivalentă cu postdatarea unui cec de hârtie.

### Limitări ale timpilor de blocare pentru tranzacții

**nLocktime** are limitarea că, deși face posibilă cheltuirea unor ieșiri în viitor, nu face imposibilă cheltuirea lor până la acel moment. Să explicăm asta cu următorul exemplu.

Alice semnează o tranzacție care cheltuiește una dintre ieșirile ei la adresa lui Bob și stabilește valoarea **nLocktime** pentru tranzacție la 3 luni în viitor. Alice trimit acea tranzacție lui Bob să o păstreze. Cu această tranzacție, Alice și Bob știu că:

- Bob nu poate transmite tranzacția pentru a răscumpăra fondurile decât după ce au trecut 3 luni.
- Bob poate transmite tranzacția după 3 luni.

Totuși:

- Alice poate crea o altă tranzacție fără a avea un timp de blocare, cheltuind de două ori aceeași intrări. Astfel, Alice poate cheltui aceeași UTXO înainte de expirarea celor 3 luni.
- Bob nu are nicio garanție că Alice nu va face asta.

Este important să înțelegem limitele **nLocktime** pentru tranzacție. Singura garanție este că Bob nu o va putea răscumpăra înainte de 3 luni. Nu există nicio garanție că Bob va primi fondurile. Pentru a obține o astfel de garanție, restricția de timp trebuie să fie plasată pe UTXO și să facă parte din scriptul de blocare, mai degrabă decât din tranzacție. Acest lucru este realizat prin următoarea formă de timp de blocare, denumită Verifică Control Timp Blocare (Check Lock Time Verify).

## Verifică Control Timp Blocare (Check Lock Time Verify - CLTV)

În decembrie 2015, o nouă formă de timp de blocare (timelock) a fost introdusă în bitcoin ca o actualizare bifurcare soft. Pe baza unei specificații din BIP-65, un operator nou numit **CHECKLOCKTIMEVERIFY** (CLTV) a fost adăugat limbajului de scriptare. **CLTV** este un timp de blocare per ieșire, mai degrabă decât un timp de blocare per tranzacție, cum este cazul cu **nLocktime**. Acest

lucru permite o flexibilitate mult mai mare în modul de aplicare a timpilor de blocare.

În termeni simpli, prin adăugarea operatorului **CLTV** în scriptul de răscumpărare a unei ieșiri, acesta restricționează ieșirea, astfel încât aceasta să nu poate fi cheltuită decât după scurgerea timpului specificat.

**TIP** În timp ce **nLocktime** este un timp de blocare la nivel de **tranzacție**, **CLTV** este un timp de blocare la nivel de **ieșire**.

**CLTV** nu înlocuiește **nLocktime**, ci mai degrabă restricționează UTXO-uri specifice, astfel încât acestea pot fi cheltuite doar într-o tranzacție viitoare cu **nLocktime** setat la o valoare mai mare sau egală.

Operatorul **CLTV** primește un parametru ca intrare, exprimat ca număr în același format ca **nLocktime** (fie o înălțime a blocului, fie un timp de epocă Unix). După cum indică sufixul **VERIFY**, **CLTV** este tipul de operator care oprește execuția scriptului dacă rezultatul este FALSE. Dacă are ca rezultat TRUE, execuția continuă.

Pentru a bloca o ieșire cu **CLTV**, introduceți operatorul în scriptul de răscumpărare a ieșirii în tranzacția care creează ieșirea. De exemplu, dacă Alice plătește adresa lui Bob, ieșirea ar conține în mod normal un script P2PKH astfel:

```
DUP HASH160 <Bob's Public Key Hash> EQUALVERIFY CHECKSIG
```

Pentru a o bloca la o dată anume, să spunem peste 3 luni, tranzacția ar fi o tranzacție P2SH cu un script de răscumpărare ca acesta:

```
<now + 3 months> CHECKLOCKTIMEVERIFY DROP DUP HASH160 <Bob's Public Key Hash>  
EQUALVERIFY CHECKSIG
```

unde **<now + 3 months>** este o înălțime a blocului sau o valoare a duratei estimată la 3 luni de la minarea tranzacției: înălțimea curentă a blocului + 12.960 (blocuri) sau timpul curent de epocă Unix + 7.760.000 (secunde). Deocamdată nu vă faceți griji în legătură cu operatorul **DROP** care urmează după **CHECKLOCKTIMEVERIFY**; va fi explicat în scurt timp.

Când Bob încearcă să cheltuiască această UTXO, el construiește o tranzacție care face referire la UTXO ca intrare. Își folosește semnătura și cheia publică în scriptul de deblocare a acelei intrări și stabilește ca timpul de blocare al tranzacției **nLocktime** să fie egal sau mai mare cu timpul de blocare pe care Alice l-a specificat pentru **CHECKLOCKTIMEVERIFY**. Bob transmite apoi tranzacția în rețea bitcoin.

Tranzacția lui Bob este evaluată după cum urmează. Dacă parametrul pentru **CHECKLOCKTIMEVERIFY** pe care Alice l-a setat este mai mic sau egal cu valoarea **nLocktime** a tranzacției de cheltuire, atunci execuția scriptului continuă (acționează ca și cum nu a fost executată "nici o operațiune" sau NOP). În caz contrar, execuția scriptului se oprește și tranzacția este considerată invalidă.

Mai precis, **CHECKLOCKTIMEVERIFY** eșuează și oprește executarea, marcând tranzacția invalidă dacă (sursa: BIP-65):

1. stiva este goală; sau
2. elementul din vârf al stivei este mai mic de 0; sau
3. tipul timpului de blocare (înălțime versus marcă de timp) al elementului din vârful stivei și câmpul `nLocktime` nu sunt aceleiasi; sau
4. elementul din vârful stivei este mai mare decât câmpul `nLocktime` setat pentru tranzacție; sau
5. câmpul `nSequence` al intrării este 0xffffffff.

**NOTE**

`CLTV` și `nLocktime` utilizează același format pentru a descrie timpii de blocare, fie o înălțime a blocului, fie timpul scurs în secunde de la epoca Unix. În mod critic, atunci când este utilizat împreună, formatul `nLocktime` trebuie să se potrivească cu cel al `CLTV` setat pentru ieșiri - acestea trebuie să facă referință fie la înălțimea blocului, fie la timpul în secunde.

După executare, dacă `CLTV` este satisfăcut, parametrul de timp care l-a precedat rămâne ca elementul din vârful stivei și poate fi nevoie să fie îndepărtat, cu `DROP`, pentru executarea corectă a operatorilor ulteriori. De multe ori veți vedea în scripturi `CHECKLOCKTIMEVERIFY` urmat de `DROP` din acest motiv.

Folosind `nLocktime` în combinație cu `CLTV`, scenariul descris în [Limitări ale timpilor de blocare pentru tranzacții](#) se schimbă. Alice nu mai poate cheltui banii (pentru că sunt blocați cu cheia lui Bob) și Bob nu își poate cheltui înainte de expirarea perioadei de 3 luni.

Prin introducerea funcționalității timpilor de blocare (timelock) direct în limbajul de scriptare, `CLTV` ne permite să dezvoltăm niște scripturi complexe foarte interesante.

Standardul este definit la [BIP-65 \(CHECKLOCKTIMEVERIFY\)](#).

## Timpi Relativi de Blocare

`nLocktime` și `CLTV` sunt ambele *timpi de blocare absoluți* prin faptul că specifică un punct absolut în timp. Următoarele două funcționalități pentru timpi de blocare pe care le vom examina sunt *timpi de blocare relativi* prin faptul că ele specifică, ca și condiție de cheltuire a unei ieșiri, un timp scurs de la confirmarea ieșirii în lanțul-de-blocuri.

Timpii relativi de blocare sunt utili deoarece permit un lanț de două sau mai multe tranzacții interdependente să fie păstrate în afara lanțului (off chain), impunând în același timp o restricție de timp asupra unei tranzacții care depinde de timpul scurs de la confirmarea unei tranzacții anterioare. Cu alte cuvinte, ceasul nu începe să numere până când UTXO-ul nu este înregistrat în lanțul-de-blocuri. Această funcționalitate este utilă în special în canalele bidirectionale cu stare și în Lightning Networks, aşa cum vom vedea în [Canale de Plată și Canale de Stare](#).

Timpii relativi de blocare, la fel ca timpii absoluți, sunt implementați atât ca o funcționalitate la nivel de tranzacție, cât și ca un operator la nivel de script. Timpul relativ de blocare la nivelul tranzacției este implementat ca regulă de consens asupra valorii `nSequence`, un câmp al tranzacției care este setat în fiecare intrare din tranzacție. Timpii relativi de blocare la nivel de script sunt implementați folosind operatorul `CHECKSEQUENCEVERIFY` (CSV).

Timpii relativi de blocare sunt implementați în conformitate cu specificațiile de la [BIP-68, Relative](#)

BIP-68 și BIP-112 au fost activate în mai 2016, ca o actualizare bifurcare soft la regulile de consens.

## Timpi Relativi de Blocare folosind nSequence

Timpii relativi de blocare pot fi setați pe fiecare intrare a unei tranzacții, prin setarea câmpului **nSequence** în fiecare intrare.

### Semnificația originală a nSequence

Câmpul **nSequence** a fost inițial conceput (dar niciodată implementat corespunzător) pentru a permite modificarea tranzacțiilor din mempool. În această utilizare, o tranzacție care conține intrări cu o valoare **nSequence** sub  $2^{32} - 1$  (0xFFFFFFFF) însemna o tranzacție care nu a fost încă "finalizată". O astfel de tranzacție va fi păstrată în mempool până când aceasta va fi înlocuită cu o altă tranzacție care cheltuie aceleași intrări cu o valoare **nSequence** mai mare. Odată ce s-a primit o tranzacție ale cărei intrări au o valoare de 0xFFFFFFFF pentru **nSequence**, aceasta va fi considerată "finalizată" și minată.

Intenția inițială a **nSequence** nu a fost niciodată implementată în mod corespunzător, iar valoarea **nSequence** este setată în mod obișnuit la 0xFFFFFFFF în tranzacțiile care nu utilizează timpi de blocare. Pentru tranzacțiile cu **nLocktime** sau **CHECKLOCKTIMEVERIFY**, valoarea **nSequence** trebuie să fie setată la mai puțin de  $2^{31}$  pentru ca protecția de timp de blocare să aibă un efect, așa cum este explicitat mai jos.

### nSequence ca un timp relativ de blocare impus de consens

De la activarea BIP-68, se aplică noi reguli de consens pentru orice tranzacție care conține o intrare a cărei valoare **nSequence** este mai mică de  $2^{31}$  (bitul  $1 \ll 31$  nu este setat). Programatic, asta înseamnă că, dacă cel mai semnificativ bit (bit  $1 \ll 31$ ) nu este setat, este un indicator care înseamnă "timp relativ de blocare" activat. În caz contrar (bitul  $1 \ll 31$  este setat), valoarea **nSequence** este rezervată altor utilizări, cum ar fi activarea **CHECKLOCKTIMEVERIFY**, **nLocktime**, Opt-In-Replace-By-Fee și a altor evoluții viitoare.

Intrările tranzacției cu valori **nSequence** mai mici de  $2^{31}$  sunt interpretate ca având un timp relativ de blocare. O astfel de tranzacție este valabilă numai după ce intrarea a îmbătrânit cu valoarea timpului relativ de blocare. De exemplu, o tranzacție cu o singură intrare cu un timp relativ de blocare **nSequence** de 30 de blocuri este valabilă numai atunci când au trecut cel puțin 30 de blocuri din momentul în care UTXO-ul la care se face referință în intrare a fost minat. Deoarece **nSequence** este un câmp per intrare, iar o tranzacție poate conține oricâte intrări cu timp de blocare, atunci toate intrările trebuie să aibă o vârstă suficientă pentru ca tranzacția să fie validă. O tranzacție poate include atât intrări cu timpi de blocare (**nSequence**  $< 2^{31}$ ), cât și intrări fără un timp relativ de blocare (**nSequence**  $\geq 2^{31}$ ).

Valoarea **nSequence** este specificată fie în blocuri, fie în secunde, dar într-un format ușor diferit decât am văzut folosit la **nLocktime**. Un indicator de tip este utilizat pentru a diferenția între o valoare care se referă la blocuri și una care se referă la timpul scurs în secunde. Indicatorul de tip este setat în al 23-lea cel mai puțin semnificativ bit (adică, valoarea  $1 \ll 22$ ). Dacă este setat indicatorul tip, atunci valoarea **nSequence** este interpretată ca un multiplu de 512 secunde. Dacă

indicatorul tip nu este setat, valoarea `nSequence` este interpretată ca un număr de blocuri.

Atunci când se interpretează `nSequence` ca un timp relativ de blocare, sunt considerați doar cei 16 biți mai puțin semnificativi. Odată ce indicatorii (biții 32 și 23) sunt evaluați, valoarea `nSequence` este de obicei "mascată" cu o mască de 16 biți (de exemplu, `nSequence` & `0x0000FFFF`).

[Definiția BIP-68 a codificării `nSequence`](#) (Sursa: BIP-68) arată structura binară a valorii `nSequence`, astăzi cum este definită de BIP-68.

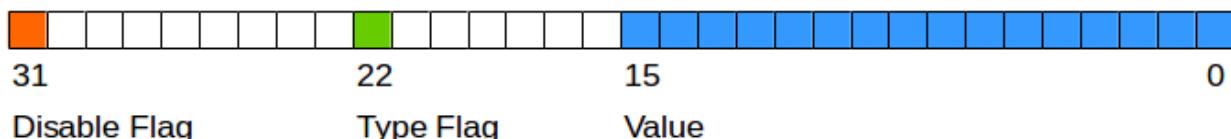


Figure 48. Definiția BIP-68 a codificării `nSequence` (Sursa: BIP-68)

Timpii relativi de blocare bazați pe aplicarea consensului asupra valorii `nSequence` sunt definiți în BIP-68.

Standardul este definit la [BIP-68, Relative lock-time using consensus-enforced sequence numbers](#).

## Timpi Relativi de Blocare cu CSV

La fel ca CLTV și `nLocktime`, există un operator de script pentru timpii relativi de blocare care utilizează valoarea `nSequence` în scripturi. Acest operator este `CHECKSEQUENCEVERIFY`, denumit în mod ușual pe scurt `CSV`.

Când este evaluat, operatorul `CSV` în scriptul de răscumpărare al unei UTXO permite cheltuirea numai într-o tranzacție a cărei intrare are valoarea `nSequence` mai mare sau egală cu parametrul `CSV`. În esență, acest lucru restricționează cheltuirea UTXO-ului până când au trecut un anumit număr de blocuri sau secunde în raport cu timpul în care UTXO a fost minat.

Ca și în cazul CLTV, valoarea din `CSV` trebuie să corespundă formatului valorii corespunzătoare din `nSequence`. Dacă `CSV` este specificat în termeni de blocuri, atunci astăzi trebuie să `nSequence`. Dacă `CSV` este specificat în termeni de secunde, atunci astăzi trebuie și `nSequence`.

Timpii relativi de blocare cu `CSV` sunt utili în special atunci când mai multe tranzacții (în lanț) sunt create și semnate, dar nu propagă, atunci când sunt păstrate în afara lanțului ("off-chain"). O tranzacție secundară nu poate fi utilizată până când tranzacția părinte nu a fost propagată, minată și îmbătrânită până la momentul specificat în timpul relativ de blocare. O aplicare a acestui caz de utilizare poate fi văzută în [Canale de Plată și Canale de Stare](#) și [Canale de Plată Rutate \(Lightning Network\)](#).

`CSV` este definit în detaliu la [BIP-112, CHECKSEQUENCEVERIFY](#).

## Timpul-Median-Trecut (Median-Time-Past)

Ca parte a activării timpilor relativi de blocare, a existat, de asemenea, o modificare a modului în care se calculează "data" pentru timpii de blocare (atât absoluți cât și relativi). În bitcoin există o diferență subtilă, dar foarte semnificativă, între timpul de pe ceas și timpul de consens. Bitcoin este o rețea descentralizată, ceea ce înseamnă că fiecare participant are propria sa perspectivă asupra timpului. Evenimentele din rețea nu au loc instantaneu peste tot. Latența rețelei trebuie luată în considerare în perspectiva fiecărui nod. În cele din urmă, totul este sincronizat pentru a crea un registru comun. Bitcoin ajunge la un consens la fiecare 10 minute cu privire la starea registrului, așa cum a existat în *trecut*.

Mărcile de timp (timestamps) puse în antetul blocului sunt stabilite de către mineri. Există un anumit grad de libertate permis de regulile de consens pentru a ține cont de diferențele de precizie a ceasului dintre nodurile descentralizate. Cu toate acestea, acest lucru creează un stimulent nefericit pentru mineri să mintă în legătură cu data unui bloc, astfel încât să obțină comisioane suplimentare prin includerea tranzacțiilor cu timpi de blocare care încă nu sunt mature. Consultați secțiunea următoare pentru mai multe informații.

Pentru a elimina stimulentul de a minti și pentru a consolida securitatea timpilor de blocare, a fost propus și activat un BIP în același timp cu BIP-urile pentru timpii relativi de blocare. Aceasta este BIP-113, care definește o nouă măsurare de consens a timpului numită *Timpul-Median-Trecut* (Median-Time-Past).

Timpul-Median-Trecut (Median-Time-Past) este calculat citind timpul din ultimele 11 blocuri și găsind mediana. Acest timp median devine apoi timp de consens și este utilizat pentru toate calculele timpilor de blocare. Prin preluarea punctului mediu de la aproximativ două ore în trecut, se reduce influența timpului oricărui bloc. Prin încorporarea a 11 blocuri, niciun miner nu poate influența timpul pentru a obține comisioane din tranzacțiile cu un timp de blocare care nu s-a scurs încă.

Timpul-Median-Trecut modifică implementarea calculelor de timp pentru [nLocktime](#), [CLTV](#), [nSequence](#), și [CSV](#). Timpul de consens calculat de Timpul-Median-Trecut este întotdeauna la aproximativ o oră în urma ceasului de pe perete. Dacă creați tranzacții cu timp de blocare, ar trebui să luați în calcul acest lucru când estimați valoarea dorită pentru [nLocktime](#), [nSequence](#), [CLTV](#), și [CSV](#).

Timpul-Median-Trecut este specificat în [BIP-113](#).

## Apărare Folosind Timpul de Blocare Împotriva Smulgerii de Comisioane

Smulgerea de comisioane (fee-sniping) este un scenariu de atac teoretic, în care minerii încearcă să rescrie blocurile din trecut pentru a "smulge" comisioane mai mari din blocuri viitoare pentru a-și maximiza profitabilitatea.

De exemplu, să spunem că cel mai recent bloc existent este blocul #100.000. Dacă în loc să încerce să mineze blocul #100,001 pentru extinderea lanțului, unii mineri încearcă să remineze blocul #100.000. Acești mineri pot alege să includă orice tranzacție validă (care încă nu a fost minată) în blocul candidat #100.000. Ei nu trebuie să remineze blocul cu aceleși tranzacții. De fapt, aceștia au stimulentul de a selecta cele mai rentabile (cea mai mare taxă per kB) tranzacții pe care să le

includă în blocul lor. Aceștia pot include orice tranzacții care s-au aflat în blocul "vechi" #100.000, precum și orice tranzacții din mempool-ul actual. În esență, au opțiunea de a trage tranzacțiile din "prezent" în "trecutul" rescris când re-creează blocul #100.000.

Astăzi, acest atac nu este foarte profitabil, deoarece recompensa per bloc minat este mult mai mare decât comisioanele totale per bloc. Dar la un moment dat în viitor, comisioanele de tranzacție vor compune majoritatea recompensei (sau chiar întreaga recompensă). În acel moment, acest scenariu devine inevitabil.

Pentru a preveni "smulgerea comisioanelor", atunci când Bitcoin Core creează tranzacții, se folosește `nLocktime` pentru a le limita de la "următorul bloc", în mod implicit. În scenariul nostru, Bitcoin Core ar seta `nLocktime` la 100,001 la orice tranzacție pe care ar crea-o. În circumstanțe normale, acest `nLocktime` nu are efect - oricum tranzacțiile pot fi incluse doar începând cu blocul #100,001 (următorul bloc).

Însă, în urma unui atac de bifurcare a lanțului-de-blocuri, minerii nu vor fi capabili să extragă tranzacții cu comisioane mari din mempool, deoarece toate aceste tranzacții ar avea timpi de blocare de la blocul #100,001. Aceștia pot doar să remineze blocul cu numărul #100.000 cu orice tranzacții au fost valide la acea dată, neobținând în esență niciun nou comision.

Pentru a realiza acest lucru, Bitcoin Core stabilește `nLocktime` la toate tranzacțiile noi la `<current block # + 1>` și stabilește `nSequence` pe toate intrările la 0xFFFFFFFF pentru a activa `nLocktime`.

## Scripturi cu Control al Execuției (Clauze Condiționate)

Una dintre funcționalitățile mai puternice ale Bitcoin Script este controlul fluxului de execuție, cunoscut și sub denumirea de clauze condiționale. Probabil că sunteți familiarizați cu controlul fluxului de execuție din diverse limbaje de programare care utilizează forma `IF ... THEN ... ELSE`. Clauzele condiționate din Bitcoin arată un pic diferit, dar sunt, în esență, aceleași structuri.

La un nivel de bază, operatorii condiționali bitcoin ne permit să construim un script de răscumpărare care are două moduri de a fi deblocat, în funcție de rezultatul `TRUE/FALSE` de evaluare a unei condiții logice. De exemplu, dacă x este `TRUE` (adevărat), scriptul de răscumpărare este A, iar altfel (`ELSE`) scriptul de răscumpărare este B.

În plus, expresiile condiționate bitcoin pot fi "imbricate" la nesfârșit, ceea ce înseamnă că o clauză condițională poate conține o altă în cadrul ei, care conține o altă, etc. Nu există nicio limită pentru imbricare, dar regulile de consens impun o limită pentru dimensiunea maximă, în octeți, a unui script.

Bitcoin implementează controlul fluxului de execuție folosind operatorii `IF`, `ELSE`, `ENDIF`, și `NOTIF`. În plus, expresiile condiționate pot conține operatori booleeni, cum ar fi `BOOLAND`, `BOOLOR` și `NOT`.

La prima vedere, scripturile bitcoin de control al fluxului de execuție pot părea confuze. Acest lucru se datorează faptului că Bitcoin Script este un limbaj pe stivă. În același mod în care `1 + 1` arată "invers" atunci când este exprimat ca `1 1 ADD`, clauzele de control al fluxului de execuție în bitcoin arată, de asemenea, "invers".

În majoritatea limbajelor de programare tradiționale (procedurale), controlul fluxului de execuție arată astfel:

*Pseudocod al controlului fluxului de execuție în majoritatea limbajelor de programare*

```
if (condition):  
    code to run when condition is true  
else:  
    code to run when condition is false  
code to run in either case
```

Într-un limbaj bazat pe stivă precum Bitcoin Script, condiția logică vine înainte de **IF**, ceea ce îl face să arate "invers", astfel:

*Controlul fluxului de execuție în Bitcoin Script*

```
condition  
IF  
    code to run when condition is true  
ELSE  
    code to run when condition is false  
ENDIF  
code to run in either case
```

Când citiți Bitcoin Script, amintiți-vă că condiția evaluată vine *înainte* de operatorul **IF**.

## Clauze condiționale cu operatorii de tip **VERIFY**

O altă formă condițională în Bitcoin Script este orice operator care se termină în **VERIFY**. Sufixul **VERIFY** înseamnă că dacă condiția evaluată nu este **TRUE** (adevărată), execuția scriptului se termină imediat și tranzacția este considerată invalidă.

Spre deosebire de o clauză **IF**, care oferă căi alternative de execuție, sufixul **VERIFY** acționează ca o *clauză gardă*, continuând doar dacă este îndeplinită o precondiție.

De exemplu, scriptul următor necesită semnătura lui Bob și o pre-imagine (secret) care produce un rezumat (hash) specific. Ambele condiții trebuie îndeplinite:

*Un script de răscumpărare cu o clauză gardă **EQUALVERIFY**.*

```
HASH160 <expected hash> EQUALVERIFY <Bob's Pubkey> CHECKSIG
```

Pentru a-l răscumpăra, Bob trebuie să construiască un script de deblocare care să prezinte o pre-imagine validă și o semnătură:

*Un script de deblocare pentru a satisface scriptul de răscumpărare de mai sus*

```
<Bob's Sig> <hash pre-image>
```

Fără a prezenta pre-imaginiea, Bob nu poate ajunge la partea din script care verifică semnătura sa.

Acest script poate fi scris cu un **IF** în schimb:

*Un script de răscumpărare cu o clauză gardă **IF***

```
HASH160 <expected hash> EQUAL
IF
  <Bob's Pubkey> CHECKSIG
ENDIF
```

Scriptul de deblocare al lui Bob este identic:

*Un script de deblocare pentru a satisface scriptul de răscumpărare de mai sus*

```
<Bob's Sig> <hash pre-image>
```

Scriptul cu **IF** face același lucru ca și folosirea unui operator cu sufixul **VERIFY**; ambele funcționează ca și clauze gardă. Cu toate acestea, construcția **VERIFY** este mai eficientă, folosind cu doi operatori mai puțin.

Deci, când folosim **VERIFY** și când folosim **IF**? Dacă tot ce încercăm să facem este să atașăm o precondiție (clauză gardă), atunci **VERIFY** este mai bun. Dacă, totuși, dorim să avem mai mult de o cale de execuție, atunci avem nevoie de o clauză **IF ... ELSE**.

**TIP** Un operator precum **EQUAL** va împinge rezultatul (**TRUE/FALSE**) pe stivă, lăsându-l acolo pentru evaluarea de către următorii operatori. În schimb, operatorul **EQUALVERIFY** nu lasă nimic pe stivă. Codurile care se termină cu **VERIFY** nu lasă rezultatul pe stivă.

## Utilizarea Controlului Fluxului de Execuție în Scripturi

O utilizare foarte întâlnită pentru controlul fluxului în Bitcoin Script este construirea unui script de răscumpărare care oferă mai multe căi de execuție, fiecare cale fiind un mod diferit de răscumpărare a UTXO.

Să ne uităm la un exemplu simplu, în care avem doi semnatari, Alice și Bob, și fiecare dintre ei este în măsură să răscumpere. Folosind multisemnătură, acest lucru ar fi exprimat ca un script multisemnătură 1 din 2. De dragul demonstrației, vom face același lucru cu o clauză **IF**:

```
IF
  <Alice's Pubkey> CHECKSIG
ELSE
  <Bob's Pubkey> CHECKSIG
ENDIF
```

Privind acest script de răscumpărare, s-ar putea să vă întrebați: "Unde este condiția? Nu există nimic care să precede clauza **IF**!"

Condiția nu face parte din scriptul de răscumpărare. În schimb, condiția va fi oferită în scriptul de deblocare, permitându-i lui Alice și Bob să ”aleagă” ce cale de execuție doresc.

Alice răscumpără cu scriptul de deblocare:

```
<Alice's Sig> 1
```

1 la sfârșit servește ca condiție (TRUE) care va face ca clauza IF să execute prima cale de răscumpărare pentru care Alice are o semnătură.

Pentru ca Bob să răscumpere, ar trebui să aleagă a doua cale de execuție dând o valoare FALSE (falsă) clauzei IF:

```
<Bob's Sig> 0
```

Scriptul de deblocare al lui Bob pune un 0 pe stivă, determinând clauza IF să execute al doilea script (ELSE), care necesită semnătura lui Bob.

Din moment ce clauzele IF pot fi imbinate, putem crea un ”labyrinth” de căi de execuție. Scriptul de deblocare poate furniza o ”hartă” care selectează calea de execuție care este efectiv executată:

```
IF
  script A
ELSE
  IF
    script B
  ELSE
    script C
  ENDIF
ENDIF
```

În acest scenariu, există trei căi de execuție (script A, script B și script C). Scriptul de deblocare oferă o cale sub forma unei secvențe de valori TRUE sau FALSE. Pentru a selecta calea script B, de exemplu, scriptul de deblocare trebuie să se termine în 1 0 (TRUE, FALSE). Aceste valori vor fi împinsă pe stivă, astfel încât a doua valoare (FALSE) să ajungă în partea de sus a stivei. Clauza exterioară IF scoate valoarea FALSE și execută prima clauză ELSE. Apoi, valoarea TRUE ajunge în vârful stivei și este evaluată de IF-ul interior (imbricat) selectând calea de execuție B.

Folosind această formă, putem construi scripturi de răscumpărare cu zeci sau sute de căi de execuție, fiecare oferind un mod diferit de a răscumpăra o UTXO. Pentru a cheltui, construim un script de deblocare care navighează pe calea de execuție punând valorile corespunzătoare TRUE și FALSE pe stivă la fiecare punct de control al fluxului de execuție.

## Exemplu de Script Complex

În această secțiune vom combina multe dintre concepții din acest capitol într-un singur exemplu.

Exemplul nostru folosește povestea lui Mohammed, proprietarul companiei din Dubai care operează o afacere de import/export.

În acest exemplu, Mohammed dorește să creeze pentru companie un cont de capital cu reguli flexibile. Schema pe care o creează necesită diferite niveluri de autorizare în funcție de timpi de blocare. Participanții la schema multisemnătură sunt Mohammed, cei doi parteneri Saeed și Zaira și avocatul companiei Abdul. Cei trei parteneri iau decizii pe baza regulii majorității, deci doi dintre cei trei trebuie să fie de acord. Cu toate acestea, în cazul unei probleme cu cheile lor, ei doresc ca avocatul lor să poată recupera fondurile împreună cu una dintre cele trei semnături ale partenerilor. În cele din urmă, dacă toți partenerii sunt indisponibili sau incapacitați pentru o perioadă, ei doresc ca avocatul să poată gestiona contul în mod direct.

Iată scriptul de răscumpărare pe care Mohammed îl creează pentru a realiza acest lucru (prefixul este numărul liniei în forma XX):

*Mulitsemnătură variabilă cu timp-de-blocare*

```
01 IF
02   IF
03     2
04   ELSE
05     <30 days> CHECKSEQUENCEVERIFY DROP
06     <Abdul the Lawyer's Pubkey> CHECKSIGVERIFY
07     1
08   ENDIF
09   <Mohammed's Pubkey> <Saeed's Pubkey> <Zaira's Pubkey> 3 CHECKMULTISIG
10 ELSE
11   <90 days> CHECKSEQUENCEVERIFY DROP
12   <Abdul the Lawyer's Pubkey> CHECKSIG
13 ENDIF
```

Scriptul lui Mohammed implementează trei căi de execuție folosind clauze **IF**...**ELSE** imbricate.

În prima cale de execuție, acest script funcționează ca un o simplă multisemnătură 2-din-3 cu cei trei parteneri. Această cale de execuție constă din liniile 3 și 9. Linia 3 stabilește cvorumul multisemnăturii la 2 (2-din-3). Această cale de execuție poate fi selectată punând **TRUE** **TRUE** la sfârșitul scriptului de deblocare:

*Scriptul de deblocare pentru prima cale de execuție (multisemnătura 2-din-3)*

```
0 <Mohammed's Sig> <Zaira's Sig> TRUE TRUE
```

**TIP** **0** la începutul acestui script de deblocare se datorează unui defect al **CHECKMULTISIG** care scoate o valoare suplimentară de pe stivă. Valoarea suplimentară este ignorată de **CHECKMULTISIG**, dar trebuie să fie prezentă sau scriptul eșuează. Adăugare **0** (în mod obișnuit) este o soluție de rezolvare a erorii, aşa cum este descris în [Un defect în execuția CHECKMULTISIG](#).

A doua cale de execuție poate fi utilizată numai după ce au trecut 30 de zile de la crearea UTXO-

ului. La acel moment, este nevoie de semnătura lui Abdul avocatul, și a unuia dintre cei trei parteneri (o multisemnătură 1-din-3). Acest lucru este realizat la linia 7, care stabilește cvorumul pentru multisig la 1. Pentru a selecta această cale de execuție, scriptul de deblocare ar trebui să se termine în FALSE TRUE:

*Scriptul de deblocare pentru a doua cale de execuție (avocat + 1-din-3)*

```
0 <Saeed's Sig> <Abdul's Sig> FALSE TRUE
```

**TIP** De ce FALSE TRUE? Nu este invers? Deoarece cele două valori sunt împinse pe stivă, cu FALSE împins mai întâi, apoi TRUE împins al doilea. Prin urmare, TRUE este scos *primul* de către operatorul IF.

În cele din urmă, a treia cale de execuție îi permite lui Abdul avocatul să cheltuiască fondurile singur, dar numai după 90 de zile. Pentru a selecta această cale de execuție, scriptul de deblocare trebuie să se termine în FALSE:

*Deblocarea scriptului pentru a treia cale de execuție (doar de către avocat)*

```
<Abdul's Sig> FALSE
```

Încercați să rulați scriptul pe hârtie pentru a vedea cum se comportă pe stivă.

Alte câteva lucruri de luat în considerare atunci când citiți acest exemplu. Vedeți dacă puteți găsi răspunsurile:

- De ce nu poate avocatul răscumpăra în orice moment a treia cale de execuție, selectând-o cu FALSE pe scriptul de deblocare?
- Câte căi de execuție pot fi utilizate la 5, 35 și respectiv, 105 zile, după ce UTXO-ul a fost minat?
- Sunt fondurile pierdute dacă avocatul își pierde cheia? Răspunsul dumneavoastră se schimbă dacă au trecut 91 de zile?
- Cum ”reseteză” partenerii ceasul la fiecare 29 sau 89 de zile pentru a împiedica avocatul să acceseze fondurile?
- De ce unele coduri CHECKSIG din acest script au sufixul VERIFY în timp ce altele nu?

## Martor Segregat

Funcționalitatea de Martorul Segregat (Segregated Witness - segwit) este o îmbunătățire adusă regulilor de consens bitcoin și protocolului de rețea, propusă și implementată ca bifurcare soft BIP-9 care a fost activat în rețeaua bitcoin la 1 august 2017.

În criptografie, termenul ”martor” este folosit pentru a descrie o soluție pentru un puzzle criptografic. În termeni bitcoin, martorul îndeplinește o condiție criptografică plasată pe o ieșire de tranzacție necheltuită (UTXO).

În contextul bitcoin, o semnătură digitală este *un tip de martor*, dar un martor este mai pe larg orice

soluție care poate satisface condițiile impuse unei UTXO și poate debloca acea UTXO pentru a fi cheltuită. Termenul "martor" este un termen mai general pentru un "script de deblocare" sau "scriptSig".

Înainte de introducerea segwit, fiecare intrare dintr-o tranzacție era urmată de datele martor care au deblocat-o. Datele martor au fost încorporate în tranzacție ca parte a fiecărei intrări. Termenul *martor segregat* (segregated witness), sau *segwit* pe scurt, înseamnă pur și simplu separarea semnăturii sau a scriptului de deblocare pentru o ieșire specifică. Gândiți-vă că la "scriptSig separat" sau "semnătura separată" în cea mai simplă formă.

Prin urmare, Martorul Segregat este o modificare arhitecturală asupra bitcoin care are ca scop mutarea datelor martor din câmpul *scriptSig* (script de deblocare) a unei tranzacții într-o structură de date *martor separată* care însoțește o tranzacție. Clienții pot solicita datele tranzacției cu sau fără datele martor care le însoțesc.

În această secțiune vom analiza unele dintre beneficiile Martor Segregat, vom descrie mecanismul folosit pentru instalarea și implementarea acestei schimbări de arhitectură și vom demonstra utilizarea Martor Segregat în tranzacții și adrese.

Martorul Segregat este definit de următoarele BIP-uri:

#### **BIP-141**

Definiția principală a Martor Segregat.

#### **BIP-143**

Verificarea Semnăturii Tranzacției pentru Programul Martor Versiunea 0

#### **BIP-144**

Servicii de-la-egal-la-egal - Mesaje de rețea noi și formate de serializare

#### **BIP-145**

*getblocktemplate* Actualizări pentru Martor Segregat (pentru minerit)

#### **BIP-173**

Formatul adresei Base32 pentru ieșirile martor native v0-16

## **De ce Martor Segregat?**

Martorul Segregat este o schimbare arhitecturală care are mai multe efecte asupra scalabilității, securității, stimulentelor economice și performanței bitcoin:

### **Maleabilitatea Tranzacției**

Prin mutarea martorului în afara tranzacției, rezumatul (hash-ul) tranzacției folosit ca identificator nu mai include datele martor. Deoarece datele martor sunt singura parte a tranzacției care poate fi modificată de un terț (vezi [Identificatori de tranzacție](#)), eliminarea acestora elimină și posibilitatea atacurilor de maleabilitate a tranzacțiilor. Cu Martor Segregat, rezumatele (hash-urile) tranzacțiilor devin imutabile față de oricine altcineva în afara creatorului tranzacției, ceea ce îmbunătățește considerabil implementarea multor altor protocoale care se bazează pe construcția avansată a tranzacțiilor bitcoin, cum ar fi canalele de

plată, tranzacții înlănțuite și rețele lightning.

## Versionarea Scripturilor

Odată cu introducerea scripturilor Martor Segregat, fiecare script de blocare este precedat de un număr de *versiune script*, similar cu modul în care tranzacțiile și blocurile au numere de versiune. Adăugarea unui număr de versiune de script permite actualizarea limbajului de script într-un mod compatibil (întotdeauna, folosind bifurcări soft) pentru a introduce noi operatori de script, sintaxă nouă sau semantică nouă. Capacitatea de a actualiza limbajul de scriptare într-un mod nedisruptiv va accelera mult rata de inovație în bitcoin.

## Scalarea Rețelei și a Stocării

Datele martor sunt adesea un mare contribuitor la dimensiunea totală a unei tranzacții. Scripturi mai complexe, precum cele utilizate pentru multisemnătură sau canale de plată, sunt foarte mari. În unele cazuri, aceste scripturi reprezintă majoritatea (mai mult de 75%) din datele dintr-o tranzacție. Prin mutarea datelor martor în afara tranzacției, Martorul Segregat îmbunătățește scalabilitatea bitcoin. Nodurile pot să curețe datele martor după validarea semnăturilor sau să le ignore cu totul atunci când fac verificări de plată simplificate. Datele martor nu trebuie transmise tuturor nodurilor și nu trebuie să fie stocate pe disc de către toate nodurile.

## Optimizarea Verificării Semnăturii

Martorul Segregat îmbunătățește funcțiile de semnătură (**CHECKSIG**, **CHECKMULTISIG**, etc.) prin reducerea complexității de calcul a algoritmului. Înainte de segwit, algoritmul folosit pentru producerea unei semnături necesită un număr de operații de rezumare proporțional cu dimensiunea tranzacției. Calculele pentru rezumarea datelor au crescut în  $O(n^2)$  raportat la numărul de operațiuni de semnătură, introducând o povară de calcul substanțială pe toate nodurile care verifică semnătura. Cu segwit, algoritmul este schimbat pentru a reduce complexitatea la  $O(n)$ .

## Îmbunătățirea Semnării Offline

Semnăturile Martor Segregat includ încorporate valoarea (suma) la care face referire fiecare intrare din rezumatul (hash-ul) semnat. Anterior, un dispozitiv de semnare offline, cum ar fi un portofel hardware, ar fi trebuit să verifice valoarea fiecărei intrări înainte de a semna o tranzacție. Acest lucru se realiza de obicei prin transmiterea unei cantități mari de date despre tranzacțiile anterioare menționate ca intrări. Din moment ce suma face parte din rezumatul (hash-ul) de angajament semnat, un dispozitiv offline nu are nevoie de tranzacțiile anterioare. Dacă sumele nu se potrivesc (sunt prezentate greșit de un sistem online compromis), semnătura nu va fi validă.

## Cum Funcționează Martorul Segregat

La prima vedere, Martorul Segregat pare să fie o modificare a modului în care se construiesc tranzacțiile și, prin urmare, o caracteristică la nivel de tranzacție, dar nu este așa. Mai degrabă, Martorul Segregat este o schimbare a modului în care sunt cheltuite UTXO-urile individuale și, prin urmare, este o caracteristică **per-iesire**.

O tranzacție poate cheltui ieșiri Martor Segregat sau ieșiri tradiționale (martor încorporat), sau ambele. Prin urmare, nu are prea mult sens să ne referim la o tranzacție ca "tranzacție Martor

Segregat". Mai degrabă, ar trebui să ne referim la ieșiri specifice ale tranzacțiilor ca "ieșiri Martor Segregat".

Când o tranzacție cheltuiește o UTXO, trebuie să ofere un martor. Într-o UTXO tradițională, scriptul de blocare necesită ca datele martor să fie *incorporate* în partea de intrări a tranzacției care cheltuie UTXO-ul. Totuși, o UTXO Martor Segregat specifică un script de blocare care poate fi satisfăcut cu datele martor exterioare intrării (segregate).

## Bifurcare Soft (Compatibilitate cu Versiunile Anterioare)

Martorul Segregat reprezintă o schimbare semnificativă a modului în care sunt structurate ieșirile și tranzacțiile. O astfel de modificare ar necesita, în mod normal, o schimbare simultană în fiecare nod și portofel bitcoin pentru a schimba regulile de consens - ceea ce este cunoscut sub numele de bifurcare hard. În schimb, martorul segregat este introdus cu o schimbare mult mai puțin perturbatoare, care este compatibilă cu versiunile anterioare, cunoscută sub numele de bifurcare soft. Acest tip de îmbunătățire permite software-ului neactualizat să ignore modificările și să continue să funcționeze fără întreruperi.

Ieșirile Martor Segregat sunt construite astfel încât sistemele mai vechi care nu folosesc segwit să le poată totuși valida. Pentru un portofel sau nod vechi, o ieșire Martor Segregat pare o ieșire pe care *oricine o poate cheltui*. Astfel de ieșiri pot fi cheltuite cu o semnătură goală, prin urmare, faptul că nu există nicio semnătură în cadrul tranzacției (este segregată) nu invalidează tranzacția. Portofelele și nodurile mai noi, însă, văd ieșirea Martor Segregat și se așteaptă să găsească un martor valid pentru aceasta în datele martor ale tranzacției.

## Exemple de Ieșire și Tranzacție cu Martor Segregat

Să ne uităm la unele dintre exemplele noastre de tranzacții și să vedem cum s-ar schimba acestea când folosim Martor Segregat. Vom analiza mai întâi cum se transformă o Plată-către-Cheie-Publică (P2PKH) cu ajutorul programului Martor Segregat. Apoi, ne vom uita la echivalentul Martor Segregat pentru scripturile Plată-către-Rezumat-Script (P2SH). În cele din urmă, vom analiza modul în care ambele programe precedente cu Martori Segregat pot fi *incorporate* într-un script P2SH.

### Plată-către-Martor-Rezumat-Cheie-Publică (Pay-to-Witness-Public-Key-Hash - P2WPKH)

În [Cumpărarea unei Cești de Cafea](#), Alice a creat o tranzacție pentru a-i plăti lui Bob o ceașcă de cafea. Acea tranzacție a creat o ieșire P2PKH cu o valoare de 0,015 BTC care a fost cheltuită de Bob. Scriptul ieșirii arată astfel:

*Exemplu script ieșire P2PKH*

```
DUP HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7 EQUALVERIFY CHECKSIG
```

Cu Martor Segregat, Alice ar crea un script Plată-către-Martor-Rezumat-Cheie-Publică (P2WPKH), care arată astfel:

### Exemplu script ieșire P2WPKH

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

După cum vedeți, scriptul de blocare a unei ieșiri cu Martorilor Segregat este mult mai simplu decât cel al unei ieșiri tradiționale. Este format din două valori care sunt împins pe stiva de evaluare a scriptului. Pentru un client bitcoin vechi (fără segwit), cele două împingeri ar arăta ca o ieșire pe care o poate cheltui oricine și nu are nevoie de o semnătură (sau mai degrabă, poate fi cheltuită cu o semnătură goală). Pentru un client mai nou, care folosește segwit, primul număr (0) este interpretat ca un număr de versiune (*versiunea martor*), iar a doua parte (20 de octeți) este echivalentul unui script de blocare cunoscut sub numele de program *martor*. Programul martor de 20 de octeți este pur și simplu rezumatul (hash-ul) cheii publice, ca într-un script P2PKH.

Acum, să ne uităm la tranzacția corespunzătoare pe care Bob o folosește pentru a cheltui această ieșire. Pentru scriptul original (nonsegwit), tranzacția lui Bob ar trebui să includă o semnătură în intrarea tranzacției:

*Tranzacția decodată care arată o ieșire P2PKH ce este cheltuită cu o semnătură*

```
[...]
"Vin" : [
  "txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fdb8a57286c345c2f2",
  "vout": 0,
    "scriptSig": "<Bob's scriptSig>",
]
[...]
```

Cu toate acestea, pentru a cheltui ieșirea cu Martor Segregat, tranzacția nu are o semnătură pe această intrare. În schimb, tranzacția lui Bob are un script *scriptSig* gol, și include un Martor Segregat în exteriorul tranzacției în sine:

*Tranzacția decodată care arată o ieșire P2WPKH cheltuită folosind date martor separate*

```
[...]
"Vin" : [
  "txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fdb8a57286c345c2f2",
  "vout": 0,
    "scriptSig": "",
]
[...]
"witness": "<Bob's witness data>"
[...]
```

### Construcția portofelului pentru P2WPKH

Este extrem de important de reținut faptul că P2WPKH ar trebui să fie creat doar de beneficiar (destinatar) și nu convertit de expeditor dintr-o cheie publică cunoscută, script P2PKH sau adresă. Expeditorul nu are cum să știe dacă portofelul destinatarului are capacitatea de a construi tranzac-

ții segwit și de a cheltui ieșirile P2WPKH.

În plus, ieșirile P2WPKH trebuie să fie construite din rezumatul unei chei publice *comprimate*. Cheile publice necomprimate nu sunt standard în segwit și e posibil să fie dezactivate explicit de o viitoare bifurcare soft. Dacă rezumatul (hash-ul) folosit în P2WPKH a provenit de la o cheie publică necomprimată, este posibil să fie necheltuibil și puteți pierde fonduri. Ieșirile P2WPKH ar trebui create de portofelul beneficiarului, derivând o cheie publică comprimată din cheia lor privată.

#### WARNING

P2WPKH ar trebui să fie construit de beneficiar (destinatar) prin transformarea unei chei publice comprimate într-un rezumat (hash) P2WPKH. Niciodată nu ar trebui să transformați un script P2PKH, o adresă bitcoin sau o cheie publică necomprimată într-un script martor P2WPKH.

### Plată-către-Martor-Rezumat-Script (Pay-to-Witness-Script-Hash - P2WSH)

Al doilea tip de program martor corespunde unui script Plată-către-Rezumat-Script (P2SH). Am văzut acest tip de script în [Plată-către-Rezumat-Script \(Pay-to-Script-Hash - P2SH\)](#). În acel exemplu, P2SH a fost folosit de compania lui Mohammed pentru a exprima un script multisemnătură. Plățile către compania lui Mohammed au fost codate cu un script de blocare ca acesta:

*Exemplu script ieșire P2SH*

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

Acest script P2SH face referire la rezumatul (hash-ul) unui *script de răscumpărare* care definește o multisemnătură 2-din-3 pentru a cheltui fonduri. Pentru a cheltui această ieșire, compania lui Mohammed va prezenta scriptul de răscumpărare (al cărui rezumat se potrivește cu rezumatul scriptului în ieșirea P2SH) și semnăturile necesare pentru a satisface scriptul de răscumpărare, toate în interiorul intrării tranzacției:

*Tranzacție decodată care prezintă cheltuirea unei ieșiri P2SH*

```
[...]
"Vin" : [
  "txid": "abcdef12345...",
  "vout": 0,
    "scriptSig": "<SigA> <SigB> <2 PubA PubB PubC PubD PubE 5 CHECKMULTISIG",
]
]
```

Acum, să ne uităm la modul în care acest exemplu ar fi actualizat la segwit. Dacă clienții lui Mohammed foloseau un portofel compatibil cu segwit, ar fi făcut o plată, creând o ieșire Plată-către-Martor-Rezumat-Script (P2WSH) care ar arăta astfel:

*Exemplu script ieșire P2WSH*

```
0 a9b7b38d972cabc7961dbfbcb841ad4508d133c47ba87457b4a0e8aae86dbb89
```

Din nou, la fel ca în exemplul P2WPKH, puteți vedea că scriptul echivalent cu Martor Segregat este

mult mai simplu și omite diferenți operanzi pe care îi vedeți în scripturile P2SH. În schimb, programul Martor Segregat constă din două valori împinse pe stivă: o versiune martor (0) și rezumatul SHA256 pe 32 de octeți al scriptului de răscumpărare.

În timp ce P2SH folosește rezumatul pe 20 octeți **RIPEMD160 (SHA256 (script))**, programul martor P2WSH folosește un rezumat pe 32 de octeți **SHA256 (script)**. Această diferență în selectarea algoritmului de rezumare este deliberată și este **TIP** utilizată pentru a diferenția între cele două tipuri de programe martor (P2WPKH și P2WSH) în funcție de lungimea rezumatului și pentru a asigura o securitate mai puternică pentru P2WSH (128 biți de securitate pentru P2WSH față de 80 biți de securitate pentru P2SH).

Compania lui Mohammed poate cheltui ieșirea P2WSH prezentând scriptul corect de răscumpărare și semnături suficiente pentru a-l satisface. Atât scriptul de răscumpărare, cât și semnăturile vor fi separate în *exteriorul* tranzacției de cheltuire ca parte a datelor martor. În cadrul intrării tranzacției, portofelul lui Mohammed va pune o valoarea goală pentru **scriptSig**:

*Tranzacție decodată care arată o ieșire P2WSH cheltuită folosind date martor separate*

```
[...]
"Vin" : [
  "txid": "abcdef12345...",
  "vout": 0,
    "scriptSig": "",
]
[...]
"witness": "<SigA> <SigB> <2 PubA PubB PubC PubD PubE 5 CHECKMULTISIG>"
[...]
```

## Diferențierea între P2WPKH și P2WSH

În cele două secțiuni anterioare, am demonstrat două tipuri de programe de martor: [Plată-către-Martor-Rezumat-Cheie-Publică \(Pay-to-Witness-Public-Key-Hash - P2WPKH\)](#) și [Plată-către-Martor-Rezumat-Script \(Pay-to-Witness-Script-Hash - P2WSH\)](#). Ambele tipuri de programe martor constau dintr-un număr de versiune pe un singur octet urmat de un rezumat mai lung. Acestea arată foarte asemănător, dar sunt interpretate foarte diferit: unul este interpretat ca un rezumat de cheie publică, care este satisfăcut de o semnătură, iar celălalt ca un rezumat de script, care este satisfăcut de un script de răscumpărare. Diferența critică dintre ele este lungimea rezumatului:

- Rezumatul cheii publice în P2WPKH este de 20 de octeți
- Rezumatul scriptului în P2WSH este de 32 de octeți

Aceasta este singura diferență care permite unui portofel să diferențieze între cele două tipuri de programe martor. Analizând lungimea rezumatului, un portofel poate determina ce tip de program martor este, P2WPKH sau P2WSH.

## Trecerea la Martor Segregat

După cum putem vedea din exemplele anterioare, trecerea la Martor Segregat este un proces în două etape. În primul rând, portofelele trebuie să creeze ieșiri speciale de tip segwit. Apoi, aceste ieșiri pot fi cheltuite de portofele care știu să construască tranzacții cu Martor Segregat. În exemplele precedente, portofelul lui Alice putea folosi segwit și era capabil să creeze ieșiri speciale cu scripturi Martor Segregat. Portofelul lui Bob, de asemenea, poate folosi segwit și este capabil să cheltuiască aceste ieșiri. Ceea ce nu poate fi evident din exemplu este că, în practică, portofelul lui Alice trebuie să știe că Bob folosește un portofel segwit și poate cheltui aceste ieșiri. În caz contrar, dacă portofelul lui Bob nu este actualizat și Alice încearcă să efectueze plăți segwit către Bob, portofelul lui Bob nu va putea detecta aceste plăți.

**TIP** Pentru tipurile de plată P2WPKH și P2WSH, atât portofelele expeditorului cât și destinatarul trebuie actualizate pentru a putea utiliza segwit. Mai mult, portofelul expeditorului trebuie să știe că portofelul destinatarului poate folosi segwit.

Martorul Segregat nu va fi implementat simultan în întreaga rețea. Mai degrabă, Martorul Segregat este implementat ca o actualizare compatibilă cu versiunile precedente, unde *clientii vechi și noi pot coexista*. Dezvoltatorii de portofele vor actualiza în mod independent software-ul portofel pentru a adăuga funcționalitățile segwit. Tipurile de plată P2WPKH și P2WSH sunt utilizate atunci când atât expeditorul cât și destinatarul pot folosi segwit. P2PKH și P2SH tradiționale vor continua să funcționeze pentru portofelele neactualizate. Aceasta creează două scenarii importante, care sunt abordate în secțiunea următoare:

- Posibilitatea portofelului unui expeditor care nu folosește segwit de a efectua o plată către portofelul unui destinatar care poate procesa tranzacții segwit
- Posibilitatea portofelului unui expeditor care folosește segwit de a distinge între destinatarii care folosesc segwit și cei care nu folosesc, bazat pe *adresele lor*.

## Încorporarea Martor Segregat în P2SH

Să presupunem, de exemplu, că portofelul lui Alice nu este actualizat să folosească segwit, dar portofelul lui Bob este actualizat și poate trata tranzacții segwit. Alice și Bob pot folosi tranzacții "vechi" non-segwit. Însă Bob ar dori să folosească segwit pentru a reduce comisioanele de tranzacție, profitând de reducerea care se aplică datelor martor.

În acest caz portofelul lui Bob poate construi o adresă P2SH care conține un script segwit în interiorul său. Portofelul lui Alice consideră că este o adresă P2SH "normală" și poate face plăți către ea fără să știe informații despre segwit. Portofelul lui Bob poate cheltui această plată cu o tranzacție segwit, profitând din plin de segwit și reducând comisioanele de tranzacție.

Ambele forme de scripturi martor, P2WPKH și P2WSH, pot fi încorporate într-o adresă P2SH. Primul este notat ca P2SH (P2WPKH), iar al doilea este notat ca P2SH(P2WSH).

## Plată-către-Martor-Rezumat-Cheie-Publică în interiorul Plată-către-Rezumat-Script

Prima formă de script martor pe care o vom examina este P2SH(P2WPKH). Acesta este un program martor Plată-către-Martor-Rezumat-Cheie-Publică, încorporat într-un script Plată-către-Rezumat-Script, astfel încât să poată fi folosit de un portofel care nu știe segwit.

Portofelul lui Bob construiește un program martor P2WPKH cu cheia publică a lui Bob. Acest program martor este apoi rezumat, iar rezumatul rezultat este codat ca un script P2SH. Scriptul P2SH este convertit într-o adresă bitcoin, una care începe cu un "3", așa cum am văzut în secțiunea [Plată-către-Rezumat-Script \(Pay-to-Script-Hash - P2SH\)](#).

Portofelul lui Bob începe cu programul martor P2WPKH pe care l-am văzut mai devreme:

*Programul martor P2WPKH al lui Bob*

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

Programul martor P2WPKH este format din versiunea martorului și rezumatul cheii publice pe 20 de octeți al lui Bob.

Portofelul lui Bob rezumă apoi programul martor precedent, mai întâi cu SHA256, apoi cu RIPEMD160, producând încă un rezumat pe 20 de biți.

Să folosim *bx* de la linia de comandă pentru a replica că:

*HASH160 din programul martor P2WPKH*

```
echo \
'0 [ab68025513c3dbd2f7b92a94e0581f5d50f654e7]' \
| bx script-encode | bx sha256 | bx ripemd160
3e0547268b3b19288b3adef9719ec8659f4b2b0b
```

În continuare, rezumatul scriptului de răscumpărare este convertit într-o adresă bitcoin. Să folosim din nou *bx* de la linia de comandă:

*P2SH address*

```
echo \
'3e0547268b3b19288b3adef9719ec8659f4b2b0b' \
| bx address-encode -v 5
37Lx99uaGn5avKBxiW26HqedQE3LrDCZru
```

Acum, Bob poate afișa această adresă iar clienții pot să plătească pentru cafeaua lor. Portofelul lui Alice poate efectua o plată către *37Lx99uaGn5avKBxiW26HqedQE3LrDCZru*, la fel cum ar face-o către orice altă adresă bitcoin.

Pentru a-i plăti lui Bob, portofelul lui Alice ar bloca ieșirea cu un script P2SH:

```
HASH160 3e0547268b3b19288b3adef9719ec8659f4b2b0b EQUAL
```

Chiar dacă portofelul lui Alice nu are suport pentru segwit, plata pe care o creează poate fi cheltuită de Bob cu o tranzacție segwit.

## Plată-către-Martor-Rezumat-Script în interiorul Plată-către-Rezumat-Script

În mod similar, un program martor P2WSH pentru un script multisemnătură sau alt script complicat poate fi încorporat într-un script și adresă P2SH, ceea ce face posibil ca orice portofel să facă plăți compatibile cu segwit.

După cum am văzut în [Plată-către-Martor-Rezumat-Script \(Pay-to-Witness-Script-Hash - P2WSH\)](#), Compania lui Mohammed utilizează plăți cu Martor Segregat cu scripturi multisemnătură. Pentru a face posibil ca orice client să poată efectua plăți către compania lui, indiferent dacă portofelele lor sunt actualizate pentru segwit sau nu, portofelul lui Mohammed poate încorpora programul martor P2WSH într-un script P2SH.

În primul rând, portofelul lui Mohammed rezumă scriptul de răscumpărare cu SHA256 (o singură dată). Să folosim *bx* pentru a face aceasta de la linia de comandă:

*Portofelul lui Mohammed creează un program martor P2WSH*

```
echo \
2 \
[04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395D7EEC
6984D83F1F50C900A24DD47F569FD4193AF5DE762C587] \
[04A2192968D8655D6A935BEAF2CA23E3FB87A3495E7AF308EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD24
23657738C0C2B1D5CE65C97D78D0E34224858008E8B49] \
[047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812F393DA7D4420D7E1A91
62F0279CFC10F1E8E8F3020DECDBC3C0DD389D9977965] \
[0421D65CBD7149B255382ED7F78E946580657EE6FDA162A187543A9D85BAAA93A4AB3A8F044DADA618D08
7227440645ABE8A35DA8C5B73997AD343BE5C2AFD94A5] \
[043752580AFA1ECED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF518C0
22DD618DA774D207D137AAB59E0B000EB7ED238F4D800] \
5 CHECKMULTISIG \
| bx script-encode | bx sha256
9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

În continuare, rezumatul scriptului de răscumpărare este transformat într-un program martor P2WSH:

```
0 9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

Apoi, programul martor însuși este rezumat cu SHA256 și RIPEMD160, producând un nou rezumat pe 20 de octeți, aşa cum este folosit în P2SH tradițional. Să folosim *bx* de la linia de comandă pentru a face asta:

*HASH160 din programul martor P2WSH*

```
echo \
'0 [9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73]' \
| bx script-encode | bx sha256 | bx ripemd160
86762607e8fe87c0c37740cddee880988b9455b2
```

În continuare, portofelul construiește o adresă bitcoin P2SH din acest rezumat. Din nou, folosim *bx* pentru a calcula de la linia de comandă:

#### Adresa bitcoin P2SH

```
echo \
'86762607e8fe87c0c37740cddee880988b9455b2' \
| bx address-encode -v 5
3Dwz1MXhM6EfFoJChHCxh1jWHb8GQqRenG
```

Acum, clienții lui Mohammed pot efectua plăți către această adresă fără a fi nevoie să suporte segwit. Pentru a trimite o plată către Mohammed, un portofel ar bloca ieșirea cu următorul script P2SH:

*Scriptul P2SH folosit pentru a bloca plățile pentru scriptul multisemnătură a lui Mohammed*

```
HASH160 86762607e8fe87c0c37740cddee880988b9455b2 EQUAL
```

Compania lui Mohammed poate apoi să realizeze tranzacții segwit pentru a cheltui aceste plăți, profitând de funcționalitățile segwit, inclusiv comisioane de tranzacție mai mici.

#### Adrese Martor Segregat

Chiar și după activarea segwit, va dura ceva timp până când majoritatea portofelelor vor fi actualizate. La început, segwit va fi încorporat în P2SH, aşa cum am văzut în secțiunea precedentă, pentru a ușura compatibilitatea între portofelele care nu folosesc segwit și cele care folosesc.

Cu toate acestea, odată ce portofelele acceptă pe larg segwit, are sens să codificați scripturile martor direct într-un format de adresă nativ conceput pentru segwit, decât să-l încorporați în P2SH.

Formatul de adresă nativ segwit este definit în BIP-173:

#### BIP-173

Formatul adresei Base32 pentru ieșirile martor native v0-16

BIP-173 codifică doar scripturile martor (P2WPKH și P2WSH). Nu este compatibil cu scripturile P2PKH sau P2SH non-segwit. BIP-173 este o codare Base32 cu verificare, similară cu codificarea Base58 a unei adrese bitcoin "tradiționale". Adresele BIP-173 mai sunt numite adrese *bech32*, pronunțate "beh-ch thirty two", care fac aluzie la utilizarea unui algoritm "BCH" de detectare a erorilor și a unui set de codare pe 32 de caractere.

Adresele BIP-173 folosesc 32 de caractere alfanumerice cu litere mici, selectate cu atenție pentru a reduce erorile de citire sau dactilografiere. Alegând un set de caractere cu litere mici, bech32 este mai ușor de citit, de vorbit și de 45% mai eficient de codat în coduri QR.

Algoritmul de detectare a erorilor BCH este o îmbunătățire semnificativă față de algoritmul precedent de control (de la Base58Check), permitând nu numai detectarea, ci și *corectarea* erorilor. Interfețele de introducere a adresei (cum ar fi câmpurile text din formulare) pot detecta și eviden-

ția caracterul cel mai probabil greșit atunci când au detectat o eroare.

Din specificația BIP-173, iată câteva exemple de adrese bech32:

### Mainnet P2WPKH

bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4

### Testnet P2WPKH

tb1qw508d6qejxtdg4y5r3zarvary0c5xw7kxpjzsx

### Mainnet P2WSH

bc1qrp33g0q5c5txsp9arysr4k6zdkfs4nce4xj0gdcccefvpysxf3qccfmv3

### Testnet P2WSH

tb1qrp33g0q5c5txsp9arysr4k6zdkfs4nce4xj0gdcccefvpysxf3q0sl5k7

După cum puteți vedea în aceste exemple, un sir de caractere segwit bech32 are o lungime de până la 90 de caractere și este format din trei părți:

### Partea lizibilă pentru om

Acest prefix "bc" sau "tb" identifică mainnet (rețeaua principală) sau testnet (rețeaua de test).

### Separatorul

Cifra "1", care nu face parte din setul de codificare pe 32 de caractere și poate apărea doar în această poziție ca separator

### Partea de date

Un minim de 6 caractere alfanumerice, martorul script codificat cu sumă de control

În acest moment, doar câteva portofele acceptă sau produc adrese segwit bech32 native, dar pe măsură ce adoptarea segwit crește, le veți vedea din ce în ce mai des.

### Identifieri de tranzacție

Unul dintre cele mai mari avantaje ale Martorului Segregat este faptul că elimină maleabilitatea tranzacțiilor de către terți.

Înainte de segwit, tranzacțiile puteau avea semnăturile lor modificate în mod subtil de către terți, schimbând ID-ul tranzacției (rezumatul) fără a modifica proprietăți fundamentale (intrări, ieșiri, sume). Acest lucru a creat oportunități pentru atacuri denial-of-service, precum și atacuri împotriva unui software portofel prost scris, care presupune că rezumatele de tranzacție neconfirmate sunt imutabile.

Odată cu introducerea Martorului Segregat, tranzacțiile au doi identifieri, **txid** și **wtxid**. ID-ul tranzacției tradiționale **txid** este rezumatul dublu-SHA256 al tranzacției serializate, fără datele martor. O tranzacție **wtxid** este rezumatul dublu-SHA256 al noului format de serializare al tranzacției cu datele martor.

ID-ul tradițional **txid** este calculat exact în același mod ca și în cazul unei tranzacții nonsegwit. Cu toate acestea, din moment ce tranzacția segwit are **scriptSig**-urile goale pentru fiecare intrare, nu

există nicio parte a tranzacției care să poată fi modificată de o terță parte. Prin urmare, într-o tranzacție segwit, **txid** este imutabil de către o terță parte, chiar și atunci când tranzacția este neconfirmată.

**wtxid** este ca un ID "extins", prin faptul că rezumatul încorporează și datele martor. Dacă o tranzacție este transmisă fără date martor, atunci **wtxid** și **txid** sunt identice. Rețineți că, deoarece **wtxid** include datele martor (semnături) și întrucât datele martor pot fi maleabile, **wtxid** ar trebui să fie considerat maleabil până la confirmarea tranzacției. Doar **txid**-ul unei tranzacții segwit poate fi considerat imutabil de către terți și numai dacă *toate* intrările tranzacției sunt intrări segwit.

**TIP**

Tranzacțiile Martor Segregat au două ID-uri: **txid** și **wtxid**. **txid** este rezumatul tranzacției fără datele martor, iar **wtxid** este rezumatul cu datele martor incluse. **txid**-ul unei tranzacții în care toate intrările sunt segwit nu este susceptibil la maleabilitatea tranzacțiilor de către terți.

## Noul Algoritm de Semnare pentru Martor Segregat

Martorul Segregat modifică semantica celor patru funcții de verificare a semnăturilor (**CHECKSIG**, **CHECKSIGVERIFY**, **CHECKMULTISIG**, și **CHECKMULTISIGVERIFY**), schimbând modul în care este calculat rezumatul angajamentului de tranzacție.

Semnăturile din tranzacțiile bitcoin sunt aplicate pe un *rezumat de angajament*, care este calculat din datele tranzacției, blocând anumite părți ale datelor care indică angajamentul semnatarului pentru aceste valori. De exemplu, într-o simplă semnătură de tipul **SIGHASH\_ALL**, rezumatul de angajament include toate intrările și ieșirile.

Din păcate, modul în care a fost calculat rezumatul de angajament a introdus posibilitatea ca un nod care verifică semnătura să fie obligat să efectueze un număr semnificativ de calcule de rezumare. Mai exact, operațiile de rezumare cresc în  $O(n^2)$  raportat la numărul de operațiuni de semnare din tranzacție. Prin urmare, un atacator ar putea crea o tranzacție cu un număr foarte mare de operațiuni de semnare, determinând întreaga rețea bitcoin să efectueze sute sau mii de operațiuni de rezumare pentru a verifica tranzacția.

Segwit a reprezentat o oportunitate de a aborda această problemă schimbând modul în care este calculat rezumatul angajamentului. Pentru programele martor segwit versiunea 0, verificarea semnăturii are loc folosind un algoritm de rezumare de angajament îmbunătățit, așa cum este specificat în BIP-143.

Noul algoritm atinge două obiective importante. În primul rând, numărul de operații de rezumare crește mult mai gradual cu  $O(n)$  raportat la numărul de operațiuni de semnătură, reducând posibilitatea de a crea atacuri denial-of-service cu tranzacții excesiv de complexe. În al doilea rând, rezumatul de angajament include, de asemenea, valoarea (sumele) fiecărei intrări ca parte a angajamentului. Aceasta înseamnă că un semnatar se poate angaja la o valoare specifică de intrare fără a fi nevoie să "aducă" și să verifice tranzacția anterioară la care a făcut referire. În cazul dispozitivelor offline, cum ar fi portofelele hardware, acest lucru simplifică foarte mult comunicarea între gazdă și portofelul hardware, eliminând nevoia de a parurge tranzacții anterioare pentru validare. Un portofel hardware poate accepta valoarea de intrare "așa cum este declarată" de către o gazdă de încredere. Deoarece semnătura nu este validă dacă acea valoare de intrare nu este corectă, portofelul hardware nu trebuie să valideze valoarea înainte de semnarea

intrării.

## Stimulente economice pentru Martor Segregat

Nodurile de minerit bitcoin și nodurile complete suportă costuri pentru resursele utilizate pentru a susține rețeaua bitcoin și lanțul-de-bocuri. Pe măsură ce volumul tranzacțiilor bitcoin crește, la fel crește și costul resurselor (procesor, lățime de bandă a rețelei, spațiu pe disc, memorie). Minerii sunt compensați pentru aceste costuri prin comisioane proporționale cu dimensiunea (în octeți) a fiecarei tranzacții. Nodurile complete care nu minează nu sunt compensate, deci ele suportă aceste costuri deoarece au nevoie să ruleze un nod complet de validare a indexului, poate pentru că folosesc nodul pentru a derula o afacere bitcoin.

Fără comisioane de tranzacție, creșterea datelor bitcoin ar putea escalada dramatic. Comisioanele sunt destinate să alinieze nevoile utilizatorilor bitcoin cu sarcina pe care tranzacțiile o au asupra rețelei, printr-un mecanism de stabilire a prețurilor într-o piață liberă.

Calculul comisioanelor pe baza mărimei tranzacției tratează toate datele din tranzacție ca fiind egale ca și cost. Dar, din perspectiva nodurilor complete și a minerilor, anumite părți ale unei tranzacții creează costuri mult mai mari. Fiecare tranzacție adăugată rețelei bitcoin afectează consumul a patru resurse:

### Spațiu pe Disc

Fiecare tranzacție este stocată în lanțul-de-blocuri, adăugând la dimensiunea totală a lanțului-de-blocuri. Lanțul-de-blocuri este stocat pe disc, dar stocarea poate fi optimizată prin "retezarea" tranzacțiilor mai vechi.

### CPU

Fiecare tranzacție trebuie validată, ceea ce necesită timp de procesor.

### Lățime de bandă

Fiecare tranzacție este transmisă (prin propagarea prin inundare) în rețea cel puțin o dată. Fără nici o optimizare în protocolul de propagare a blocului, tranzacțiile sunt transmise din nou ca parte a unui bloc, dublând impactul asupra capacitatei rețelei.

### Memorie

Nodurile care validează tranzacțiile păstrează indexul UTXO sau întreg setul UTXO în memorie pentru a accelera validarea. Deoarece memoria este cu cel puțin un ordin de mărime mai scumpă decât spațiul pe disc, creșterea setului UTXO contribuie în mod disproportional la costul rulării unui nod.

După cum puteți vedea din listă, nu fiecare parte a unei tranzacții are un impact egal asupra costului de rulare a unui nod sau asupra capacitatei bitcoin de a scala pentru a susține mai multe tranzacții. Cea mai scumpă parte a unei tranzacții sunt ieșirile nou create, deoarece acestea sunt adăugate la setul UTXO din memorie. Prin comparație, semnăturile (datele martor) adaugă cea mai mică povară rețelei și costurilor de rulare a unui nod, deoarece datele martor sunt validate o singură dată și nu se mai folosesc niciodată. În plus, imediat după primirea unei noi tranzacții și validarea datelor martor, nodurile pot renunța la datele respective. Dacă comisioanele sunt calculate pentru dimensiunea tranzacției, fără a face discriminări între aceste două tipuri de date, atunci stimulentele de piață ale comisioanelor nu sunt aliniate la costurile reale impuse de o

tranzacție. De fapt, structura actuală a comisioanelor încurajează comportamentul opus, deoarece datele martor sunt cea mai mare parte a unei tranzacții.

Stimulentele create de comisioane contează pentru că afectează comportamentul portofelelor. Toate portofelele trebuie să implementeze o strategie pentru asamblarea tranzacțiilor care ia în considerare o serie de factori, cum ar fi confidențialitatea (reducerea reutilizării adreselor), fragmentarea (producerea de mărunțiș) și comisioanele. Dacă comisioanele motivează copleșitor portofelele să folosească cât mai puține intrări în tranzacții, acest lucru poate duce la strategii de alegere a UTXO-urilor și a adreselor de rest care din neatenție pot umfla setul UTXO.

Tranzacțiile consumă UTXO-uri în intrările lor și creează UTXO-uri noi cu ieșirile lor. Prin urmare, o tranzacție, care are mai multe intrări decât ieșiri, va duce la o scădere a setului UTXO, în timp ce o tranzacție care are mai multe ieșiri decât intrări va duce la o creștere a setului UTXO. Să luăm în considerare *diferența* dintre intrări și ieșiri și să o numim la "UTXO-nou-Net". Aceasta este o metrică importantă, deoarece ne spune ce impact va avea o tranzacție asupra celei mai scumpe resurse din toată rețeaua, setul UTXO din memorie. O tranzacție cu un UTXO-nou-Net pozitiv crește povara. O tranzacție cu un UTXO-nou-Net negativ reduce povara. Prin urmare, am dori să încurajăm tranzacțiile care au un UTXO-nou-Net negativ sau neutru, cu zero UTXO-nou-Net.

Să ne uităm la un exemplu de stimulente care sunt create prin calculul comisionului de tranzacție, cu și fără Martor Segregat. Vom analiza două tranzacții diferite. Tranzacția A este o tranzacție cu 3 intrări și 2 ieșiri, care are o valoare UTXO-nou-Net de -1, ceea ce înseamnă că va consuma cu un UTXO mai mult decât creează, reducând setul UTXO cu unul. Tranzacția B este o tranzacție cu 2 intrări și 3 ieșiri, care are o valoare UTXO-nou-Net de 1, ceea ce înseamnă că adaugă o UTXO la setul UTXO, impunând un cost suplimentar întregii rețele bitcoin. Ambele tranzacții folosesc scripturi multisemnătură (2-din-3) pentru a demonstra modul în care scripturile complexe cresc impactul martorului segregat asupra comisioanelor. Să presupunem un comision de tranzacție de 30 satoshi pe octet și o reducere de 75% pentru datele martor:

#### Fără Martor Segregat

Comision tranzacție A: 25.710 satoshi  
Comision tranzacție B: 18.990 satoshi

#### Cu Martor Segregat

Comision tranzacție A: 8.130 satoshi  
Comision tranzacție B: 12,045 satoshi

Ambele tranzacții sunt mai puțin costisitoare atunci când este folosit martorul segregat. Dar, comparând costurile dintre cele două tranzacții, vedem că înainte de Martor Segregat, comisionul este mai mare pentru tranzacția care are un UTXO-nou-Net negativ. După Martor Segregat, comisioanele de tranzacție se aliniază cu stimulentul de a reduce crearea de noi UTXO-uri, fără a penaliza din greșeală tranzacțiile cu multe intrări.

Martorul Segregat are, prin urmare, două efecte principale asupra comisioanelor plătite de utilizatorii bitcoin. În primul rând, segwit reduce costul general al tranzacțiilor prin reducerea

datelor martor și creșterea capacitatei lanțului-de-blocuri bitcoin. În al doilea rând, reducerea datelor martor corectează o aliniere greșită a stimulentelor care ar putea să creeze din neatenție mai multe date în setul UTXO.

## Rețeaua Bitcoin

### Arhitectura de rețea De-la-Egal-la-Egal

Bitcoin este structurat cu o arhitectură de rețea de-la-egal-la-egal peste internet. Termenul de-la-egal-la-egal (peer-to-peer), sau P2P, înseamnă că toate calculatoarele care participă la rețea sunt egale unul cu celălalt, că sunt toate egale, că nu există noduri "speciale" și că toate nodurile împart sarcina furnizării serviciilor de rețea. Nodurile de rețea se interconectează într-o rețea celulară cu o topologie "plată". Nu există vreun server, nici un serviciu centralizat și nici o ierarhie în rețea. Nodurile dintr-o rețea P2P furnizează și consumă servicii în același timp, reciprocitatea fiind stimulentul pentru participare. Rețelele P2P sunt, în mod inerent, rezistente, descentralizate și deschise. Un exemplu preeminent al unei arhitecturi de rețea P2P a fost internetul timpuriu, unde nodurile din rețeaua IP erau egale. Arhitectura de internet de astăzi este mai ierarhică, dar Protocolul Internet își păstrează încă esența de topologie plată. Dincolo de bitcoin, cea mai mare și de succes aplicație a tehnologiilor P2P este schimbul de fișiere, cu Napster fiind pionier și BitTorrent ca fiind cea mai recentă evoluție a arhitecturii.

Arhitectura de rețea P2P bitcoin este mult mai mult decât o alegere topologică. Bitcoin este prin design un sistem de numerar digital P2P, iar arhitectura rețelei este atât o reflecție cât și o fundație a acestei caracteristici de bază. Descentralizarea controlului este un principiu de proiectare de bază care poate fi atins și menținut doar printr-o rețea de consens P2P descentralizată.

Termenul "rețea bitcoin" se referă la colecția de noduri care rulează protocolul P2P bitcoin. Pe lângă protocolul P2P bitcoin, există și alte protocoale, cum ar fi Stratum, care sunt utilizate pentru minerit și portofele ușoare sau mobile. Aceste protocoale suplimentare sunt furnizate de servere de rutare gateway care acceseză rețeaua bitcoin folosind protocolul P2P bitcoin și apoi extind acea rețea la nodurile care rulează alte protocoale. De exemplu, serverele Stratum conectează nodurile de minerit Stratum prin protocolul Stratum la rețeaua principală bitcoin și creează puncte de la protocolul Stratum la protocolul P2P bitcoin. Folosim termenul "rețea bitcoin extinsă" pentru a ne referi la rețeaua globală care include protocolul P2P bitcoin, protocoalele bazinelor de minerit, protocolul Stratum și orice alte protocoale conexe care conectează componentele sistemului bitcoin.

### Tipuri de Noduri și Roulurile lor

Deși nodurile din rețeaua P2P bitcoin sunt egale, ele pot avea diferite roluri depinzând de funcționalitatea pe care o susțin. Un nod bitcoin este o colecție de funcții: rutare, bază de date lanț-de-blocuri, minerit și servicii pentru portofel. Un nod complet cu toate aceste patru funcții este prezentat în [Un nod de rețea bitcoin cu toate cele patru funcții: portofel, miner, bază de date completă lanț-de-blocuri și rutare de rețea](#).

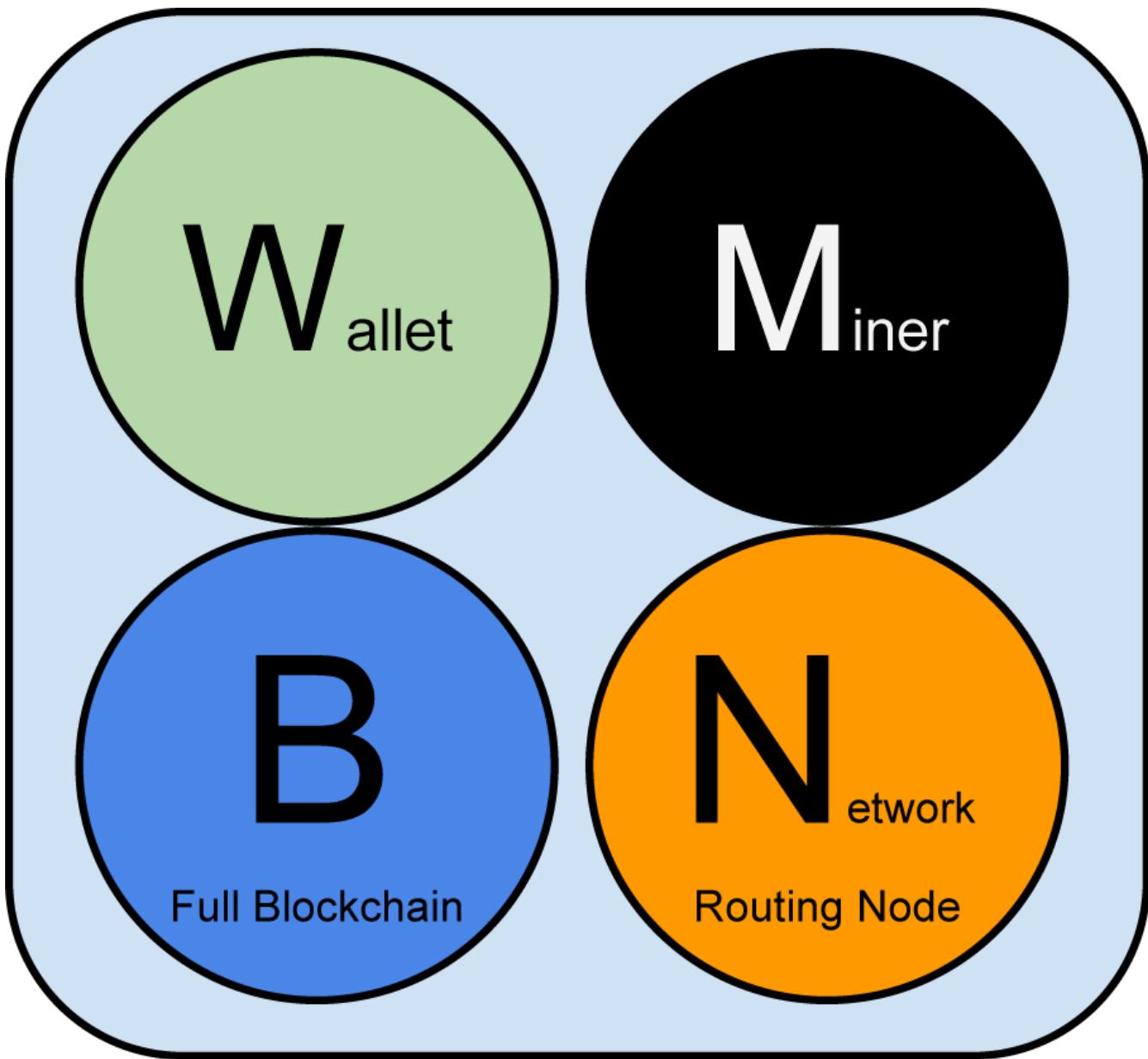


Figure 49. Un nod de rețea bitcoin cu toate cele patru funcții: portofel, miner, bază de date completă lanț-de-blocuri și rutare de rețea

Toate nodurile includ funcția de rutare pentru a participa la rețea și ar putea include și alte funcționalități. Toate nodurile validează și propagă tranzacțiile și blocurile, și descoperă și mențin conexiunile cu alte noduri. În exemplul cu noduri complete din [Un nod de rețea bitcoin cu toate cele patru funcții: portofel, miner, bază de date completă lanț-de-blocuri și rutare de rețea](#), funcția de rutare este indicată printr-un cerc numit "Nod Rețea de Rutare" (Network Routing Node) sau cu litera "N".

Unele noduri, numite noduri complete, mențin, de asemenea, o copie completă și actualizată a lanțului-de-blocuri. Nodurile complete pot verifica în mod autonom și autoritar orice tranzacție fără referință externă. Unele noduri mențin doar un subset al lanțului-de-blocuri și verifică tranzacțiile folosind o metodă numită *verificare simplificată a platii* (simplified payment verification) sau SPV. Aceste noduri sunt cunoscute sub numele de noduri SPV sau noduri suple (lightweight). În exemplul cu noduri complete din figură, funcția bazei de date lanț-de-blocuri cu nod-complet este indicată de un cerc numit "lanț-de-blocuri complet" (Full Blockchain) sau litera "B." În [Rețeaua extinsă bitcoin care prezintă diferite tipuri de noduri, gateway-uri și protocoale](#), nodurile SPV sunt desenate fără cercul "B", arătând că nu au o copie completă a lanțului-de-blocuri.

Nodurile de minerit concurează pentru a crea blocuri noi prin rularea de hardware specializat pentru rezolvarea algoritmului Dovadă-de-Lucru. Unele noduri de minerit sunt, de asemenea, noduri complete, menținând o copie completă a lanțului-de-blocuri, în timp ce altele sunt noduri suple (lightweight) care participă la bazinul de minerit și depind de un server de bazin să mențină un nod complet. Funcția de minerit este prezentată în nodul complet sub forma unui cerc numit "Miner" sau litera "M".

Portofelele pentru utilizatori ar putea face parte dintr-un nod complet, aşa cum se întâmplă de obicei în cazul clientilor bitcoin desktop. Din ce în ce mai multe portofele pentru utilizatori, în special cele care rulează pe dispozitive cu restricții de resurse, cum ar fi smartphone-urile, sunt noduri SPV. Funcția portofel este prezentată în [Un nod de rețea bitcoin cu toate cele patru funcții: portofel, miner, bază de date completă lanț-de-blocuri și rutare de rețea](#) ca un cerc numit "Portofel" (Wallet) sau litera "W".

În plus față de principalele tipuri de noduri din protocolul P2P bitcoin, există servere și noduri care rulează alte protocoale, cum ar fi protocoale specializate pentru bazin de minerit și protocoale suple (lightweight) pentru accesul clientilor.

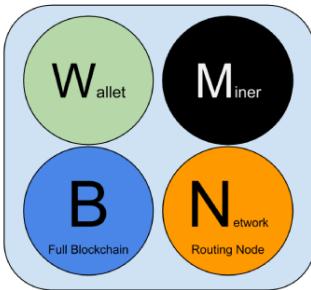
[Diferite tipuri de noduri în rețeaua bitcoin extinsă](#) arată cele mai frecvente tipuri de noduri din rețeaua bitcoin extinsă.

## Rețeaua Bitcoin Extinsă

Rețeaua principală bitcoin, care rulează protocolul P2P bitcoin, constă între 5.000 și 8.000 de noduri de ascultare care rulează diverse versiuni ale clientului referință bitcoin (Bitcoin Core) și câteva sute de noduri care rulează diverse alte implementări ale protocolului P2P bitcoin, precum Bitcoin Classic, Bitcoin Unlimited, BitcoinJ, Libbitcoin, btcd și bcoin. Un procent mic din nodurile din rețeaua P2P bitcoin sunt, de asemenea, noduri de minerit, care concurează în procesul de minare, validând tranzacțiile și creând noi blocuri. Diverse companii mari interfațează cu rețeaua bitcoin rulând clienti cu noduri complete bazate pe clientul Bitcoin Core, cu copii complete ale lanțului-de-blocuri și un nod de rețea, dar fără funcții de minerit sau de portofel. Aceste noduri acționează ca routere de margine de rețea, permitând să se construiască deasupra lor alte servicii (burse-de-schimb, portofele, exploratori de blocuri, procesarea plășilor pentru comercianți).

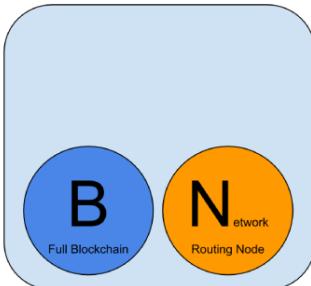
Rețeaua extinsă bitcoin include rețeaua care rulează protocolul bitcoin P2P, descris mai devreme, precum și noduri care rulează protocoale specializate. În rețeaua P2P principală bitcoin sunt atașate o serie de servere de bazin și gateway-uri de protocol care conectează noduri care rulează alte protocoale. Aceste alte noduri de protocol sunt în mare parte noduri de bazin de minerit (a se vedea [Minerit și Consens](#)) și clienti portofel supli, care nu țin o copie completă a lanțului-de-blocuri.

[Rețeaua extinsă bitcoin care prezintă diferite tipuri de noduri, gateway-uri și protocoale](#) arată rețeaua bitcoin extinsă cu diferitele tipuri de noduri, servere gateway, routere de margine și clienti portofel și diferitele protocoale pe care le folosesc pentru a se conecta între ele.



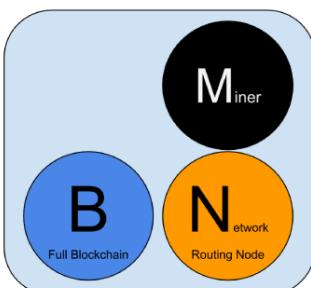
## Reference Client (Bitcoin Core)

Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.



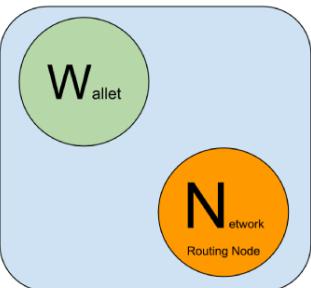
## Full Block Chain Node

Contains a full Blockchain database, and Network routing node on the bitcoin P2P network.



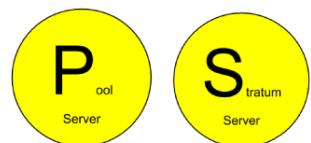
## Solo Miner

Contains a mining function with a full copy of the blockchain and a bitcoin P2P network routing node.



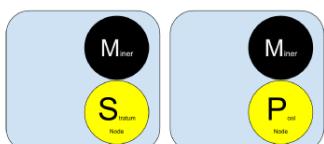
## Lightweight (SPV) wallet

Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.



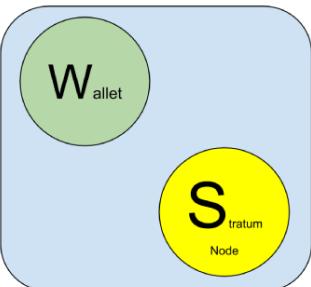
## Pool Protocol Servers

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.

Figure 50. Diferite tipuri de noduri în rețeaua bitcoin extinsă

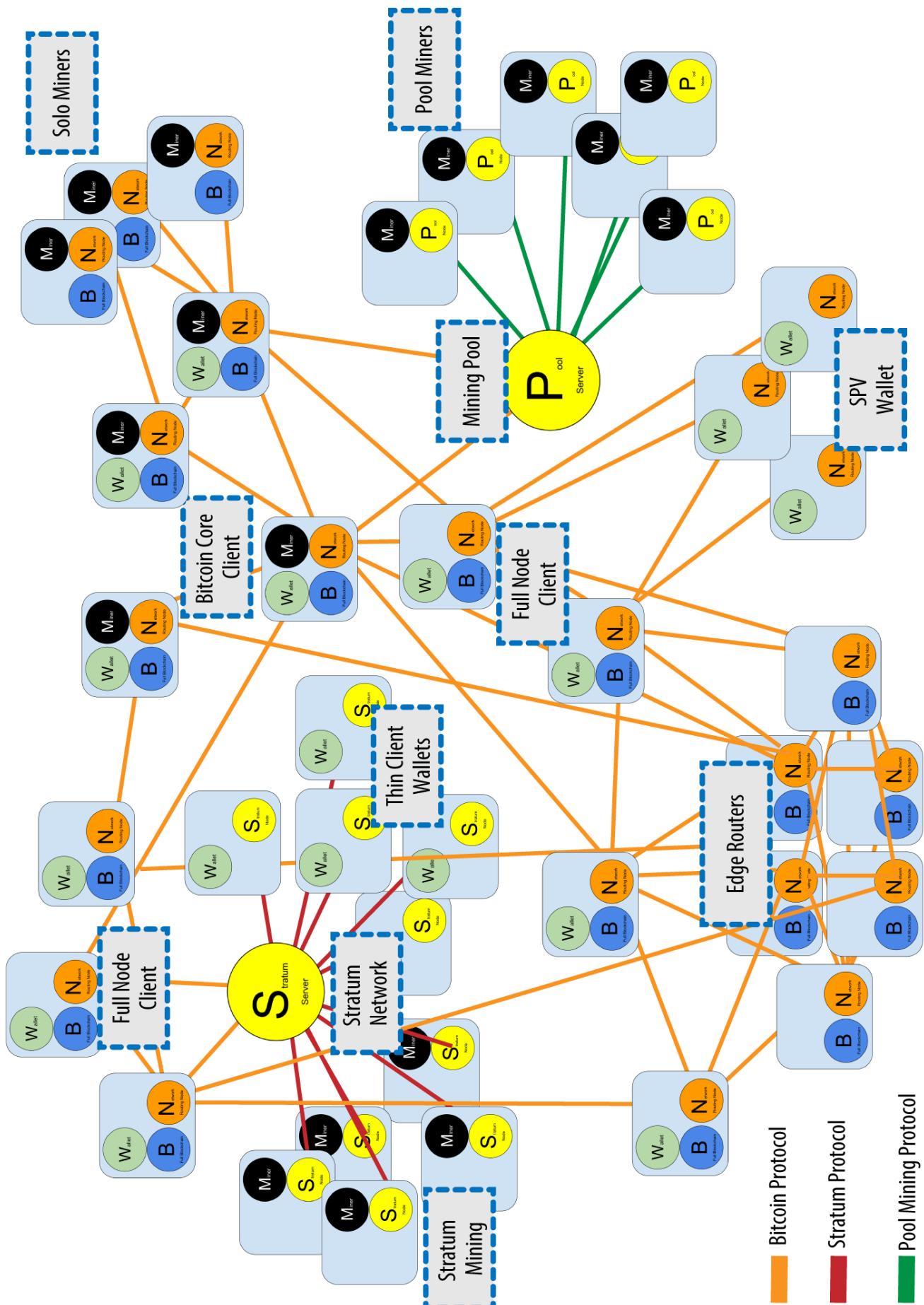


Figure 51. Rețeaua extinsă bitcoin care prezintă diferite tipuri de noduri, gateway-uri și protocoale

## Rețelele Releu Bitcoin

În timp ce rețeaua P2P bitcoin răspunde nevoilor generale ale unei largi varietăți de tipuri de noduri, prezintă o latență de rețea prea mare pentru nevoile specializate ale nodurilor de minerit.

Minerii Bitcoin sunt angajați într-o competiție contra-cronometru pentru a rezolva problema Dovadă-de-Lucru și a extinde lanțul-de-blocuri (vezi [Minerit și Consens](#)). În timp ce participă la această competiție, minerii bitcoin trebuie să reducă la minimum timpul dintre propagarea unui bloc câștigător și începutul rundei următoare a competiției. În domeniul mineritului, latența rețelei este direct legată de marjele de profit.

O *Rețea Releu Bitcoin* este o rețea care încearcă să minimizeze latența în transmiterea blocurilor între mineri. Prima *Rețea Releu Bitcoin* a fost creată de dezvoltatorul core Matt Corallo în 2015 pentru a permite sincronizarea rapidă a blocurilor între mineri cu latență foarte scăzută. Rețeaua era formată din mai multe noduri specializate găzduite pe infrastructura Amazon Web Services din întreaga lume și servea la conectarea majorității minerilor și bazinelor de minerit.

Rețeaua Releu Bitcoin inițială a fost înlocuită în 2016 prin introducerea *Fast Internet Bitcoin Relay Engine* sau [FIBRE](#), creată de asemenea de dezvoltatorul core Matt Corallo. FIBER este o rețea de relee bazată pe UDP, care transmite blocuri în cadrul unei rețele de noduri. FIBER implementează o optimizare a *blocului compact* pentru a reduce și mai mult cantitatea de date transmise și latența rețelei.

O altă rețea de relee (încă în faza de propunere) este [Falcon](#), bazată pe cercetări făcute la Universitatea Cornell. Falcon folosește "cut-through-routing" în loc de "store-and-forward" pentru a reduce latența prin propagarea părților din blocuri, aşa cum sunt primite, mai degrabă decât a șteptarea până la primirea unui bloc complet.

Rețelele de relee nu sunt un înlocuitor pentru rețeaua P2P bitcoin. În schimb, sunt rețele de suprapunere care asigură conectivitate suplimentară între nodurile cu nevoi specializate. La fel cum autostrăzile nu sunt înlocuitori pentru drumurile rurale, ci mai degrabă scurtături între două puncte cu trafic intens, încă mai aveți nevoie de drumuri mici pentru a vă conecta la autostrăzi.

## Descoperirea Rețelei

Când un nou nod începe, trebuie să descopere alte noduri bitcoin din rețea pentru a putea participa. Pentru a începe acest proces, un nou nod trebuie să descopere cel puțin un alt nod existent în rețea și să se conecteze la acesta. Locația geografică a altor noduri este irelevantă; topologia rețelei bitcoin nu este definită geografic. Prin urmare, orice noduri bitcoin existente pot fi selectate la întâmplare.

Pentru a se conecta la un nod cunoscut, nodurile stabilesc o conexiune TCP, de obicei la portul 8333 (portul cunoscut în general ca cel folosit de bitcoin), sau la un port alternativ, dacă este furnizat. La stabilirea unei conexiuni, nodul va porni o "strângere de mâna" (a se vedea [Strângerea inițială de mâna între noduri \(semeni\)](#)) prin transmiterea unui mesaj de *versiune*, care conține informații de identificare, incuzând:

## **nVersion**

Versiunea protocolului P2P bitcoin pe care clientul o "vorbește" (de exemplu, 70002)

## **nLocalServices**

O listă de servicii locale acceptate de nod, în prezent doar **NODE\_NETWORK**

## **nTime**

Data și ora curentă

## **addrYou**

Adresa IP a celuilalt nod, din perspectiva nodului curent

## **addrMe**

Adresa IP a nodului local, descoperită de nodul local

## **subver**

O sub-versiune care arată tipul de software care rulează pe acest nod (de exemplu, [/Satoshi:0.9.2.1/](#))

## **BestHeight**

Înălțimea blocului din lanțul-de-blocuri al acestui nod

(Consultați [GitHub](#) pentru un exemplu de mesaj de *versiune*.)

Mesajul de *versiune* este întotdeauna primul mesaj trimis de orice nod către un alt nod. Nodul local care primește un mesaj de *versiune* va examina versiunea **nVersion** raportată de la distanță și va decide dacă nodul de la distanță este compatibil. Dacă nodul de la distanță este compatibil, nodul local va recunoaște mesajul de *versiune* și va stabili o conexiune prin trimiterea unui **verack**.

Cum găsește un nou nod alte noduri (semeni)? Prima metodă constă în interogarea DNS folosind o serie de "semințe DNS" (DNS seeds), care sunt servere DNS care furnizează o listă de adrese IP a nodurilor bitcoin. Unele dintre aceste semințe DNS oferă o listă statică de adrese IP a nodurilor stabile de ascultare bitcoin. Unele dintre semințele DNS sunt implementări personalizate ale BIND (Berkeley Internet Name Daemon) care returnează un subset aleatoriu dintr-o listă de adrese de noduri bitcoin colectate de un crawler sau un nod bitcoin care rulează de multă vreme. Clientul Bitcoin Core conține numele a cinci semințe DNS diferite. Diversitatea proprietarilor și diversitatea implementării diferitelor semințe DNS oferă un nivel ridicat de fiabilitate pentru procesul inițial de pornire. În clientul Bitcoin Core, opțiunea de a utiliza semințele DNS este controlată de opțiunea **-dnsseed** (setată la 1 în mod implicit, pentru a utiliza sămânța DNS).

În mod alternativ, unui nod care tocmai a pornit și nu știe nimic din rețea trebuie să i se ofere adresa IP a cel puțin unui nod bitcoin, după care poate stabili conexiuni prin prezentări ulterioare. Argumentul din linia de comandă **-seednode** poate fi utilizat pentru a vă conecta la un singur nod doar pentru prezentări care îl utilizează ca o sămânță. După ce nodul de sămânță inițial este utilizat pentru a forma prezentări, clientul se va deconecta de la acesta și va folosi nodurile nou-descoperite.

# Node A

# Node B

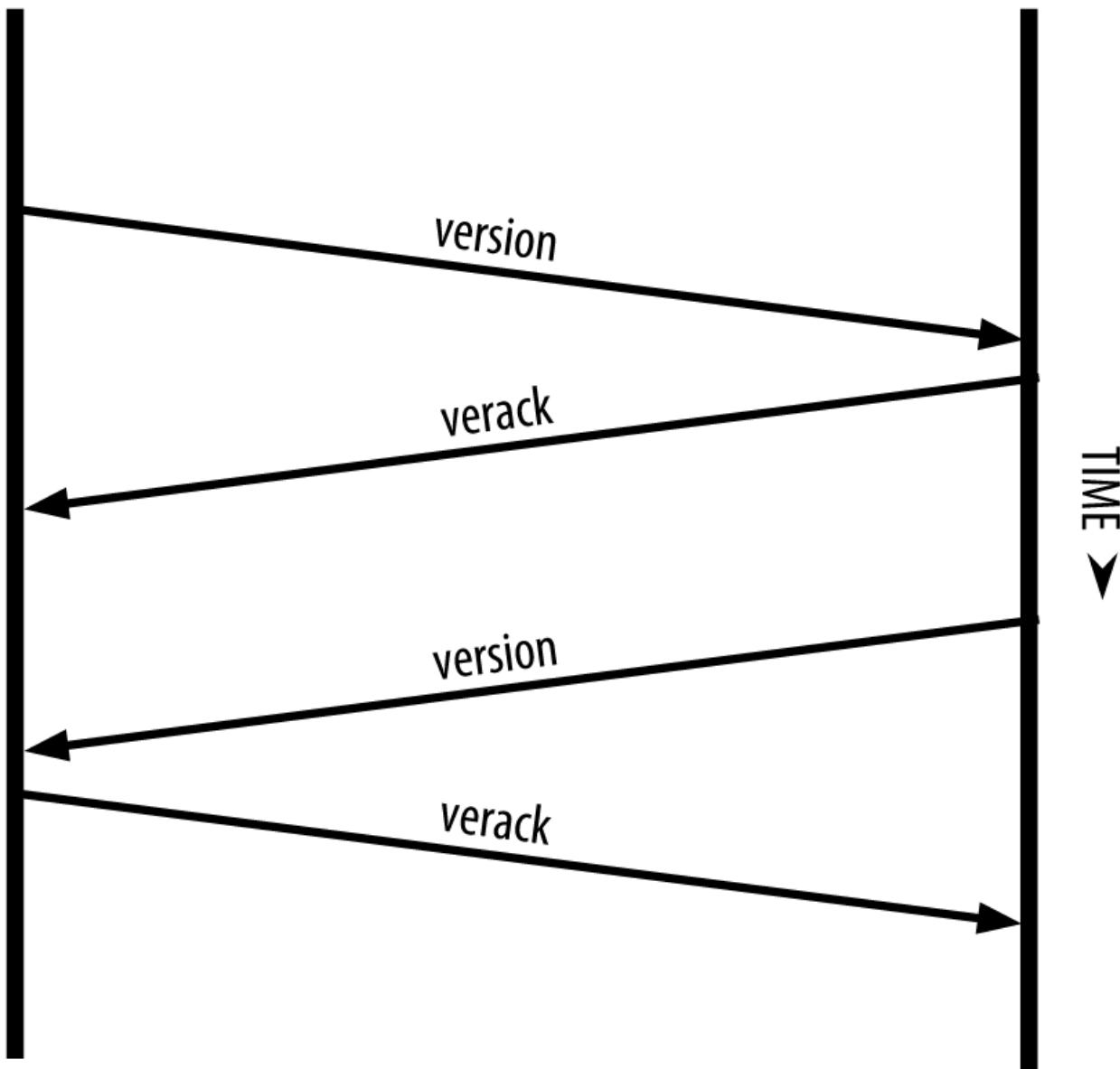


Figure 52. Strângerea inițială de mâna între noduri (semeni)

Odată ce una sau mai multe conexiuni sunt stabilite, noul nod va trimite un mesaj **addr** care conține propria adresă IP către vecinii săi. La rândul lor, vecinii vor transmite mesajul **addr** către vecinii lor, asigurându-se că nodul nou conectat devine bine cunoscut și mai bine conectat. În plus, nodul nou conectat poate trimite **getaddr** către vecini, cerându-le să returneze o listă de adrese IP ale altor semeni. În acest fel, un nod poate găsi semeni (peers) pentru a se conecta și pentru a face cunoscută existența sa în rețea pentru ca alte noduri să îl găsească. **Propagarea și descoperirea adresei** arată protocolul de descoperire a adresei.

# Node A

# Node B

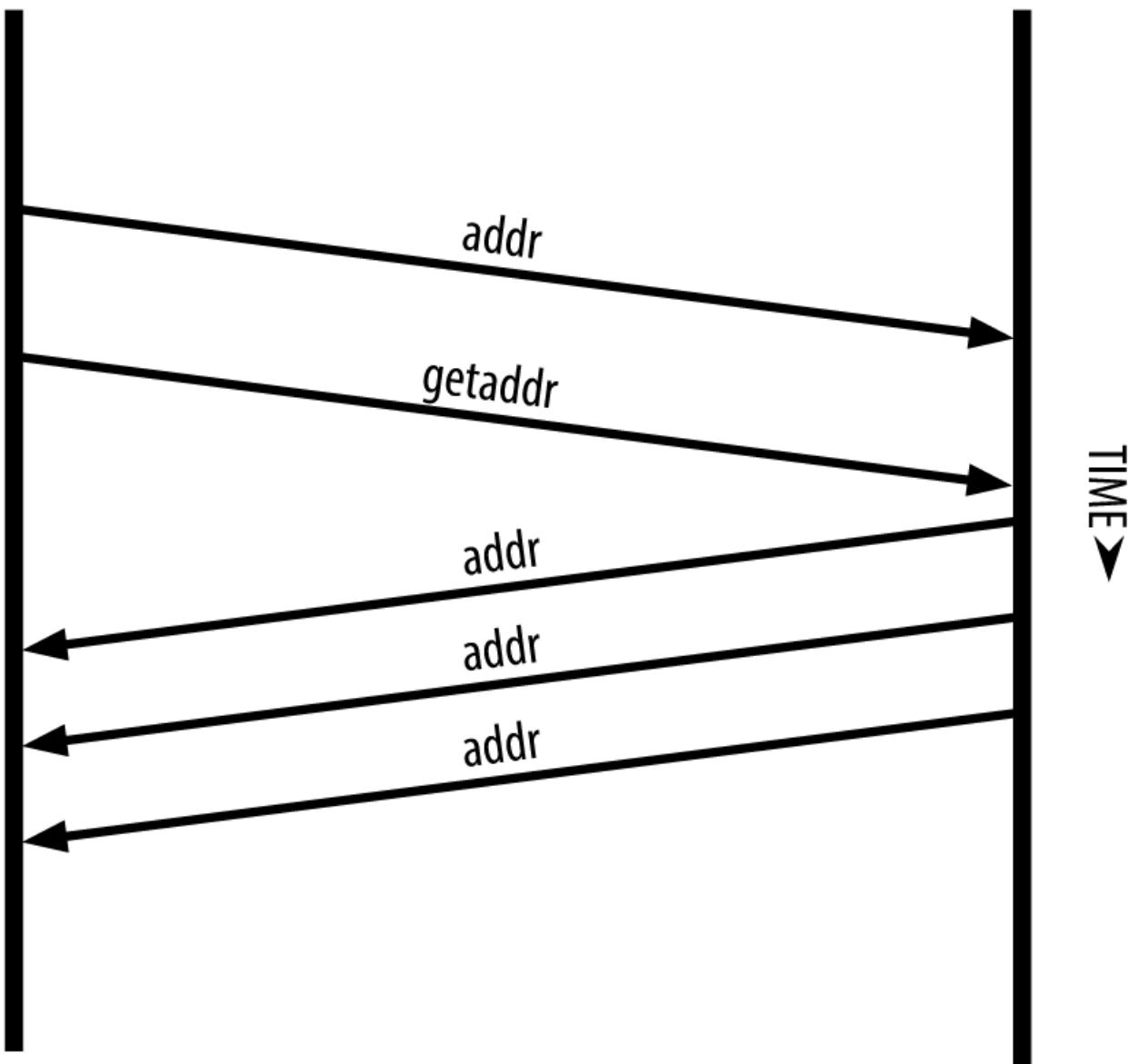


Figure 53. Propagarea și descoperirea adresei

Un nod trebuie să se conecteze la câțiva semenii (peers) diferiți pentru a stabili diverse căi (rute) în rețeaua bitcoin. Căile nu sunt persistente - nodurile vin și pleacă - și astfel nodul trebuie să continue să descopere noi noduri, deoarece pierde conexiunile vechi, precum și să ajute alte noduri atunci când pornesc. O singură conexiune este necesară pentru pornirea inițială, deoarece primul nod poate oferi prezentări către semenii săi și acei semenii pot oferi prezentări suplimentare. De asemenea, este inutil și este o risipă de resurse de rețea să vă conectați la mai mult de o mână de noduri. După pornirea inițială, un nod își va aminti cele mai recente conexiuni cu semenii, astfel încât, dacă este repornit, poate restabili rapid conexiunile cu rețeaua sa anterioară. Dacă niciunul dintre foștii semenii nu răspunde la solicitarea de conectare a acestuia, nodul poate utiliza nodurile de sămânță pentru a porni din nou.

Pe un nod care rulează clientul Bitcoin Core, puteți enumera conexiunile cu semenii (peer) cu comanda `getpeerinfo`:

```
$ 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  
  }  
]
```

Pentru a nu ține cont de gestionarea automată a semenilor (peers) și pentru a specifica o listă de adrese IP, utilizatorii pot oferi opțiunea **-connect=<IPAddress>** și să specifică una sau mai multe adrese IP. Dacă se utilizează această opțiune, nodul se va conecta numai la adresele IP selectate, în loc să descopere și să mențină automat conexiunile cu semenii.

Dacă nu există trafic pe o conexiune, nodurile vor trimite periodic un mesaj pentru a menține conexiunea. Dacă un nod nu a comunicat pe o conexiune timp de mai mult de 90 de minute, se presupune că este deconectat și se va căuta un alt seamăn (peer). Astfel, rețeaua se adaptează dinamic la nodurile tranzitorii și la problemele de rețea și poate crește și micșora organic, după cum este necesar, fără niciun control central.

# Noduri Complete

Nodurile complete sunt noduri care mențin un lanț-de-blocuri complet cu toate tranzacțiile. Mai exact, probabil că ar trebui numiți "noduri lanț-de-blocuri complet". În primii ani de bitcoin, toate nodurile erau noduri complete, iar în prezent clientul Bitcoin Core este un nod complet lanț-de-blocuri. În ultimii doi ani, cu toate acestea, au fost introduse noi forme de clienți bitcoin care nu mențin un lanț-de-blocuri complet, ci funcționează ca și clienți supli (lightweight). Le vom examina mai detaliat în secțiunea următoare.

Nodurile lanț-de-blocuri complete mențin o copie completă și la zi a lanțului-de-blocuri bitcoin cu toate tranzacțiile, pe care îl construiesc și îl verifică în mod independent, începând chiar de la primul bloc (blocul geneză) și construind până la cel mai recent bloc cunoscut din rețea. Un nod lanț-de-blocuri complet poate verifica în mod independent și autoritar orice tranzacție fără a recurge sau a depinde de orice alt nod sau sursă de informații. Nodul lanț-de-blocuri complet se bazează pe rețea pentru a primi actualizări despre noile blocuri de tranzacții, pe care apoi le verifică și le încorporează în copia sa locală a lanțului-de-blocuri.

Rularea unui nod lanț-de-blocuri complet va oferă experiența bitcoin pură: verificarea independentă a tuturor tranzacțiilor fără a fi nevoie să vă bazați sau să aveți încredere în orice alte sisteme. Este ușor să vă dați seama dacă rulați un nod complet, deoarece necesită mai mult de o sută de GB de spațiu pe disc pentru a stoca lanțul-de-blocuri complet. Dacă aveți nevoie de mult spațiu pe disc și durează două-trei zile pentru a vă sincroniza în rețea, atunci execuți un nod complet. Aceasta este prețul independenței complete și al libertății față de o autoritate centrală.

Există câteva implementări alternative ale clienților bitcoin cu lanț-de-blocuri complet, construite folosind diferite limbaje de programare și arhitecturi software. Cu toate acestea, cea mai comună implementare este clientul de referință Bitcoin Core, cunoscut și sub numele de client Satoshi. Peste 75% din nodurile din rețea bitcoin rulează diverse versiuni ale Bitcoin Core. Este identificat ca "Satoshi" în textul sub-versiunii trimis în mesajul de *versiune* și afișat de comanda `getpeerinfo` așa cum am văzut anterior; de exemplu, [/Satoshi:0.8.6/](#).

## Schimbul de "Inventar"

Primul lucru pe care îl va face un nod complet odată ce se conectează la semenii (peers) este să încerce să construiască un lanț-de-blocuri complet. Dacă este un nod complet nou și nu are deloc un lanț-de-blocuri, atunci cunoaște un singur bloc, blocul geneză, care este încorporat static în software-ul client. Începând cu blocul nr. 0 (blocul geneză), noul nod va trebui să descarce sute de mii de blocuri pentru a se sincroniza cu rețea și pentru a restabili întregul lanț-de-blocuri.

Procesul de sincronizare a lanțului-de-blocuri începe cu mesajul de *versiune*, deoarece acesta conține `BestHeight`, înățimea curentă a lanțului-de-blocuri pentru un nod (număr de blocuri). Un nod va vedea mesajele de *versiune* de la semenii săi, va ști câte blocuri au fiecare și va putea compara cu câte blocuri are în propriul său lanț-de-blocuri. Nodurile pereche vor schimba un mesaj `getblocks` care conține rezumatul (amprenta digitală) a blocului superior de pe lanțul-de-blocuri local. Unul dintre semenii (peers) va putea identifica rezumatul primit ca aparținând unui bloc care nu se află în vârf, ci mai degrabă aparține unui bloc mai vechi, deducând astfel că propriul său lanț-de-blocuri local este mai lung decât cel al semenilor săi.

Nodul care are lanțul-de-blocuri mai lung are mai multe blocuri decât celălalt nod și poate identifica de care blocuri are nevoie celălalt nod pentru a ”ajunge din urmă”. Aceasta va identifica primele 500 de blocuri pentru a partaja și transmite rezumatul lor folosind un mesaj `inv` (inventar). Nodul căruia îi lipsesc o parte din aceste blocuri le va recupera, prin emiterea unei serii de mesaje `getdata` care solicită datele complete ale blocului și identificând blocurile solicitate folosind rezumatele din mesajul `inv`.

Să presupunem, de exemplu, că un nod are doar blocul geneză. Aceasta va primi apoi un mesaj `inv` de la semenii săi care conține rezumatele următoarelor 500 de blocuri din lanț. Aceasta va începe să solicite blocuri de la toți semenii săi conectați, distribuind sarcina și asigurându-se că nu va copleși niciun seamăn cu solicitări. Nodul urmărește câte blocuri sunt ”în tranzit” pe conexiunea de la semenii, adică blocuri pe care le-a solicitat, dar nu le-a primit, verificând că nu depășește o anumită limită (`MAX_BLOCKS_IN_TRANSIT_PER_PEER`). În acest fel, dacă are nevoie de foarte multe blocuri, va trimite noi solicitări doar pe măsură ce solicitările anterioare sunt îndeplinite, permitând colegilor să controleze ritmul actualizărilor și să nu coplesească rețeaua. Pe măsură ce fiecare bloc este primit, acesta este adăugat la lanțul-de-blocuri, așa cum vom vedea în [Lanțul-de-Blocuri](#). Pe măsură ce lanțul-de-blocuri local este construit, sunt solicitate și primește mai multe blocuri, iar procesul continuă până când nodul se apropi de restul rețelei.

Acest proces de comparare a lanțului-de-blocuri local cu semenii și de a recupera blocurile lipsă se întâmplă de fiecare dată când un nod se deconectează pentru o perioadă oarecare de timp. Indiferent dacă un nod a fost deconectat timp de câteva minute și lipsesc câteva blocuri, sau o lună și lipsesc câteva mii de blocuri, începe prin a trimite `getblocks`, primește un răspuns `inv` și începe să descarce blocurile care lipsesc. [Nod care își sincronizează lanțul-de-blocuri prin preluarea blocurilor de la un seamăn](#) arată protocolul de propagare a inventarului și a blocurilor.

**Node A**

**Node B**

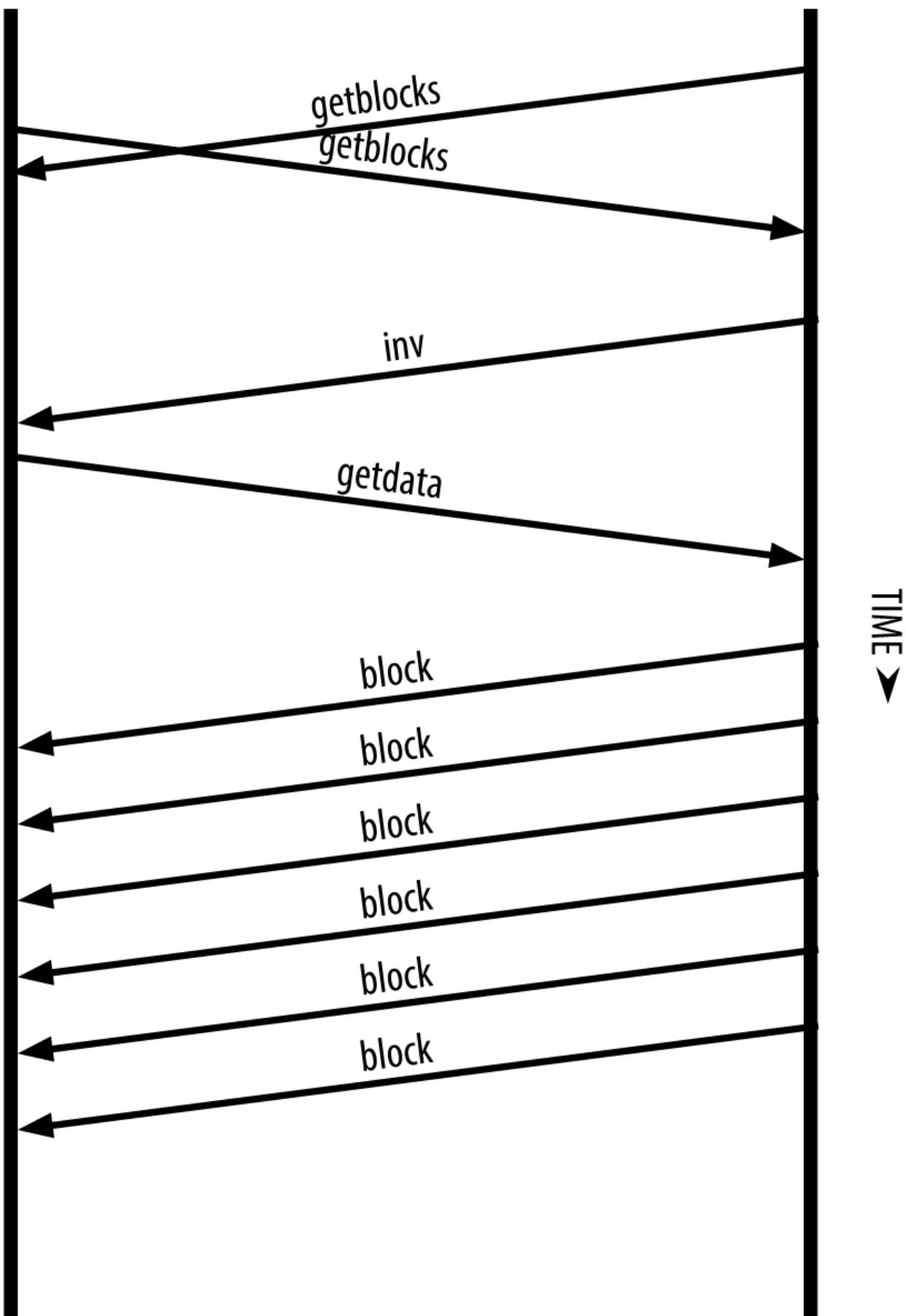


Figure 54. Nod care își sincronizează lanțul-de-blocuri prin preluarea blocurilor de la un seamăn

# Noduri de Verificare Simplificată a Plății (SPV)

Nu toate nodurile au capacitatea de a stoca lanțul-de-blocuri complet. Mulți clienți bitcoin sunt proiectați pentru a rula pe dispozitive cu spațiu și putere limitată, cum ar fi smartphone-uri, tablete sau sisteme încorporate. Pentru astfel de dispozitive, se utilizează o metodă *simplificată de verificare a plății* (simplified payment verification - SPV) pentru a le permite să funcționeze fără a stoca lanțul-de-blocuri complet. Aceste tipuri de clienți se numesc clienți SPV sau clienți supli. Pe măsură ce adoptarea bitcoin crește, nodul SPV devine cea mai frecventă formă de nod bitcoin, în special pentru portofelele bitcoin.

Nodurile SPV descarcă doar anteturile blocurilor și nu descarcă tranzacțiile incluse în fiecare bloc. Lanțul rezultat de blocuri, fără tranzacții, este de 1.000 de ori mai mic decât lanțul-de-blocuri complet. Nodurile SPV nu pot construi o imagine completă a tuturor UTXO-urilor disponibile pentru cheltuire, deoarece nu au informații despre toate tranzacțiile din rețea. Nodurile SPV verifică tranzacțiile folosind o metodă ușor diferită care se bazează pe semeni pentru a oferi la cerere viziuni parțiale ale părților relevante ale lanțului-de-blocuri.

Ca o analogie, un nod complet este ca un turist într-un oraș străin, echipat cu o hartă detaliată a fiecărei străzi și a tuturor adreselor. Prin comparație, un nod SPV este ca un turist într-un oraș străin, care solicită străinilor indicații de orientare rând pe rând, în timp ce el stie doar de o stradă principală. Deși ambii turiști pot verifica existența unei străzi vizitând-o, turistul fără hartă nu știe ce se află pe vreuna dintre străzile laterale și nu știe ce alte străzi există. Situat în fața străzii Bisericii 23, turistul fără hartă nu poate ști dacă există alte zeci de adrese "Bisericii 23" în oraș și dacă aceasta este cea corectă. Cea mai bună șansă a unui turist fără hartă este să întrebe destui oameni și să speră că unii dintre ei nu încearcă să-l jefuiască.

SPV verifică tranzacțiile prin referire la *adâncimea* lor în lanțul-de-blocuri în loc de *înălțime*. În timp ce un nod lanț-de-blocuri complet va construi un lanț complet verificat de mii de blocuri și tranzacții care ajung până la (în timp) blocul geneză, un nod SPV va verifica lanțul tuturor blocurilor (dar nu toate tranzacțiile) și va lega acel lanț la tranzacția dorită.

De exemplu, atunci când examinăm o tranzacție în blocul 300.000, un nod complet leagă toate cele 300.000 de blocuri până la blocul geneză și construiește o bază de date completă a UTXO-urilor, stabilind validitatea tranzacției prin confirmarea că UTXO-ul a rămas necheltuit. Un nod SPV nu poate valida dacă UTXO-ul este necheltuit. În schimb, nodul SPV va stabili o legătură între tranzacție și blocul care o conține, folosind o *cale merkle* (merkle path, vezi [Arbore Merkle](#)). Apoi, nodul SPV așteaptă până când vede cele șase blocuri 300.001 - 300.006 adunate deasupra blocului care conține tranzacția și îl verifică prin stabilirea adâncimii sale sub blocurile 300.006 - 300.001. Faptul că alte noduri din rețea au acceptat blocul 300.000 și au depus apoi efortul necesar pentru a produce alte șase blocuri deasupra acestuia, dovedește, indirect, că tranzacția nu a fost cheltuită de două ori.

Un nod SPV nu poate fi convins că o tranzacție există într-un bloc atunci când tranzacția nu există. Nodul SPV stabilește existența unei tranzacții într-un bloc prin solicitarea unei dovezi de cale merkle și prin validarea Dovezii-de-Lucru în lanțul de blocuri. Cu toate acestea, existența unei tranzacții poate fi "ascunsă" de un nod SPV. Un nod SPV poate dovedi cu siguranță că există o tranzacție, dar nu poate verifica dacă o tranzacție, cum ar fi o cheltuire dublă a aceluiași UTXO, nu există deoarece nu are o înregistrare a tuturor tranzacțiilor. Această vulnerabilitate poate fi utilizată într-un atac denial-of-service sau pentru un atac de cheltuire dublă împotriva nodurilor.

SPV. Pentru a se apăra împotriva acestui lucru, un nod SPV trebuie să se conecteze aleatoriu la mai multe noduri, pentru a crește probabilitatea ca acesta să fie în contact cu cel puțin un nod sincer. Această necesitate de conectare aleatorie înseamnă că nodurile SPV sunt, de asemenea, vulnerabile la atacurile de partajare a rețelei sau atacurile Sybil, unde sunt conectate la noduri false sau rețele false și nu au acces la noduri oneste sau la adevărată rețea bitcoin.

În cele mai multe scopuri practice, nodurile SPV bine conectate sunt suficient de sigure, ajungând la un echilibru între nevoile de resurse, practicalitate și securitate. Cu toate acestea, pentru o securitate infailibilă, nimic nu bate rularea unui nod lanț-de-blocuri complet.

**TIP**

Un nod lanț-de-blocuri complet verifică o tranzacție inspectând întregul lanț de mii de blocuri de sub aceasta pentru a garanta că UTXO nu este cheltuită, în timp ce un nod SPV verifică cât de adânc este îngropat blocul, având o mână de blocuri deasupra lui.

Pentru a obține anteturile blocului, nodurile SPV folosesc un mesaj `getheaders` în loc de `getblocks`. Seamănul (peer) care răspunde va trimite până la 2.000 de anteturi de bloc folosind un singur mesaj `headers`. Procesul este același cu cel folosit de un nod complet pentru a prelua blocurile complete. Nodurile SPV stabilesc, de asemenea, un filtru pe conexiunea cu semenii, pentru a filtra fluxul de blocuri viitoare și tranzacții trimise de colegi. Orice tranzacție de interes este preluată folosind o cerere `getdata`. Seamănul (peer) generează un mesaj `tx` care conține tranzacțiile, ca răspuns. **Un nod SPV care sincronizează anteturile blocurilor** arată sincronizarea anteturilor blocurilor.

Deoarece nodurile SPV trebuie să obțină anumite tranzacții pentru a le verifica selectiv, ele creează, de asemenea, un risc pentru confidențialitate. Spre deosebire de nodurile lanț-de-blocuri complete, care colectează toate tranzacțiile din fiecare bloc, solicitările nodului SPV pentru date specifice pot dezvăluia din neatenție adresele din portofel. De exemplu, o treță parte care monitorizează o rețea ar putea ține evidența tuturor tranzacțiilor solicitate de un portofel pe un nod SPV și le poate folosi pentru a asocia adrese bitcoin cu utilizatorul portofelului, afectând confidențialitatea utilizatorului.

# Node A

# Node B

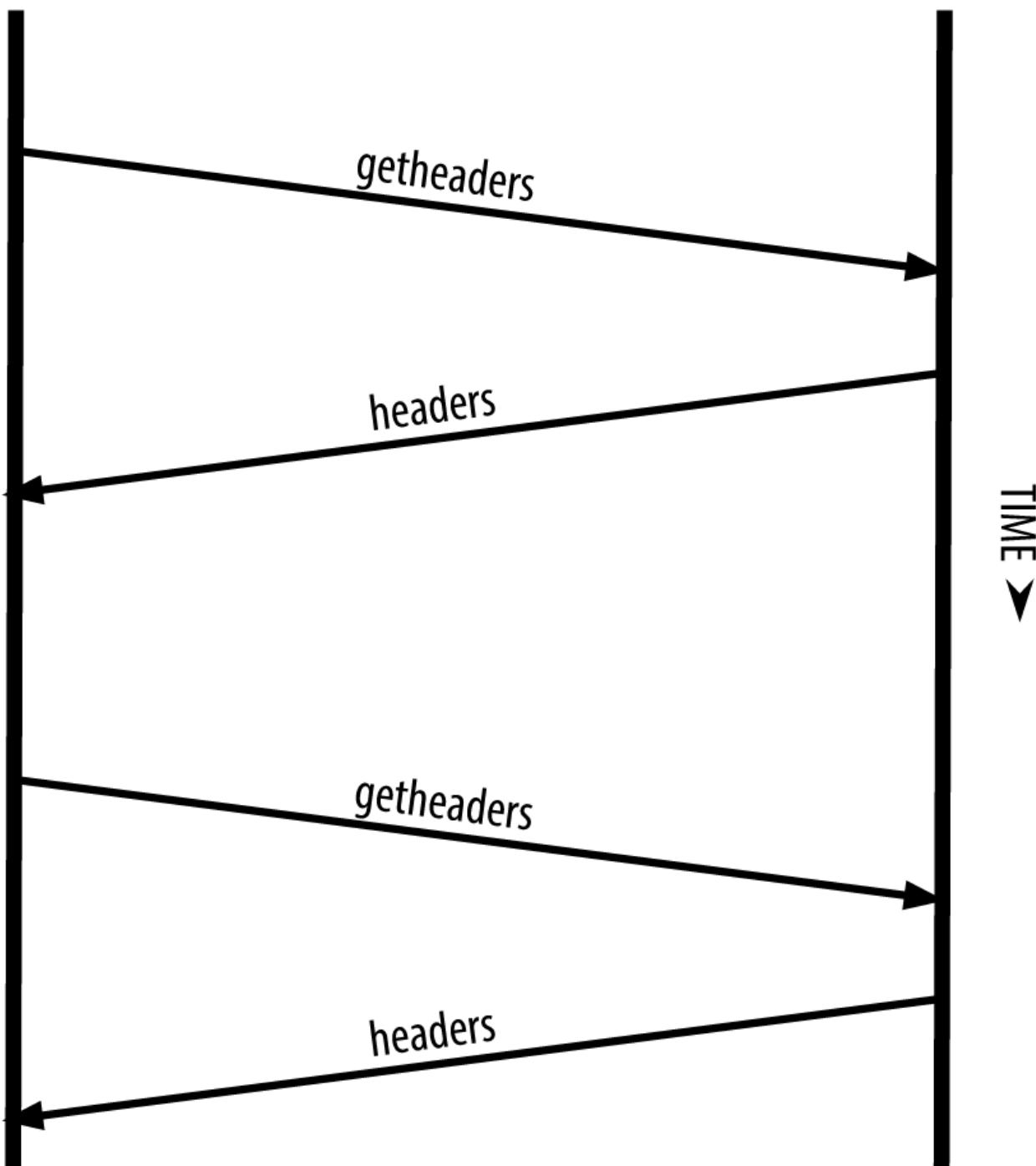


Figure 55. Un nod SPV care sincronizează anteturile blocurilor

La scurt timp după introducerea nodurilor SPV/suple, dezvoltatorii bitcoin au adăugat o caracteristică numită *filtru bloom* pentru a aborda riscurile de confidențialitate ale nodurilor SPV. Filtrele bloom permit nodurilor SPV să primească un subset de tranzacții, fără a dezvăluui cu exactitate care sunt adresele de care sunt interesate, printr-un mecanism de filtrare care folosește probabilități mai degrabă decât şabloane fixe.

## Filtre Bloom

Un filtru bloom este un filtru de căutare probabilistică, o modalitate de a descrie un şablon dorit

fără a-l specifica exact. Filtrele bloom oferă un mod eficient de a exprima un şablon de căutare în timp ce protejează confidențialitatea. Acestea sunt folosite de nodurile SPV pentru a le cere semenilor tranzacții care corespund unui tipar specific, fără a dezvăluui exact adresele, cheile sau tranzacțiile pe care le cauță.

În analogia noastră anterioară, un turist fără hartă cere indicații către o anumită adresă, "Str. Bisericii 23". Dacă cere străinilor indicații către această stradă, el își dezvăluie din greșeală destinația. Un filtru bloom este ca și cum ar întreba "Există străzi în acest cartier al căror nume se termină în  $I-C-I-I$ ?" O întrebare de genul acesta dezvăluie mult mai puțin despre destinația dorită decât să întrebe de "Str. Bisericii 23". Folosind această tehnică, un turist ar putea specifica adresa dorită cu mai multe detalii, cum ar fi "se termină cu  $R-I-C-I-I$ " sau mai puțin detaliat ca "se termină cu I". Modificând precizia căutării, turistul dezvăluie mai multe sau mai puține informații, în detrimentul obținerii unor rezultate mai mult sau mai puțin precise. Dacă întreabă folosind un tipar mai puțin specific, primește mult mai multe adrese posibile și o mai bună confidențialitate, dar multe dintre rezultate sunt irelevante. Dacă întreabă folosind un tipar foarte specific, obține mai puține rezultate, dar pierde confidențialitatea.

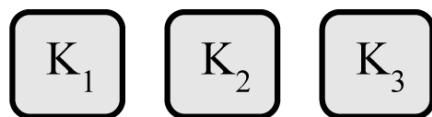
Filtrele bloom servesc această funcție permitând unui nod SPV să specifică un tipar de căutare pentru tranzacții care poate fi ajustat către precizie sau către confidențialitate. Un filtru mai specific va produce rezultate precise, dar în detrimentul dezvăluirii tiparelor de care nodul SPV este interesat, dezvăluind astfel adresele deținute de portofelul utilizatorului. Un filtru mai puțin specific va produce mai multe date despre mai multe tranzacții, multe irelevante pentru nod, dar va permite nodului să mențină o mai bună confidențialitate.

## **Cum Funcționează Filtrele Bloom**

Filtrele bloom sunt implementate ca un sir de dimensiune variabilă de  $N$  cifre binare și un număr variabil  $M$  de funcții de rezumare. Funcțiile de rezumare sunt proiectate pentru a produce întotdeauna o ieșire între 1 și  $N$ , corespunzând sirului de cifre binare. Funcțiile de rezumare sunt generate deterministic, astfel încât orice nod care implementează un filtru bloom va folosi întotdeauna aceleași funcții de rezumare și va obține aceleași rezultate pentru o intrare specifică. Prin alegerea diferitelor filtre de lungime ( $N$ ) și a unui număr diferit ( $M$ ) de funcții de rezumare, filtrul bloom poate fi reglat, variind nivelul de precizie și, prin urmare, confidențialitatea.

În [Un exemplu de filtru bloom simplist, cu un câmp de 16 biți și trei funcții de rezumare](#), folosim un sir foarte mic de 16 biți și un set de trei funcții de rezumare pentru a demonstra cum funcționeazăfiltrele bloom.

### 3 Hash Functions



### Hash Functions Output

1 to 16

### Empty Bloom Filter, 16 bit array

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	

Figure 56. Un exemplu de filtru bloom simplist, cu un câmp de 16 biți și trei funcții de rezumare

Filtrul bloom este inițializat astfel încât sirul de biți să fie pe zero. Pentru a adăuga un tipar la filtrul bloom, tiparul este rezumat de fiecare funcție de rezumare pe rând. Aplicând prima funcție de rezumare la intrare rezultă un număr cuprins între 1 și N. Bitul corespunzător din sir (indexat de la 1 la N) este găsit și setat la 1, înregistrând astfel ieșirea funcției de rezumare. Apoi, următoarea funcție de rezumare este folosită pentru a seta un alt bit și aşa mai departe. Odată ce toate M funcțiile de rezumare au fost aplicate, tiparul de căutare va fi "înregistrat" în filtrul bloom ca biți M care au fost schimbați de la 0 la 1.

[Adăugarea unui tipar "A" la filtrul bloom simplu](#) este un exemplu de adăugare a unui tipar "A" la filtrul simplu bloom prezentat în [Un exemplu de filtru bloom simplist, cu un câmp de 16 biți și trei funcții de rezumare](#).

Adăugarea unui al doilea tipar este la fel de simplă ca repetarea acestui proces. Tiparul este rezumat de fiecare funcție de rezumare pe rând, iar rezultatul este înregistrat prin setarea biților la 1. Rețineți că, pe măsură ce un filtru bloom este umplut cu mai multe tipare, rezultatul funcției de rezumare ar putea coincide cu un bit care este deja setat la 1, caz în care bitul nu este modificat. În esență, pe măsură ce mai multe tipare se înregistrează pe biți care se suprapun, filtrul bloom începe să fie saturat cu mai mulți biți setați la 1 și precizia filtrului scade. Aceasta este motivul pentru care filtrul este o structură de date probabilistică - devine mai puțin precis pe măsură ce se adaugă mai multe tipare. Precizia depinde de numărul de tipare adăugate față de dimensiunea sirului de biți (N) și de numărul funcțiilor de rezumare (M). Un sir de biți mai mare și mai multe funcții de rezumare pot înregistra mai multe tipare cu o precizie mai mare. Un sir de biți mai mic sau mai puține funcții de rezumare vor înregistra mai puține tipare și vor produce mai puțină precizie.

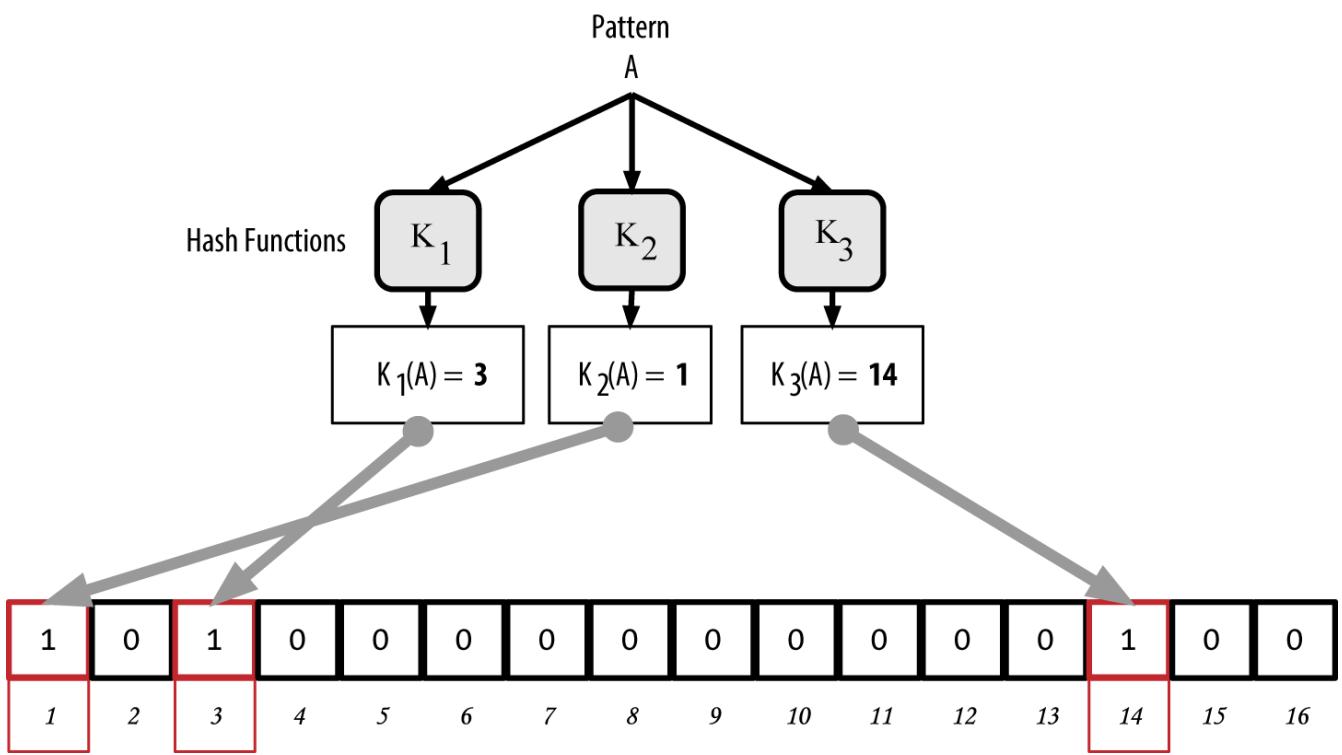


Figure 57. Adăugarea unui tipar "A" la filtrul bloom simplu

Adăugarea unui al doilea tipar "B" la filtrul bloom simplu este un exemplu de adăugare a unui al doilea tipar "B" la filtrul bloom simplu.

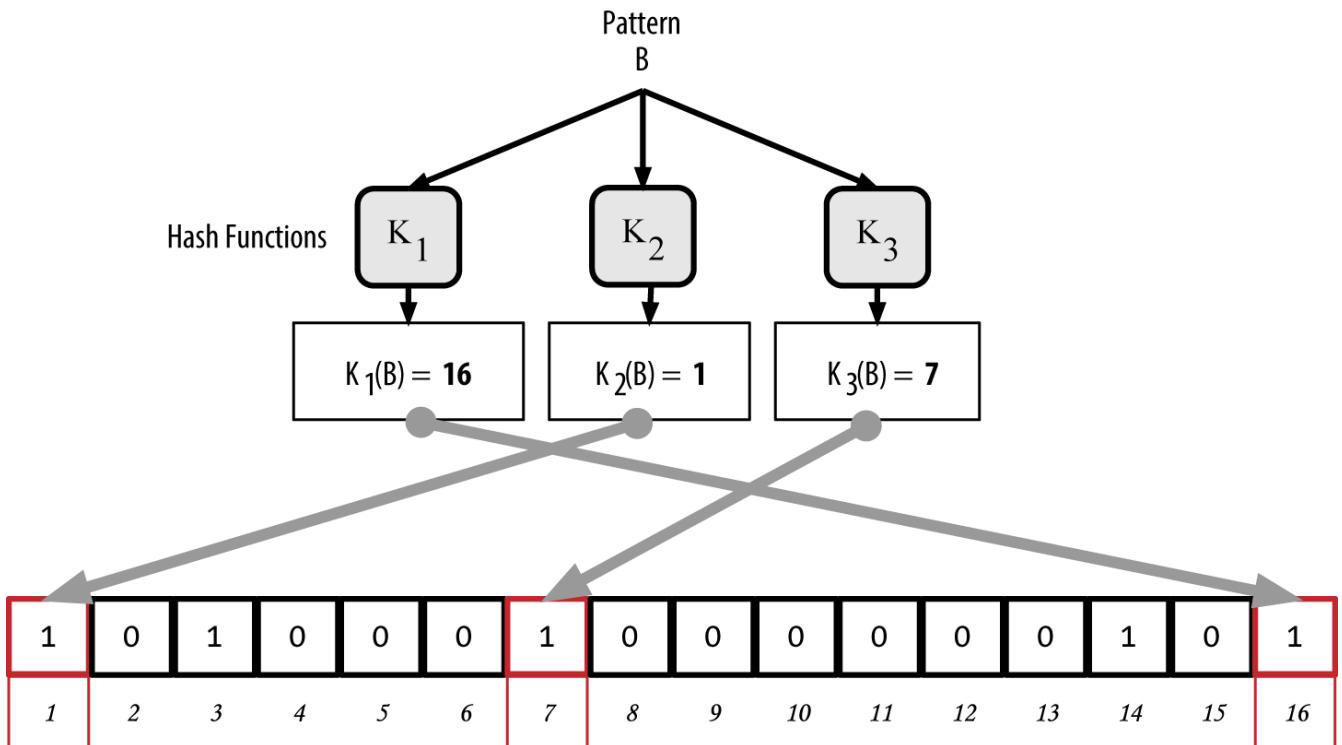


Figure 58. Adăugarea unui al doilea tipar "B" la filtrul bloom simplu

Pentru a testa dacă un tipar face parte dintr-un filtru bloom, tiparul este rezumat de fiecare funcție de rezumare, iar modelul de biți rezultat este testat față de sirul de biți. Dacă toți biții indexați de funcțiile de rezumare sunt setați la 1, atunci tiparul este *probabil* înregistrat în filtrul bloom. Deoarece biții pot fi setați din cauza suprapunerii din mai multe tipare, răspunsul nu este sigur, ci este mai degrabă probabilistic. În termeni simpli, o potrivire cu un filtru bloom este echivalent cu un "Poate, Da".

Testarea existenței tiparului "X" în filtrul bloom. Rezultatul este o potrivire pozitivă probabilistică, însemnând "Poate" este un exemplu de testare a existenței tiparului "X" în filtrul bloom simplu. Biți corespunzători sunt setați la 1, deci tiparul probabil se potrivește.

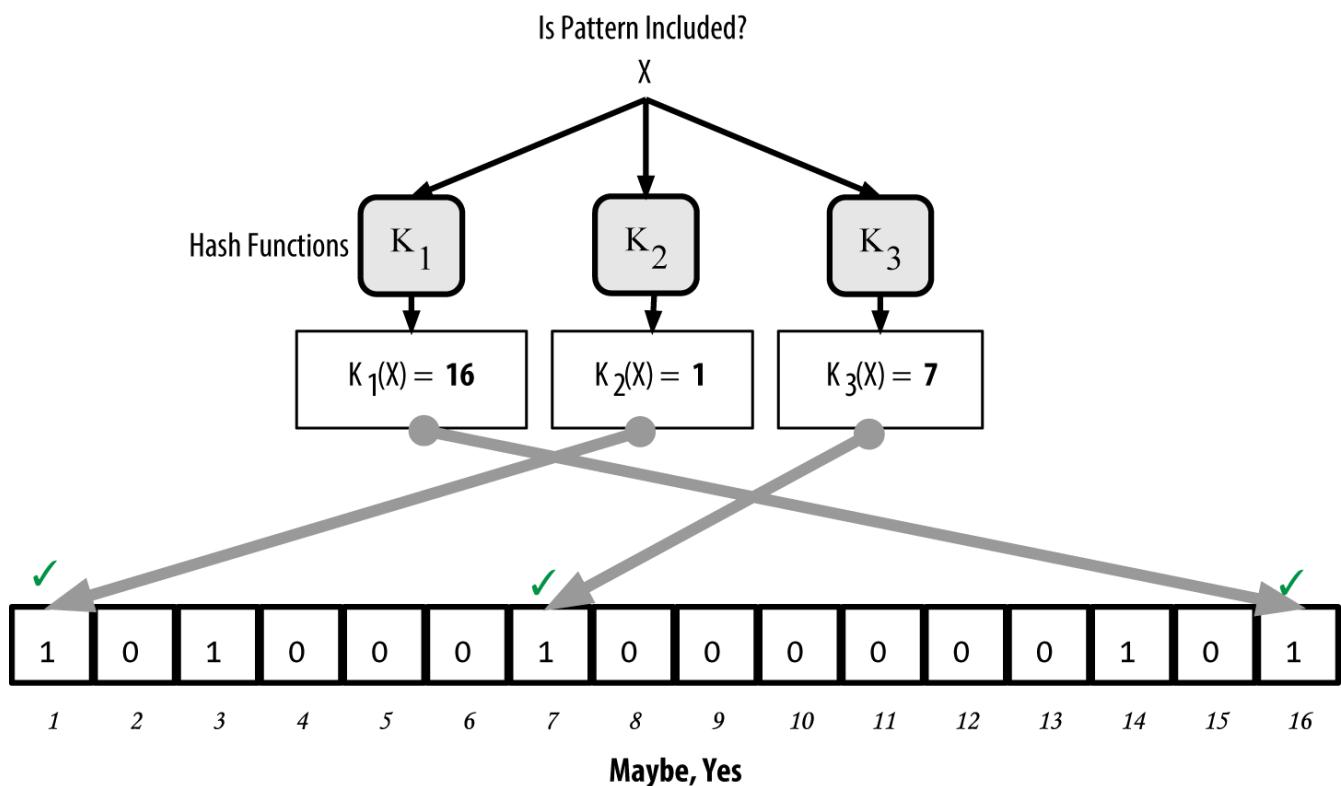


Figure 59. Testarea existenței tiparului "X" în filtrul bloom. Rezultatul este o potrivire pozitivă probabilistică, însemnând "Poate"

Dimpotrivă, dacă un tipar este testat pe filtrul bloom și oricare dintre biți este setat la 0, acest lucru dovedește că tiparul nu a fost înregistrat în filtrul bloom. Un rezultat negativ nu este o probabilitate, ci o certitudine. În termeni simpli, o potrivire negativă pe un filtru bloom este un "Absolut Nu!".

Testarea existenței tiparului "Y" în filtrul bloom. Rezultatul este o potrivire negativă absolută, însemnând "Absolut Nu!" este un exemplu de testare a existenței tiparului "Y" în filtrul bloom simplu. Unul dintre biții corespunzători este setat pe 0, deci tiparul cu siguranță nu este o potrivire.

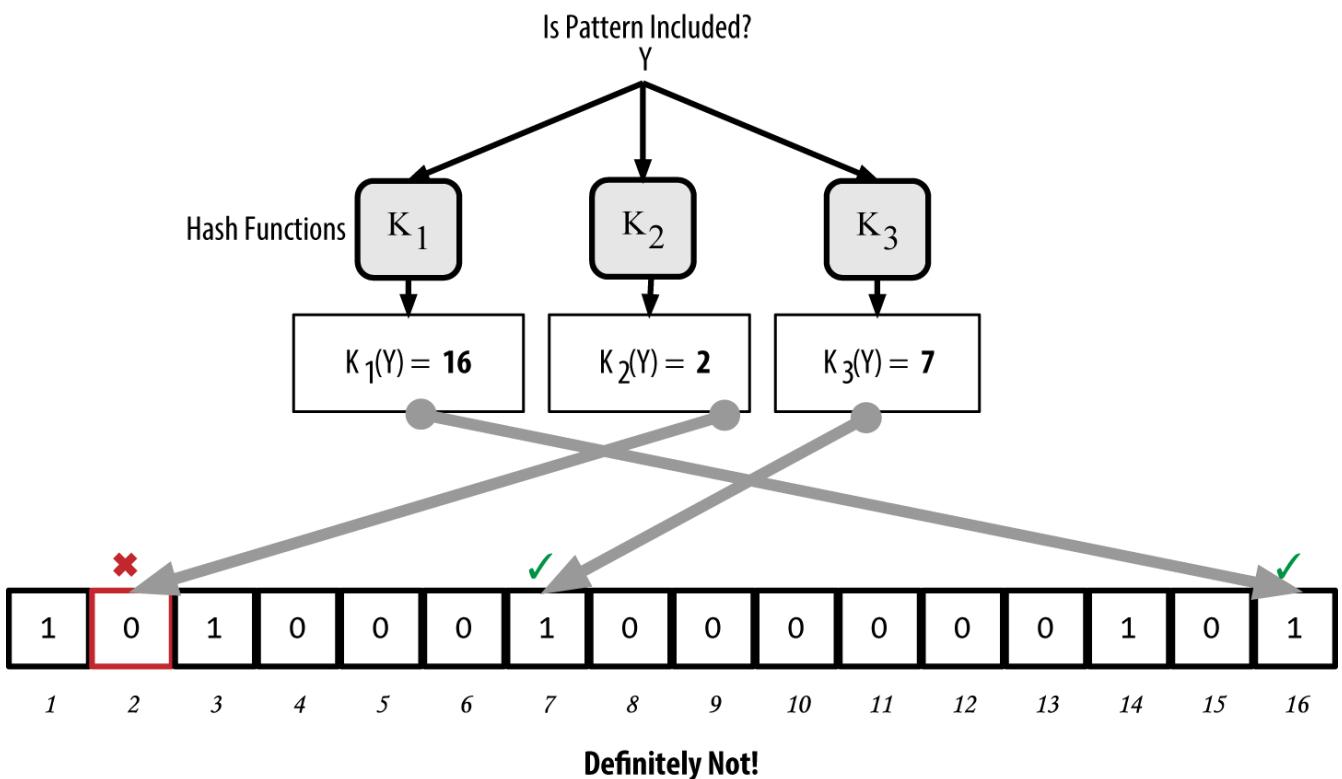


Figure 60. Testarea existenței tiparului "Y" în filtrul bloom. Rezultatul este o potrivire negativă absolută, însemnând "Absolut Nu!"

## Cum Folosesc Nodurile SPV Filtre Bloom

Filtrele bloom sunt utilizate pentru a filtra tranzacțiile (și blocurile care le conțin) pe care un nod SPV le primește de la semenii săi, selectând doar tranzacții de interes pentru nodul SPV, fără să dezvăluie care sunt adresele sau cheile de care este interesat.

Un nod SPV va inițializa un filtru bloom ca fiind "gol"; în această stare, filtrul bloom nu se va potrivi cu niciun tipar. Nodul SPV va face apoi o listă cu toate adresele, cheile și rezumatele de care este interesat. Va face acest lucru prin extragerea rezumatului cheii publice și rezumatul scriptului și ID-urile tranzacțiilor din orice UTXO controlat de portofelul său. Nodul SPV adaugă apoi fiecare dintre acestea la filtrul bloom, astfel încât filtrul bloom se va "potrivi" dacă aceste tipare sunt prezente într-o tranzacție, fără a dezvăluie tiparele în sine.

Nodul SPV va trimite apoi un mesaj **filterload** către seamăn (peer), conținând filtrul bloom pentru a fi utilizat în conexiune. Pe partea seamănului (peer), filtrele bloom sunt aplicate pentru fiecare tranzacție primită. Nodul complet verifică mai multe părți ale tranzacției față de filtrul bloom, căutând o potrivire care include:

- ID-ul tranzacției
- Componentele de date din scripturile de blocare ale fiecărei dintre ieșirile tranzacției (fiecare cheie și rezumat din script)
- Fiecare dintre intrările tranzacției
- Fiecare componentă de date a semnăturii pentru intrare (sau scripturi martor)

Verificând toate aceste componente, filtrele bloom pot fi utilizate pentru a se potrivi cu rezumate de chei publice, scripturi, valori **OP\_RETURN**, chei publice în semnături sau orice componentă viitoare a

unui contract inteligent sau a unui script complex.

După ce un filtru este stabilit, seamănul (peer) va testa apoi ieșirile fiecărei tranzacții cu filtrul bloom. Numai tranzacțiile care se potrivesc filtrului sunt trimise către nod.

Ca răspuns la un mesaj `getdata` de la nod, semenii vor trimite un mesaj `merkleblock` care conține doar anteturile blocurilor pentru blocurile care se potrivesc cu filtrul și o cale merkle (vezi [Arbore Merkle](#)) pentru fiecare tranzacție care se potrivește. Seamănul (peer) va trimite apoi mesaje `tx` care conțin tranzacțiile care se potrivesc cu filtrul.

Deoarece nodul complet trimite tranzacții către nodul SPV, nodul SPV elimină orice validări pozitive false și folosește tranzacțiile potrivite pentru a actualiza setul UTXO și soldul portofelului. Pe măsură ce își actualizează propria perspectivă asupra setului UTXO, modifică, de asemenea, filtrul bloom pentru a se potrivi cu orice tranzacții viitoare care fac referire la UTXO-ul pe care tocmai l-a găsit. Nodul complet folosește apoi noul filtru bloom pentru a potrivi noi tranzacții iar întregul proces se repetă.

Nodul care setează filtrul bloom poate să adauge tipare iterativ la filtru trimițând un mesaj `filteradd`. Pentru a goli filtrul bloom, nodul poate trimite un mesaj `filterclear`. Deoarece nu este posibil să eliminați un tipar dintr-un filtru bloom, un nod trebuie să steargă și să retrimită un nou filtru bloom dacă un model nu mai este dorit.

Protocolul de rețea și mecanismul de filtrare bloom pentru nodurile SPV este definit în [BIP-37 \(Peer Services\)](#).

## Nodurile SPV și Confidențialitatea

Nodurile care implementează SPV au o confidențialitate mai slabă decât un nod complet. Un nod complet primește toate tranzacțiile și, prin urmare, nu dezvăluie nicio informație referitor la utilizarea unei anumite adrese din portofel. Un nod SPV primește o listă filtrată de tranzacții asociate cu adresele care se află în portofelul său. Drept urmare, reduce confidențialitatea proprietarului.

Filtrele bloom sunt o modalitate de a reduce pierderea confidențialității. Fără ele, un nod SPV ar trebui să enumere în mod explicit adresele de care este interesat, creând o încălcare gravă confidențialității. Cu toate acestea, chiar și cu filtre bloom, un adversar care monitorizează traficul unui client SPV sau conectat la acesta direct ca nod în rețeaua P2P poate colecta suficiente informații pe parcursul timpului pentru a afla adresele din portofelul clientului SPV.

## Conexiuni Criptate și Autentificate

Majoritatea utilizatorilor noi de bitcoin presupun că comunicarea în rețea a unui nod bitcoin este criptată. De fapt, implementarea inițială a bitcoin comunică în totalitate în mod clar. Deși aceasta nu este o problemă majoră de confidențialitate pentru nodurile complete, este o problemă mare pentru nodurile SPV.

Ca o modalitate de a crește confidențialitatea și securitatea rețelei P2P bitcoin, există două soluții care asigură criptarea comunicațiilor: *Transport Tor și Autentificare și Criptare P2P* cu BIP-150/151.

## Transport Tor

Tor, care vine de la *Rețeaua de Rutare Onion* (The Onion Routing network), este un proiect și o rețea software care oferă criptarea și încapsularea datelor prin distribuirea aleatorie de căi de rețea care oferă anonimat, nedepistare și confidențialitate.

Bitcoin Core oferă mai multe opțiuni de configurare care vă permit să rulați un nod bitcoin cu traficul său transportat prin rețeaua Tor. În plus, Bitcoin Core poate oferi, de asemenea, un serviciu ascuns Tor care permite altor noduri Tor să se conecteze la nodul dumneavoastră direct peste Tor.

Începând cu versiunea 0.12 a Bitcoin Core, un nod va oferi automat un serviciu Tor ascuns dacă este capabil să se conecteze la un serviciu Tor local. Dacă aveți Tor instalat și procesul Bitcoin Core rulează ca utilizator cu permișii adecvate pentru a accesa cookie-ul de autentificare Tor, acesta ar trebui să funcționeze automat. Folosiți indicatorul **debug** pentru a activa depanarea Bitcoin Core pentru serviciul Tor astfel:

```
$ bitcoind --daemon --debug=tor
```

Ar trebui să vedeți "tor: ADD\_ONION successful" în loguri, ceea ce indică faptul că Bitcoin Core a adăugat un serviciu ascuns în rețeaua Tor.

Puteți găsi mai multe instrucțiuni cu privire la rularea Bitcoin Core ca serviciu ascuns Tor în documentația Bitcoin Core (*docs/tor.md*) și în diverse tutoriale online.

## Autentificare și Criptare De-la-Egal-la-Egal

Două propuneri de îmbunătățire bitcoin, BIP-150 și BIP-151, adaugă sprijin pentru autentificarea P2P și criptarea în rețeaua P2P bitcoin. Aceste două BIP-uri definesc servicii opționale care pot fi oferite de nodurile bitcoin compatibile. BIP-151 permite criptarea negociată pentru toate comunicațiile între două noduri care acceptă BIP-151. BIP-150 oferă o autentificare opțională, care permite nodurilor să-și autentifice reciproc identitatea folosind ECDSA și chei private. BIP-150 necesită ca, înainte de autentificare, cele două noduri să fi stabilit comunicări criptate conform BIP-151.

La momentul ianuarie 2017, BIP-150 și BIP-151 nu sunt implementate în Bitcoin Core. Cu toate acestea, cele două propuneri au fost puse în aplicare de cel puțin un client alternativ bitcoin numit *bcoin*.

BIP-150 și BIP-151 permit utilizatorilor să ruleze clienți SPV care se conectează la un nod complet de încredere, folosind criptare și autentificare pentru a proteja confidențialitatea clientului SPV.

În plus, autentificarea poate fi utilizată pentru a crea rețele de noduri bitcoin de încredere și pentru a preveni atacurile Man-in-the-Middle. În cele din urmă, criptarea P2P, dacă este utilizată pe scară largă, ar consolida rezistența bitcoin la analiza traficului și supravegherea vieții private, în special în țările totalitare, unde utilizarea internetului este puternic controlată și monitorizată.

Standardul este definit în [BIP-150 \(Peer Authentication\)](#) și [BIP-151 \(Peer-to-Peer Communication Encryption\)](#).

## Bazine de Tranzacții

Aproape fiecare nod din rețeaua bitcoin păstrează o listă temporară de tranzacții neconfirmate numită *memory pool*, *mempool* sau *bazin de tranzacții*. Nodurile folosesc acest bazin pentru a urmări tranzacțiile cunoscute în rețea, dar care nu sunt încă incluse în lanțul-de-blocuri. De exemplu, un nod portofel va folosi bazinul de tranzacții pentru a urmări plățile primite în portofelul utilizatorului care au fost primite de rețea, dar încă nu sunt confirmate.

Pe măsură ce tranzacțiile sunt primite și verificate, acestea sunt adăugate în bazinul de tranzacții și transmise la nodurile vecine pentru a se propaga în rețea.

Unele implementări de noduri mențin, de asemenea, un grup separat de tranzacții orfane. Dacă intrările unei tranzacții se referă la o tranzacție care nu este încă cunoscută, cum ar fi un părinte care lipsește, tranzacția orfană va fi stocată temporar în grupul orfan până la sosirea tranzacției părinte.

Atunci când o tranzacție este adăugată la bazinul de tranzacții, bazinul orfan este verificat pentru orfani care fac referire la ieșirile acestei tranzacții (copiii acesteia). Orice orfani care se potrivesc sunt apoi validați. Dacă sunt valide, aceste tranzacții sunt eliminate din bazinul de orfani și sunt adăugate în bazinul de tranzacții, completând lanțul care a început cu tranzacția părinte. Având în vedere tranzacția recent adăugată, care nu mai este orfană, procesul se repetă căutând în mod recursiv alți descendenți, până când nu se vor mai găsi niciunii. Prin acest proces, sosirea unei tranzacții părinte declanșează o reconstrucție în cascadă a unui întreg lanț de tranzacții interdependente, prin re-unirea orfanilor cu părinții lor.

Atât bazinul de tranzacții, cât și bazinul de orfani (atunci când este implementat) sunt stocate în memoria locală și nu sunt salvate pe disc; mai degrabă, sunt populate dinamic din mesajele de rețea primite. Când un nod pornește, ambele bazine sunt goale și sunt populate treptat cu noi tranzacții primite de la rețea.

Unele implementări ale clientului bitcoin mențin, de asemenea, o bază de date sau un bazin UTXO, care este setul tuturor ieșirilor necheltuite de pe lanțul-de-blocuri. Deși numele "bazin UTXO" sună similar cu bazin de tranzacții, acesta reprezintă un set diferit de date. Spre deosebire de bazinele de tranzacții și de orfani, bazinul UTXO nu este inițializat gol, ci conține, în schimb, milioane de elemente de ieșiri de tranzacții necheltuite, tot ceea ce nu este cheltuit de la blocul geneză până în prezent. Bazinul UTXO poate fi stocat în memoria locală sau sub forma unei tabele de baze de date indexate pentru stocarea persistentă.

În timp ce bazinele de tranzacții și de orfani reprezintă perspectiva locală a unui singur nod și pot varia semnificativ de la nod la nod, în funcție de momentul în care nodul a fost pornit sau repornit, bazinul UTXO reprezintă consensul emergent al rețelei și, prin urmare, va varia puțin între noduri. În plus, bazinele de tranzacții și de orfani conțin doar tranzacții neconfirmate, în timp ce bazinul UTXO conține numai ieșiri confirmate.

## Lanțul-de-Blocuri

# Introducere

Structura de date lanț-de-blocuri este o listă ordonată, înlántuită înapoi de blocuri de tranzacții. Lanțul-de-blocuri poate fi stocat ca un fișier plat sau într-o bază de date simplă. Clientul Bitcoin Core stochează metadata lanțului-de-blocuri folosind baza de date LevelDB de la Google. Blocurile sunt legate "înapoi", fiecare având o referință la blocul anterior din lanț. Lanțul-de-blocuri este adesea vizualizat ca o stivă verticală, cu blocuri stratificate unul peste altul iar primul bloc servește ca temelie a stivei. Vizualizarea blocurilor stivuite unul peste altul are ca rezultat folosirea unor termeni precum "înălțime" pentru a face referire la distanța față de primul bloc și "sus" sau "vârf" pentru a face referire la cel mai recent bloc adăugat.

Fiecare bloc din lanțul-de-blocuri este identificat de un rezumat (hash), generat din antetul blocului folosind algoritmul de rezumare criptografic SHA256. Fiecare bloc face referire, de asemenea, la un bloc anterior, cunoscut sub numele de bloc *părinte*, folosind câmpul "rezumat bloc anterior" (previous block hash) din antetul blocului. Cu alte cuvinte, fiecare bloc conține rezumatul părintelui său în interiorul propriului antet. Secvența de rezumate care leagă fiecare bloc cu părintele său creează un lanț care se continuă până la primul bloc creat vreodată, cunoscut sub numele de *bloc gezneză*.

Deși un bloc are un singur părinte, acesta poate avea temporar mai mulți copii. Fiecare dintre copii se referă la același bloc ca fiind părintele său și conține același rezumat (părinte) în câmpul "rezumat bloc anterior". Mai mulți copii apar în timpul unei "bifurcări", o situație temporară care apare atunci când diferite blocuri sunt descoperite aproape simultan de către diferiți mineri (vezi [Bifurcări ale Lanțului-de-Blocuri](#)). În cele din urmă, un singur bloc copil devine parte din lanțul-de-blocuri iar "bifurcarea" este rezolvată. Chiar dacă un bloc poate avea mai mult de un copil, fiecare bloc poate avea un singur părinte. Acest lucru se datorează faptului că un bloc are un singur câmp "rezumat bloc anterior" care face referire la un singur părinte.

Câmpul "rezumat bloc anterior" se află în interiorul antetului blocului și afectează astfel rezumatul blocului *current*. Identitatea copilului se schimbă dacă identitatea părintelui se schimbă. Când părintele este modificat în orice mod, rezumatul părintelui se schimbă. Rezumatul modificat al părintelui necesită o schimbare în câmpul "rezumat bloc anterior" al copilului. La rândul său, cauzând modificarea rezumatului copilului, ceea ce necesită o schimbare a indicelui nepotului, care la rândul său îl schimbă pe nepot și aşa mai departe. Acest efect în cascadă asigură că, odată ce un bloc are multe generații în urma lui, acesta nu poate fi schimbat fără a forța recalcularea tuturor blocurilor care îl preced. Deoarece o astfel de recalculare ar necesita un calcul enorm (și, prin urmare, un consum de energie), existența unui lanț lung de blocuri face ca istoria profundă a lanțului-de-blocuri să fie imutabilă, ceea ce este o caracteristică cheie a securității bitcoin.

O modalitate de a ne gândi la lanțul-de-blocuri este ca straturile dintr-o formațiune geologică sau un eșantion de miez de ghețar. Straturile de suprafață s-ar putea schimba odată cu anotimpurile, sau pot fi chiar spulberate înainte de a avea timp să se stabilească. Dar odată ce mergeți câțiva centimetri adâncime, straturile geologice devin din ce în ce mai stabile. În momentul în care priviți câteva sute de metri în jos, vă uitați la o imagine a trecutului care a rămas netulburată de milioane de ani. În lanțul-de-blocuri, cele mai recente câteva blocuri ar putea fi revizuite dacă există o recalculare a lanțului din cauza unei bifurcări. Cele șase blocuri din vârf sunt precum câțiva centimetri de pământ. Dar, odată ce intrați mai adânc în lanțul-de-blocuri, dincolo de șase blocuri, este din ce în ce mai puțin probabil ca blocurile să fie modificate. După 100 de blocuri, există atât

de multă stabilitate încât tranzacția coinbase - tranzacția care conține bitcoin recent minat - poate fi cheltuită. Câteva mii de blocuri înapoi (o lună) și lanțul-de-blocuri are un istoric stabilit, pentru toate scopurile practice. În timp ce protocolul permite întotdeauna unui lanț să fie anulat de un lanț mai lung și în timp ce există posibilitatea ca orice bloc să fie inversat, probabilitatea unui astfel de eveniment scade pe măsură ce trece timpul până devine infimă.

## Structura unui Bloc

Un bloc este o structură de date container care agregă tranzacțiile pentru a le include în registrul public, lanțul-de-blocuri (blockchain). Blocul este format dintr-un antet, care conține metadate, urmat de o lungă listă de tranzacții care constituie cea mai mare parte a dimensiunii sale. Antetul blocului este de 80 de octeți, în timp ce tranzacția medie este de cel puțin 400 de octeți, iar un bloc mediu conține mai mult de 1900 de tranzacții. Prin urmare, un bloc complet, cu toate tranzacțiile, este de 10.000 de ori mai mare decât antetul blocului. [Structura unui bloc](#) descrie structura unui bloc.

Table 22. Structura unui bloc

Dimensiune	Câmp	Descriere
4 octeți	Dimensiunea Blocului	Mărimea blocului, în octeți, care urmează după acest câmp
80 octeți	Antetul Blocului	Mai multe câmpuri formează antetul blocului
1–9 octeți (VarInt)	Contor Tranzacții	Câte tranzacții urmează
Variabilă	Tranzacții	Tranzacțiile înregistrate în acest bloc

## Antetul Blocului

Antetul blocului este format din trei seturi de metadate de blocuri. În primul rând, există o referință la un rezumat de bloc anterior, care conectează acest bloc la blocul anterior din lanțul-de-blocuri. Al doilea set de metadate, și anume *dificultatea, marca de timp (timestamp)* și *nonce*, se referă la competiția minieră, aşa cum este detaliat în [Minerit și Consens](#). A treia parte de metadate este rădăcina arborelui merkle, o structură de date utilizată pentru a rezuma eficient toate tranzacțiile din bloc. [Structura antetului blocului](#) descrie structura unui antet de bloc.

Table 23. Structura antetului blocului

Dimensiune	Câmp	Descriere
4 octeți	Versiunea	Un număr de versiune pentru urmărirea actualizărilor de software/protocol
32 octeți	Rezumatul Blocului Anterior	O referință la rezumatul blocului anterior (părinte) din lanț

Dimensiune	Câmp	Descriere
32 octeți	Rădăcina Merkle	Un rezumat din rădăcina arborelui merkle a tranzacțiilor acestui bloc
4 octeți	Marca de timp	Timpul aproximativ de creare al acestui bloc (secunde de la Unix Epoch)
4 octeți	Tinta de Dificultate	Tinta de dificultate a algoritmului Dovadă-de-Lucru pentru acest bloc
4 octeți	Nonce	Un contor folosit pentru algoritmul Dovadă-de-Lucru

Nonce-ul, ținta de dificultate și marca de timp sunt utilizate în procesul minier și vor fi discutate mai detaliat în [Minerit și Consens](#).

## Identifierii Blocului: Rezumatul Antetului Blocului și Înălțimea Blocului

Identifierul principal al unui bloc este rezumatul său criptografic, o amprentă digitală, realizat prin rezumarea de două ori a antetului blocului cu algoritmul SHA256. Rezumatul de 32 de octeți rezultat este denumit *rezumat bloc*, dar mai precis este *rezumat antet bloc*, pentru că este folosit doar antetul blocului pentru calcule. De exemplu, *000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f* este rezumatul primului bloc bitcoin creat vreodată. Rezumatul blocului identifică un bloc în mod unic și fără ambiguitate și poate fi derivat independent de orice nod prin simpla rezumare a antetului blocului.

Rețineți că rezumatul blocului nu este de fapt inclus în structura de date a blocului, nici atunci când blocul este transmis în rețea și nici atunci când este stocat de către un nod pe disc ca parte a lanțului-de-blocuri. În schimb, rezumatul blocului este calculat de fiecare nod pe măsură ce blocul este primit de la rețea. Rezumatul blocului ar putea fi stocat într-o tabelă de baze de date separată ca parte a metadatelor blocului, pentru a facilita indexarea și găsirea mai rapidă a blocurilor de pe disc.

O a doua modalitate de a identifica un bloc este prin poziția sa în lanțul-de-blocuri, numită *înălțimea blocului*. Primul bloc creat vreodată este la înălțimea 0 (zero) și este același bloc care a fost referențiat anterior cu următorul rezumat de bloc *000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f*. Un bloc poate fi astfel identificat în două moduri: prin referire la rezumatul blocului sau prin raportarea înălțimii blocului. Fiecare bloc ulterior adăugat "deasupra" primului bloc este o poziție "mai sus" în lanțul-de-blocuri, ca niște cutii stivuite una peste alta. Înălțimea blocului la 1 ianuarie 2017 a fost de aproximativ 446.000, ceea ce înseamnă că erau 446.000 de blocuri stivuite în partea superioară a primului bloc creat în ianuarie 2009.

Spre deosebire de rezumatul blocului, înălțimea blocului nu este un identifier unic. Deși un singur bloc va avea întotdeauna o înălțime specifică și invariabilă, inversul nu este adevărat - înăl-

țimea blocului nu identifică întotdeauna un singur bloc. Două sau mai multe blocuri ar putea avea aceeași înălțime a blocului, concurând pentru aceeași poziție în lanțul-de-blocuri. Acest scenariu este discutat în detaliu în secțiunea [Bifurcări ale Lanțului-de-Blocuri](#). Înălțimea blocului nu face parte, din structura de date a blocului; nu este stocată în bloc. Fiecare nod identifică dinamic poziția (înălțimea) unui bloc în lanțul-de-blocuri atunci când este primit de la rețeaua bitcoin. Înălțimea blocului ar putea fi, de asemenea, stocată sub formă de metadate într-un tabel indexat al bazei de date pentru o recuperare mai rapidă.

**TIP** *Rezumatul unui bloc* identifică întotdeauna un singur bloc în mod unic. De asemenea, un bloc are întotdeauna o *înălțime de bloc* specifică. Cu toate acestea, nu este întotdeauna cazul ca o înălțime specifică a blocului să poată identifica un singur bloc. Mai degrabă, două sau mai multe blocuri ar putea concura pentru o singură poziție în lanțul-de-blocuri.

## Blocul Geneză

Primul bloc din lanțul-de-blocuri se numește blocul geneză și a fost creat în 2009. Este strămoșul comun al tuturor blocurilor din lanțul-de-blocuri, ceea ce înseamnă că dacă porniți de la orice bloc și urmați lanțul înapoi în timp, veți ajunge în cele din urmă la blocul geneză.

Fiecare nod începe întotdeauna cu un lanț-de-blocuri de cel puțin un bloc, deoarece blocul geneză este codat static în software-ul clientului bitcoin, astfel încât acesta nu poate fi modificat. Fiecare nod "cunoaște" întotdeauna rezumatul și structura blocului geneză, timpul exact la care a fost creat și chiar singura tranzacție din interior. Astfel, fiecare nod are punctul de plecare pentru lanțul-de-blocuri, o "rădăcină" sigură din care să construiască un lanț-de-blocuri de încredere.

Puteți vedea blocul geneză codat static în clientul Bitcoin Core, la [chainparams.cpp](#).

Următorul identificator rezumat aparține blocului geneză:

```
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Puteți căuta rezumatul respectiv pe orice site explorator de blocuri, cum ar fi [blockchain.info](#), și veți găsi o pagină care descrie conținutul acestui bloc, cu o adresă URL care conține rezumatul:

<https://blockchain.info/block/>

```
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Folosind clientul referință Bitcoin Core de la linia de comandă:

```
$ bitcoin-cli 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" :  
  "00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"  
}
```

Blocul geneză conține un mesaj ascuns în el. Intrarea tranzacției coinbase conține textul "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks". Acest mesaj a fost menit să ofere dovada celei mai vechi date la care a fost creat acest bloc, prin referirea la titlul ziarului britanic *The Times*. De asemenea, servește ca o amintire ironică a importanței unui sistem monetar independent, lansarea bitcoin având loc în același timp cu o criză monetară mondială fără precedent. Mesajul a fost încorporat în primul bloc de către Satoshi Nakamoto, creatorul bitcoin.

# Legarea Blocurilor în Lanțul-de-Blocuri

Nodurile bitcoin complete păstrează o copie locală a lanțului-de-blocuri, începând de la blocul geneză. Copia locală a lanțului-de-blocuri este actualizată constant, deoarece blocuri noi sunt găsite și folosite pentru a extinde lanțul. Pe măsură ce un nod primește blocuri de la rețea, acesta va valida aceste blocuri și apoi le va lega de lanțul-de-blocuri existent. Pentru a stabili o legătură, un nod va examina antetul blocului primit și va căuta "rezumatul blocului anterior".

Să presupunem, de exemplu, că un nod are 277.314 blocuri în copia locală a lanțului-de-blocuri. Ultimul bloc despre care știe nodul este blocul 277.314, cu un rezumat de antet de bloc de:

0000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249

Nodul bitcoin primește apoi un nou bloc din rețea, pe care îl parcurge după cum urmează:

```

{
  "size" : 43560,
  "version" : 2,
  "previousblockhash" :
    "0000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
  "merkleroot" :
    "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
  "time" : 1388185038,
  "difficulty" : 1180923195.25802612,
  "nonce" : 4215469401,
  "tx" : [
    "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",
    "#[... many more transactions omitted ...]
      "05cf38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"
  ]
}

```

Privind acest nou bloc, nodul găsește câmpul **previousblockhash**, care conține rezumatul blocului său părinte. Este un rezumat cunoscut nodului, cel al ultimului bloc de pe lanț, la înălțimea de 277.314. Prin urmare, acest nou bloc este un copil al ultimului bloc din lanț și extinde lanțul-de-blocuri existent. Nodul adaugă acest nou bloc la capătul lanțului, ceea ce face ca lanțul-de-blocuri să fie mai lung, cu o nouă înălțime de 277.315. **Blocuri legate încruncuite prin referință la rezumatul antetului blocului anterior** arată lanțul a trei blocuri, legate prin referințe în câmpul **previousblockhash**.

## Arbore Merkle

Fiecare bloc din lanțul-de-blocuri bitcoin conține un rezumat al tuturor tranzacțiilor din bloc folosind un *arbore merkle*.

Un *arbore merkle*, cunoscut și sub denumirea de *arbore binar de rezumate*, este o structură de date utilizată pentru rezumarea și verificarea eficientă a integrității unor seturi mari de date. Arborii merkle sunt arbori binari care conțin rezumate criptografice. Termenul "arbore" este utilizat în informatică pentru a descrie o structură de date ramificată, dar acești arbori sunt de obicei afișați cu susul în jos cu "rădăcina" din partea de sus și "frunzele" din partea de jos a unei diagrame, aşa cum veți vedea în exemplele care urmează.

Block Height 277316

Header Hash:

0000000000000001b6b9a13b095e96db  
41c4a928b97ef2d944a9b31b2cc7bdc4

Previous Block Header Hash:

0000000000000002a7bbd25a417c0374  
cc55261021e8a9ca74442b01284f0569

Timestamp: 2013-12-27 23:11:54

Difficulty: 1180923195.26

Nonce: 924591752

Merkle Root: c91c008c26e50763e9f548bb8b2  
fc323735f73577effbc55502c51eb4cc7cf2e

H  
E  
A  
D  
E  
R

Transactions

Block Height 277315

Header Hash:

0000000000000002a7bbd25a417c0374  
cc55261021e8a9ca74442b01284f0569

Previous Block Header Hash:

00000000000000027e7ba6fe7bad39fa  
f3b5a83daed765f05f7d1b71a1632249

Timestamp: 2013-12-27 22:57:18

Difficulty: 1180923195.26

Nonce: 4215469401

Merkle Root: 5e049f4030e0ab2debb92378f5  
3c0a6e09548aea083f3ab25e1d94ea1155e29d

Transactions

Block Height 277314

Header Hash:

00000000000000027e7ba6fe7bad39fa  
f3b5a83daed765f05f7d1b71a1632249

Previous Block Header Hash:

00000000000000038388d97cc6f2c1d  
fe116c5e879330232f3bff1c645920bdf

Timestamp: 2013-12-27 22:55:40

Difficulty: 1180923195.26

Nonce: 3797028665

Merkle Root: 02327049330a25d4d17e53e79f  
478ccb79c53a509679b1d8a1505c5697afb326

Transactions

Figure 61. Blocuri legate într-un lanț prin referință la rezumatul antetului blocului anterior

Arboarele merkle sunt folosiți în bitcoin pentru a rezuma toate tranzacțiile dintr-un bloc, producând o amprentă digitală generală a întregului set de tranzacții, oferind un proces foarte eficient pentru a verifica dacă o tranzacție este inclusă într-un bloc. Un arbore merkle este construit prin rezumarea recursivă a perechilor de noduri până când există un singur rezumat, denumit *rădăcină*, sau *rădăcină merkle (merkle root)*. Algoritmul de rezumare criptografic utilizat în arborele merkle bitcoin constă din aplicarea SHA256 de două ori, cunoscut și sub numele de dublu-SHA256.

Când  $N$  elemente sunt rezumate și aggregate într-un arbore merkle, puteți verifica dacă un element de date este inclus în arbore cu cel mult  $2 * \log_2(N)$  operații, făcând arborele binar o structură de date foarte eficientă.

Arborele merkle este construit de jos în sus. În exemplul următor, începem cu patru tranzacții, A, B, C și D, care formează *frunze* din arborele merkle, așa cum se arată în [Calcularea nodurilor dintr-un arbore merkle](#). Tranzacțiile nu sunt stocate în arborele merkle; mai degrabă, datele lor sunt rezumate iar rezumatul rezultat este stocat în fiecare nod frunză sub forma  $H_A$ ,  $H_B$ ,  $H_C$ , and  $H_D$ :

$$H_A = \text{SHA256}(\text{SHA256}(\text{Transaction A}))$$

Perechile consecutive de noduri frunze sunt apoi rezumate într-un nod părinte, concatenând cele două rezumate și îmbinându-le împreună. De exemplu, pentru a construi nodul părinte  $H_{AB}$ , cele două rezumate de 32 de octeți ale copiilor sunt concatenate și creează un sir de 64 de octeți. Acest sir este apoi dublu-rezumat pentru a produce rezumatul nodului părinte:

$$H_{AB} = \text{SHA256}(\text{SHA256}(H_A + H_B))$$

Procesul continuă până când există un singur nod în partea de sus, nod cunoscut sub numele de *rădăcina merkle*. Acest rezumat de 32 de octeți este stocat în antetul blocului și rezumă toate datele din cele patru tranzacții. [Calcularea nodurilor dintr-un arbore merkle](#) arată modul în care rădăcina este calculată prin rezumarea perechilor de noduri.

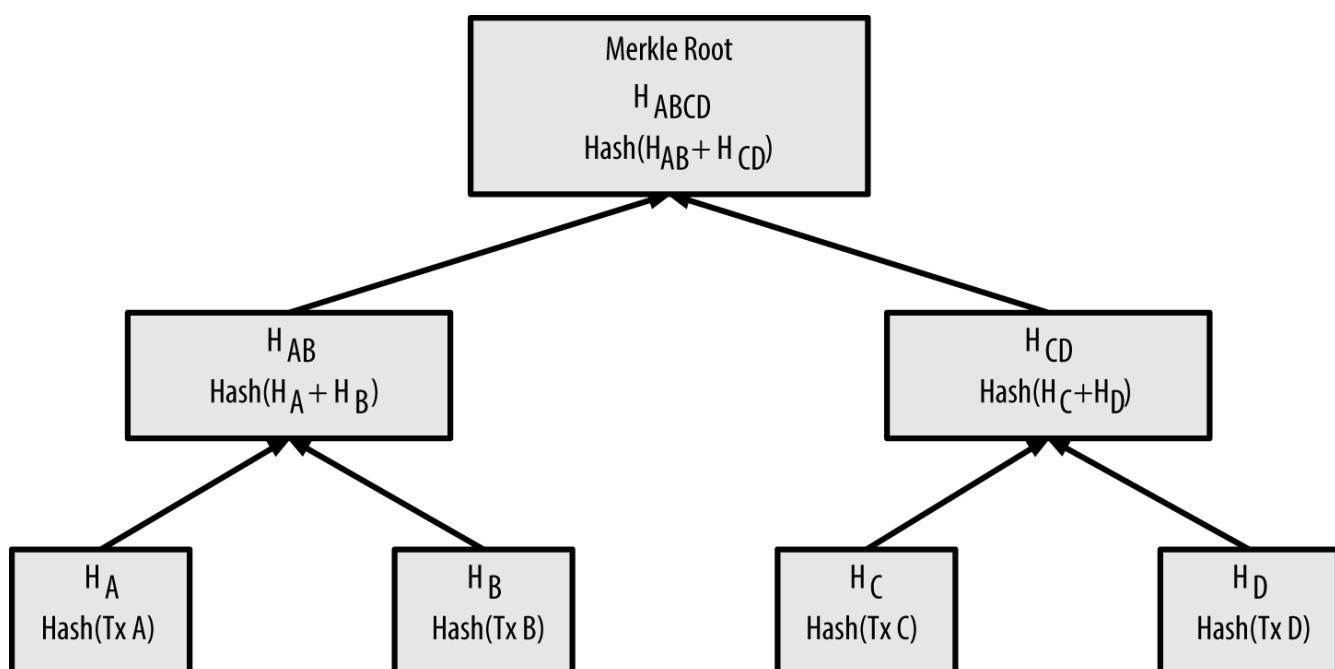


Figure 62. Calcularea nodurilor dintr-un arbore merkle

Deoarece arborele merkle este un arbore binar, are nevoie de un număr par de noduri de frunze. Dacă există un număr impar de tranzacții de sumarizat, ultimul rezumat al tranzacției va fi duplicat pentru a crea un număr egal de noduri de frunze, cunoscut și sub denumirea de *arbore echilibrat*. Acest lucru este arătat în [Duplicarea unui element rezultă într-un număr par de elemente](#), unde tranzacția C este duplicată.

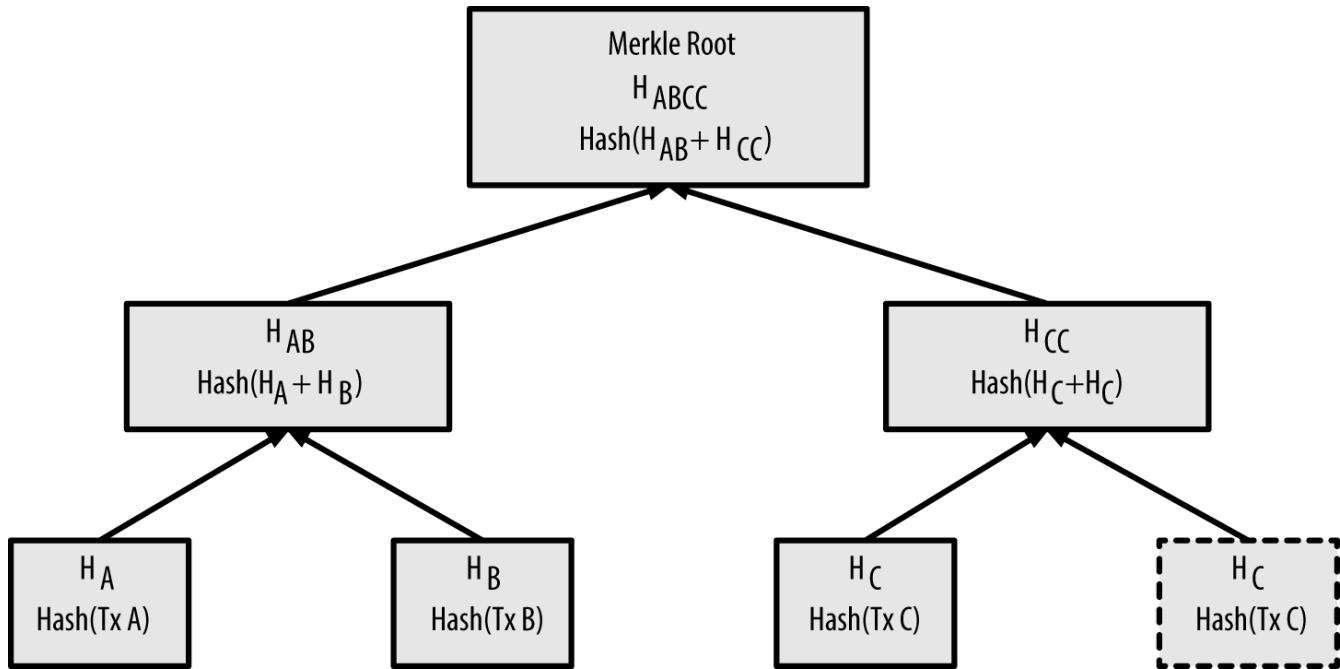


Figure 63. Duplicarea unui element rezultă într-un număr par de elemente

Aceeași metodă pentru construirea unui arbore din patru tranzacții poate fi generalizată pentru a construi arbori de orice dimensiune. În bitcoin de obicei există câteva sute până la mai mult de o mie de tranzacții într-un singur bloc, care sunt rezumate exact în același mod, producând doar 32 de octeți de date ca rădăcină merkle unică. În [Un arbore merkle care sumarizează mai multe elemente](#), veți vedea un arbore construit din 16 tranzacții. Rețineți că, deși rădăcina pare mai mare decât nodurile frunzelor din diagramă, este exact aceeași dimensiune, de doar 32 de octeți. Fie că există o tranzacție sau o sută de mii de tranzacții în bloc, rădăcina merkle le rezumă întotdeauna în 32 de octeți.

Pentru a dovedi că o tranzacție specifică este inclusă într-un bloc, un nod trebuie să producă  $\log_2(N)$  rezumate pe 32 de octeți, constituind o *cale de autentificare* sau *cale merkle* conectând tranzacția specifică cu rădăcina arborelui. Acest lucru este deosebit de important pe măsură ce numărul tranzacțiilor crește, deoarece logaritmul în baza 2 al numărului de tranzacții crește mult mai lent. Acest lucru permite nodurilor bitcoin să producă în mod eficient căi de 10 sau 12 de rezumate (320-384 octeți), ceea ce poate oferi dovada unei singure tranzacții din mai mult de o mie de tranzacții într-un bloc de un megabyte.

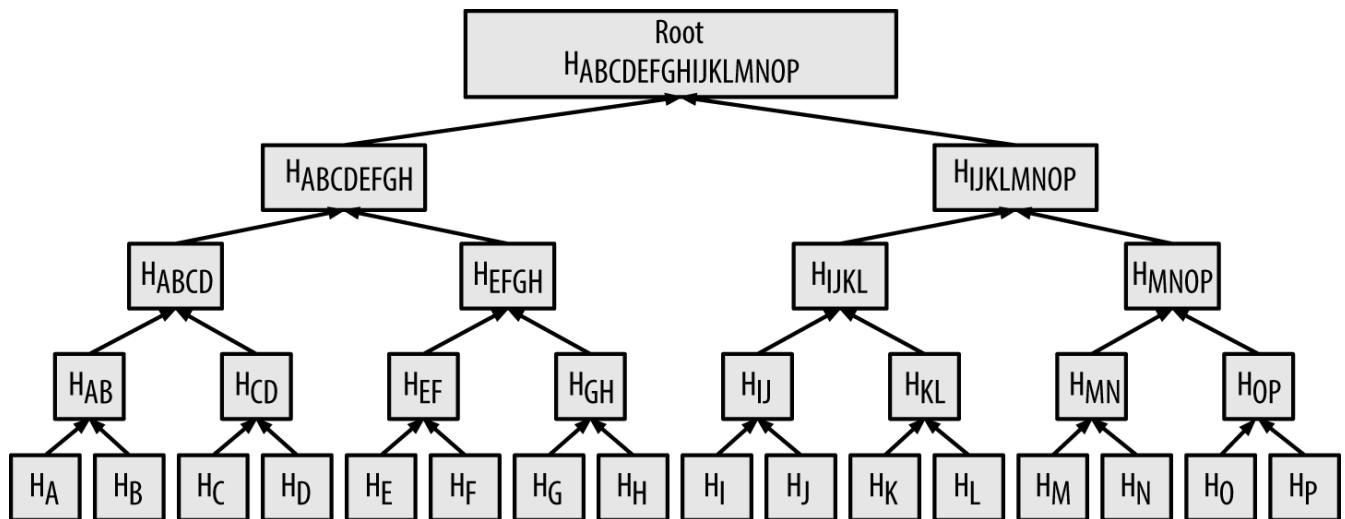


Figure 64. Un arbore merkle care sumarizează mai multe elemente

În [O cale merkle folosită pentru a dovedi inclusiunea unui element](#), un nod poate dovedi că o tranzacție K este inclusă în bloc prin producerea unei căi merkle care are doar patru rezumate pe 32 de octeți (128 octeți în total). Calea este formată din cele patru rezumate (prezentate cu un fundal negru în [O cale merkle folosită pentru a dovedi inclusiunea unui element](#))  $H_L$ ,  $H_{IJ}$ ,  $H_{MNOP}$ , și  $H_{ABCDEFGH}$ . Cu cele patru rezumate furnizate ca o cale de autentificare, orice nod poate dovedi că  $H_K$  (cu fundal negru în partea de jos a diagramei) este inclus în rădăcina merkle, calculând patru rezumate adiționale  $H_{KL}$ ,  $H_{IJKL}$ ,  $H_{IJKLMNOP}$  și rădăcina arborelui merkle (conturată într-o linie punctată în diagramă).

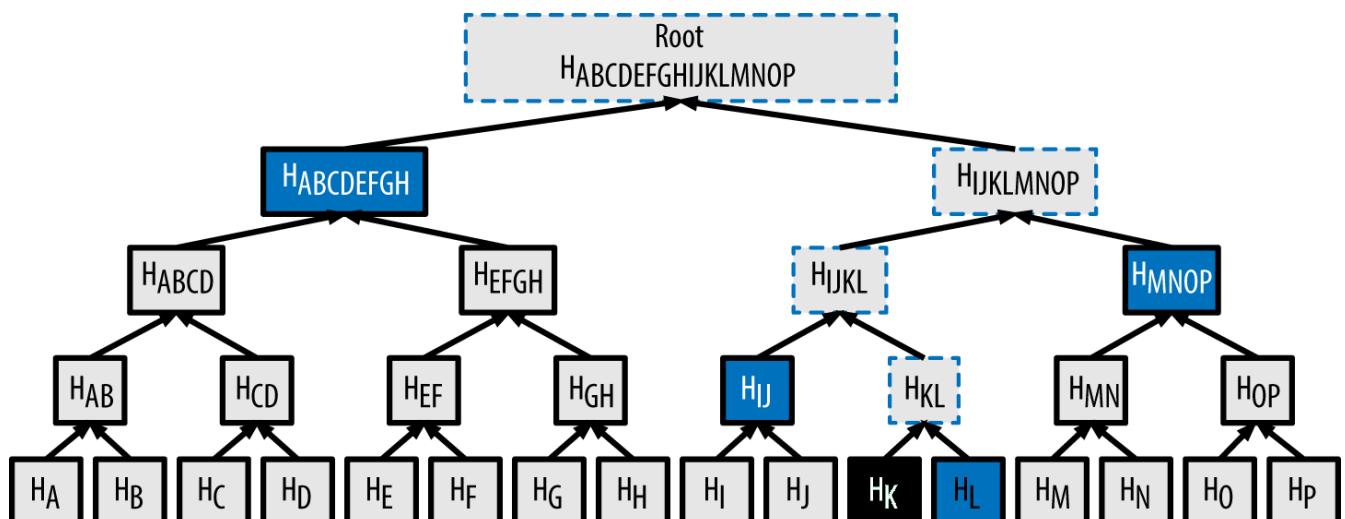


Figure 65. O cale merkle folosită pentru a dovedi inclusiunea unui element

Codul de la [Crearea unui arbore merkle](#) demonstrează procesul de creare a unui arbore merkle din rezumatele de noduri frunze până la rădăcină, folosind biblioteca libbitcoin pentru unele funcții auxiliare.

Example 20. Crearea unui arbore merkle

```
#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);
            auto concat = bc::serializer<
                decltype(concat_data.begin())>(concat_data.begin());
            concat.write_hash(*it);
            concat.write_hash(*(it + 1));
            // Hash both of the hashes.
            bc::hash_digest new_root = bc::bitcoin_hash(concat_data);
            // Add this to the new list.
            new_merkle.push_back(new_root);
        }
        // This is the new list.
        merkle = new_merkle;

        // DEBUG output -----
        std::cout << "Current merkle hash list:" << std::endl;
        for (const auto& hash: merkle)
            std::cout << " " << bc::encode_base16(hash) << std::endl;
        std::cout << std::endl;
        // -----
    }
    // Finally we end up with a single item.
    return merkle[0];
}
```

Compilarea și rularea codului exemplului merkle arată rezultatul compilării și rulării codului merkle.

*Example 21. Compilarea și rularea codului exemplului merkle*

```
$ # Compile the merkle.cpp code
$ g++ -o merkle merkle.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Run the merkle executable
$ ./merkle
Current merkle hash list:
32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d27006
30861db96905c8dc8b99398ca1cd5bhd5b84ac3264a4e1b3e65afa1bcee7540c4
```

Current merkle hash list:  
d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

Result: d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

Eficiența arborilor merkle devine evidentă odată cu creșterea dimensiunii. **Eficiența arborelui merkel** arată cantitatea de date care trebuie schimbată sub formă de cale merkle pentru a dovedi că o tranzacție face parte dintr-un bloc.

Table 24. Eficiența arborelui merkel

Număr de tranzacții	Dimensiunea aproximativă a blocului	Mărimea căii (rezumate)	Mărimea căii (octeți)
16 tranzacții	4 kilobytes	4 rezumate	128 octeți
512 tranzacții	128 kilobytes	9 rezumate	288 octeți
2048 tranzacții	512 kilobytes	11 rezumate	352 octeți
65.535 tranzacții	16 megabytes	16 rezumate	512 octeți

După cum puteți vedea din tabel, în timp ce dimensiunea blocului crește rapid, de la 4 KB cu 16 tranzacții la o dimensiune de 16 MB cu 65.535 tranzacții, calea merkle necesară pentru a dovedi includerea unei tranzacții crește mult mai lent, de la 128 octeți la numai 512 octeți. Folosind arbori merkle, un nod poate descărca doar anteturile blocurilor (80 de octeți per bloc) și poate fi în continuare capabil să identifice includerea unei tranzacții într-un bloc, obținând o cale mică merkle dintr-un nod complet, fără să stocheze sau să transmită marea majoritate a lanțului-de-blocuri, care poate avea mai mulți GB în dimensiune. Nodurile care nu mențin un lanț-de-blocuri complet, numite noduri de verificare simplificată a plăților (SPV), utilizează căi merkle pentru a verifica tranzacțiile fără a descărca blocuri complete.

## Arbori Merkle și verificarea simplificată a plății (SPV)

Arborii Merkle sunt utilizati pe scară largă de nodurile SPV. Nodurile SPV nu au toate tranzacțiile și nu descarcă blocuri complete, ci doar anteturile blocurilor. Pentru a verifica dacă o tranzacție este inclusă într-un bloc, fără a fi nevoie să descarce toate tranzacțiile din bloc, folosesc o cale de autentificare sau o cale merkle.

Luați în considerare, de exemplu, un nod SPV care este interesat de plățile primite către o adresă conținută în portofelul său. Nodul SPV va stabili un filtru bloom (vezi [Filtre Bloom](#)) pe conexiunile sale cu semenii, pentru a limita tranzacțiile primite doar la cele care conțin adresele de interes. Când un seaman vede o tranzacție care se potrivește cu filtrul bloom, acesta va trimite acel bloc folosind un mesaj [merkleblock](#). Mesajul [merkleblock](#) conține antetul blocului, precum și o cale merkle care leagă tranzacția de interes de rădăcina merkle din bloc. Nodul SPV poate folosi această cale merkle pentru a conecta tranzacția la bloc și a verifica dacă tranzacția este inclusă în bloc. Nodul SPV folosește, de asemenea, antetul blocului pentru a conecta blocul la restul lanțului-de-blocuri. Combinarea dintre aceste două legături, între tranzacție și bloc, și între bloc și lanțul-de-blocuri, dovedește că tranzacția este înregistrată în lanțul-de-blocuri. În total, nodul SPV va fi primit mai puțin de un kilobyte de date pentru antetul blocului și calea merkle, o cantitate de date care este de peste o mie de ori mai mică decât un bloc complet (aproximativ 1 megabyte în prezent).

## Lanțurile-de-blocuri Bitcoin pentru Teste

S-ar putea să fiți surprins să aflați că există mai multe lanțuri-de-blocuri bitcoin. Lanțul-de-blocuri "principal" bitcoin, cel creat de Satoshi Nakamoto pe 3 ianuarie 2009, cel cu blocul geneză pe care l-am studiat în acest capitol, se numește *mainnet*. Există și alte lanțuri-de-blocuri bitcoin utilizate în scopuri de testare: în acest moment *testnet*, *segnet* și *regtest*. Să ne uităm la fiecare pe rând.

## Testnet - locul de joacă pentru testarea Bitcoin

Testnet este numele lanțului-de-blocuri, rețelei și monedei utilizate în scopuri de testare. Testnet este o rețea P2P completă, cu portofele, bitcoin de test (monede testnet), minerit și toate celelalte caracteristici ale mainnet-ului. Există într-adevăr doar două diferențe: monedele testnet sunt inutile, iar dificultatea de minerit ar trebui să fie suficient de scăzută astfel încât oricine să poată mina monede de testnet relativ ușor (ramânând fără valoare).

Orice aplicație software destinată utilizării în producție în rețeaua bitcoin ar trebui testată mai întâi pe testnet cu monede de test. Acest lucru protejează atât dezvoltatorii împotriva pierderii banilor din cauza erorilor, cât și a rețelei împotriva funcționalităților nedorite cauzate de defecte.

Nu este ușor să păstrezi monedele fără valoare și mineritul la o dificultate scăzută. În ciuda cererilor din partea dezvoltatorilor, unii oameni utilizează echipamente de minerit avansate (GPU și ASIC) pentru a mina pe testnet. Acest lucru crește dificultatea, face imposibilă minarea cu un CPU și, în cele din urmă, devine suficient de dificil pentru a obține monede de test astfel încât acestea încep să nu mai fie fără valoare. Drept urmare, din când în când, testnet-ul trebuie curățat și repornit dintr-un nou bloc geneză, resetând dificultatea.

Testnet-ul curent se numește *testnet3*, a treia iterată a testnet-ului, a fost repornit în februarie 2011 pentru a reseta dificultatea testnet-ului anterior.

Rețineți că testnet3 este un lanț-de-blocuri mare, care depășea 20 GB la începutul lui 2017. Va dura cam o zi pentru a vă sincroniza complet și a utiliza resursele de pe calculatorul dumneavoastră. Nu la fel de mult ca și rețeaua principală, dar nici chiar "rapid". O modalitate bună de a rula un nod testnet este folosind o imagine a mașinii virtuale (de exemplu, VirtualBox, Docker, Cloud Server, etc.) dedicată acestui scop.

### Utilizarea testnet-ului

Bitcoin Core, ca aproape toate celelalte software-uri bitcoin, are suport complet pentru operarea pe testnet în loc de mainnet. Toate funcțiile Bitcoin Core funcționează pe testnet, inclusiv portofelul, mineritul de monede testnet și sincronizarea unui nod complet de testnet.

Pentru a porni Bitcoin Core pe testnet în loc de mainnet, utilizați opțiunea **testnet**:

```
$ bitcoind -testnet
```

În loguri ar trebui să vedeți că bitcoind creează o nou lanț-de-blocuri în subdirectorul *testnet3* al directorului implicit bitcoind:

```
bitcoind: Using data directory /home/username/.bitcoin/testnet3
```

Pentru a vă conecta la bitcoind, utilizați **bitcoin-cli** de la linia de comandă, dar trebuie să folosiți de asemenea opțiunea *testnet*:

```
$ bitcoin-cli -testnet getblockchaininfo
{
  "chain": "test",
  "blocks": 1088,
  "headers": 139999,
  "bestblockhash": "0000000063d29909d475a1c4ba26da64b368e56cce5d925097bf3a2084370128",
  "difficulty": 1,
  "mediantime": 1337966158,
  "verificationprogress": 0.001644065914099759,
  "chainwork": "000000000000000000000000000000000000000000000000000000000000044104410441",
  "pruned": false,
  "softforks": [
    ...
  ]
}
```

De asemenea, puteți rula pe testnet3 cu alte implementări de noduri complete, cum ar fi *btcd* (scris în Go) și *bcoin* (scris în JavaScript), pentru a experimenta și a învăța în alte limbi de programare și framework-uri.

De la începutul anului 2017, testnet3 acceptă toate caracteristicile rețelei principale, inclusiv Martor Segregat (vezi [Martor Segregat](#)). Prin urmare, testnet3 poate fi utilizat și pentru testarea caracteristicilor Martor Segregat.

## Segnet - Testnet-ul pentru Martor Segregat

În 2016, a fost lansat un testnet cu scop special pentru a ajuta la dezvoltarea și testarea Martor Segregat (aka segwit; vezi [Martor Segregat](#)). Acest lanț-de-blocuri de testare se numește *segnet* și poate fi folosit rulând o versiune specială (branch) a Bitcoin Core.

Deoarece segwit a fost adăugat la testnet3, nu mai este necesar să utilizați segnet pentru testarea funcțiilor segwit.

În viitor, este probabil să vedem și alte lanțuri-de-blocuri testnet care sunt concepute special pentru a testa o singură caracteristică sau schimbări arhitecturale majore, cum ar fi segnet.

## Regtest - Lanțul-de-blocuri Local

Regtest, care înseamnă "Regression Testing" este o caracteristică Bitcoin Core care vă permite să creați un lanț-de-blocuri local pentru testare. Spre deosebire de testnet3, care este un lanț-de-blocuri public de testare, lanțurile-de-blocuri regtest sunt destinate să fie rulate ca sisteme închise pentru testarea locală. Puteți lansa de la zero un lanț-de-blocuri regtest, creând un bloc geneză local. Puteți adăuga alte noduri în rețea sau puteți rula cu un singur nod numai pentru a testa software-ul Bitcoin Core.

Pentru a porni Bitcoin Core în modul regtest, utilizați indicatorul **regtest**:

```
$ bitcoind -regtest
```

La fel ca în cazul testnet-ului, Bitcoin Core va inițializa un nou lanț-de-blocuri în subdirectorul *regtest* al directorului implicit bitcoind:

bitcoind: Using data directory /home/username/.bitcoin/regtest

Când folosiți linia de comandă, trebuie să specificați și indicatorul `regtest`. Să încercăm comanda `getblockchaininfo` pentru a inspecta lanțul-de-blocuri regtest:

După cum puteți vedea, nu există încă blocuri. Haideți să minăm câteva (500 blocuri) și să obținem recompensa:

```
$ bitcoin-cli -regtest generate 500
[
  "7afed70259f22c2bf11e406cb12ed5c0657b6e16a6477a9f8b28e2046b5ba1ca",
  "1aca2f154a80a9863a9aac4c72047a6d3f385c4eec5441a4aafa6acaa1dada14",
  "4334ecf6fb022f30fb764c3ee778fabbd53b4a4d1950eae8a91f1f5158ed2d1",
  "5f951d34065eafeaf64e54e91d00b260294fcdfc7f05dbb5599aec84b957a7766",
  "43744b5e77c1dfce9d05ab5f0e6796ebe627303163547e69e27f55d0f2b9353",
  [...]
  "6c31585a48d4fc2b3fd25521f4515b18aefb59d0def82bd9c2185c4ecb754327"
]
```

Va dura doar câteva secunde pentru a mina toate aceste blocuri, ceea ce face cu siguranță testarea ușoară. Dacă verificați soldul portofelului, veți vedea că ați câștigat recompense pentru primele 400 de blocuri (recompensele coinbase trebuie să fie mai vechi de 100 de blocuri înainte de a le putea cheltui):

```
$ bitcoin-cli -regtest getbalance  
12462.50000000
```

# Utilizarea Lanțurilor-de-Blocuri de Test pentru Programare

Diversele lanțuri-de-blocuri bitcoin (`regtest`, `segnet`, `testnet3`, `mainnet`) oferă o serie de medii de testare pentru programarea bitcoin. Utilizați lanțurile-de-blocuri de testare dacă dezvoltăți pentru Bitcoin Core sau un alt client nod-complet de consens; o aplicație cum ar fi portofelul, bursa de schimb, site-ul de comerț electronic; sau chiar dezvoltarea de noi contracte inteligente și scripturi complexe.

Puteți utiliza lanțurile-de-blocuri de testare pentru a stabili un proces de dezvoltare a software-ului. Testați-vă codul local pe un `regtest` pe măsură ce îl dezvoltăți. După ce sunteți gata să îl încercați într-o rețea publică, treceți la `testnet` pentru a expune codul într-un mediu mai dinamic, cu mai multă diversitate de aplicații. În cele din urmă, după ce sunteți sigur că codul dumneavoastră funcționează așa cum vă așteptați, treceți la `mainnet` pentru a-l implementa în producție. Pe măsură ce efectuați modificări, îmbunătățiri, corecții de erori etc., începeți din nou procesul, implementând fiecare modificare mai întâi pe `regtest`, apoi pe `testnet` și în final în producție.

## Minerit și Consens

### Introducere

Cuvântul "minerit" este oarecum înșelător. Prin evocarea extracției de metale prețioase, ne concentreză atenția asupra recompensei pentru minerit, bitcoin nou creat în fiecare bloc. Deși mineritul este stimulat de această recompensă, scopul principal al mineritului nu este recompensa sau generarea de monede noi. Dacă vedeați mineritul doar ca procesul prin care sunt create monede, confundați mijloacele (stimulente) cu scopul procesului. Mineritul este mecanismul care stă la baza compensării descentralizate, prin care tranzacțiile sunt validate și acceptate. Mineritul este invenția care face bitcoin special, un mecanism de securitate descentralizat care stă la baza banilor digitali de-la-egal-la-egal (P2P - peer-to-peer).

Mineritul *securizează sistemul bitcoin* și permite apariția unui *consens la nivel de rețea fără o autoritate centrală*. Recompensa cu monede noi create și recompensa din comisioanele de tranzacție este o schemă de stimulare care aliniază acțiunile minerilor cu securitatea rețelei, implementând simultan aprovizionarea monetară.

**TIP**

Scopul mineritului nu este crearea de noi bitcoin. Acesta este sistemul de stimulare. Mineritul este mecanismul prin care *securizarea bitcoin este decentralizată*.

Minerii validează tranzacțiile noi și le înregistrează în registrul global. Un nou bloc, care conține tranzacții care au avut loc de la ultimul bloc, este "minat" în medie la fiecare 10 minute, adăugând astfel acele tranzacții la lanțul-de-blocuri. Tranzacțiile care fac parte dintr-un bloc și se adaugă la lanțul-de-blocuri sunt considerate "confirmate", ceea ce permite noilor proprietari de bitcoin să cheltuiască bitcoinul pe care l-au primit în acele tranzacții.

Minerii primesc două tipuri de recompense în schimbul securității pe care o oferă prin minerit:

monede noi create cu fiecare bloc nou și comisioane de tranzacție din toate tranzacțiile incluse în bloc. Pentru a câștiga această recompensă, minerii concurează pentru a rezolva o problemă matematică dificilă bazată pe un algoritm criptografic de rezumare. Soluția problemei, numită Dovadă-de-Lucru, este inclusă în noul bloc și acționează ca dovadă că minerul a investit eforturi de calcul semnificative. Competiția de rezolvare a algoritmului Dovadă-de-Lucru pentru a câștiga recompensa și dreptul de a înregistra tranzacții pe lanțul-de-blocuri este baza modelului de securitate bitcoin.

Procesul se numește minerit, deoarece recompensa (generarea de monede noi) este concepută pentru a simula diminuarea randamentului, la fel ca minarea pentru metale prețioase. Oferta de monede bitcoin este creată prin minerit, similar cu modul în care o bancă centrală emite bani noi prin tipărirea bancnotelor. Cantitatea maximă de bitcoin nou-creat pe care un miner o poate adăuga la un bloc scade aproximativ la fiecare patru ani (sau mai exact la fiecare 210.000 de blocuri). A început la 50 de bitcoin pe bloc în ianuarie 2009 și s-a redus la jumătate (la 25 de bitcoin pe bloc) în noiembrie 2012. S-a redus la jumătate la 12,5 bitcoin în iulie 2016 și din nou la 6,25 în mai 2020. Pe baza acestei formule, recompensele miniere bitcoin scad exponențial până aproximativ în anul 2140, când toată cantitatea de bitcoin (20.9999998 milioane) va fi emisă. După 2140, nu va fi emis niciun bitcoin nou.

Minerii bitcoin câștigă, de asemenea, comisioane din tranzacții. Fiecare tranzacție poate include un comision de tranzacție sub forma unui excedent de bitcoin între intrările și ieșirile tranzacției. Minerul câștigător ajunge să "păstreze restul" de la tranzacțiile incluse în blocul câștigător. Astăzi, comisioanele reprezintă 0,5% sau mai puțin din venitul unui miner, marea majoritate provenind din bitcoin-ul recent creat. Cu toate acestea, pe măsură ce recompensa scade în timp și numărul tranzacțiilor pe bloc crește, o proporție mai mare din veniturile miniere vor proveni din comisioane. Treptat, recompensa minieră va fi dominată de comisioanele de tranzacție, care vor constitui stimulentul principal pentru mineri. După 2140, cantitatea de bitcoin nouă în fiecare bloc scade la zero, iar minarea va fi stimulată doar prin comisioane de tranzacție.

În acest capitol, vom examina mai întâi mineritul ca mecanism de aprovizionare monetară și apoi vom analiza funcția cea mai importantă a mineritului: mecanismul de consens descentralizat care stă la baza securității bitcoin.

Pentru a înțelege mineritul și consensul, vom urmări tranzacția lui Alice, deoarece este prima și adăugată la un bloc de către echipamentele de minerit ale lui Jing. Apoi, vom urmări blocul pe măsură ce este minat, adăugat la lanțul-de-blocuri și acceptat de rețeaua bitcoin prin procesul de consens emergent.

## Economia Bitcoin și Crearea de Monede

Bitcoin sunt "emisi" în timpul creării fiecărui bloc la o rată fixă și în diminuare. Fiecare bloc, generat în medie la fiecare 10 minute, conține bitcoin complet nou, creat din nimic. La fiecare 210.000 de blocuri, sau aproximativ la fiecare patru ani, rata de emitere a monedei este redusă cu 50%. Pentru primii patru ani de funcționare a rețelei, fiecare bloc conținea 50 de bitcoin noi.

În noiembrie 2012, noua rată de emitere bitcoin a scăzut la 25 de bitcoin per bloc. În iulie 2016, aceasta a fost redusă din nou la 12,5 bitcoin per bloc. S-a redus la jumătate (la 6,25 bitcoin) la blocul 630 000, care a fost minat în mai 2020. Rata monedelor noi scade astfel exponențial cu peste 32 de "injumătătiri" până la blocul 6 720 000 (minat aproximativ în anul 2137), când atinge unitatea

minimă de monedă de 1 satoshi. În cele din urmă, după 6,93 milioane de blocuri, în aproximativ 2140, vor fi emise aproape 2.099.999.997.690.000 de satoshi, sau aproape 21 de milioane de bitcoin. Ulterior, blocurile nu vor conține bitcoin nou, iar minerii vor fi răsplătiți doar prin comisioanele de tranzacție. [Oferta de monedă bitcoin în timp, pe baza unei rate de emitere în scădere geometrică](#) arată totalul bitcoin în circulație în timp, ținând cont că emiterea de monedă scade.

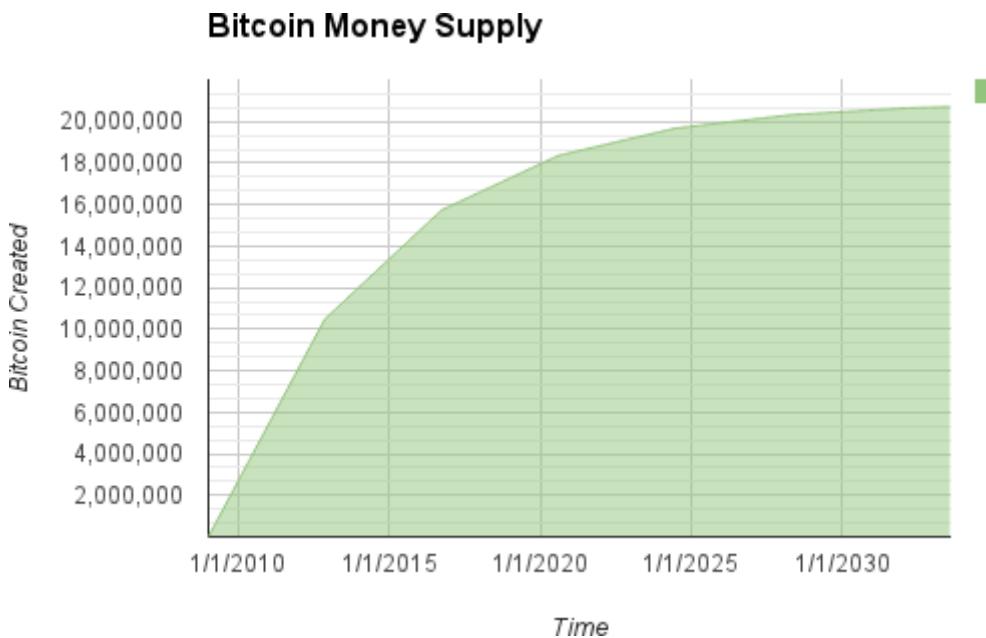


Figure 66. Oferta de monedă bitcoin în timp, pe baza unei rate de emitere în scădere geometrică

**NOTE** Numărul maxim de monede minate este limita *superioară* a posibilelor recompense miniere pentru bitcoin. În practică, un miner poate mina în mod intenționat un bloc primind mai puțin decât recompensa completă. Astfel de blocuri au fost deja minate și mai multe pot fi minate în viitor, ceea ce duce la o emitere totală mai mică a monedei.

În exemplul de cod din [Un script pentru a calcula cât bitcoin va fi pus în circulație în total](#), calculăm cantitatea totală de bitcoin care va fi pusă în circulație.

*Example 22. Un script pentru a calcula cât bitcoin va fi pus în circulație în total*

```
# Original block reward for miners was 50 BTC = 50 0000 0000 Satoshi
start_block_reward = 50 * 10**8
# 210000 is around every 4 years with a 10 minute block interval
reward_interval = 210000

def max_money():
    current_reward = start_block_reward
    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")
```

Rularea scriptului [max\\_money.py](#) arată ieșirea pentru rularea acestui script.

*Example 23. Rularea scriptului max\_money.py*

```
$ python max_money.py
Total BTC to ever be created: 209999997690000 Satoshi
```

Punerea în circulație finită și în scădere creează o ofertă monetară fixă care rezistă inflației. Spre deosebire de o monedă fiduciară (fiat currency), care poate fi tipărită în număr infinit de o bancă centrală, bitcoin nu poate fi niciodată umflat prin tipărire.

## Bani Deflaționari

Cea mai importantă și dezbatută consecință a emiterii monetare fixe și în scădere este aceea că moneda tinde să fie inherent *deflationară*. Deflația este fenomenul de apreciere a valorii datorat unei nepotriviri a ofertei și cererii care crește valoarea (și cursul de schimb) al unei monede. Opusul inflației, deflația, înseamnă că banii au mai multă putere de cumpărare în timp.

Mulți economisti susțin că o economie deflaționară este un dezastru care ar trebui evitat cu orice preț. Acest lucru se datorează faptului că într-o perioadă de deflație rapidă, oamenii tind să adune bani în loc să îi cheltuiască, sperând că prețurile vor scădea. Un astfel de fenomen s-a desfășurat în timpul "Decadei Pierdute" a Japoniei, când o prăbușire completă a cererii a împins moneda într-o spirală deflaționară.

Experții bitcoin susțin că deflația nu este rea în sine. Mai degrabă, deflația este asociată cu o prăbușire a cererii, deoarece acesta este singurul exemplu de deflație pe care trebuie să-l studiem. Într-o monedă fiduciară, cu posibilitatea de tipărire nelimitată, este foarte dificil să introduceți o spirală deflaționară, cu excepția cazului în care există o prăbușire completă a cererii și o lipsă de dorință de a imprima bani. Deflația în bitcoin nu este cauzată de o prăbușire a cererii, ci de o ofertă restrânsă predictibil.

Aspectul pozitiv al deflației este, desigur, că este opusul inflației. Inflația provoacă o devalorizare lentă, dar inevitabilă a monedei, rezultând într-o formă de impozitare ascunsă, care îi pedepsește pe cei care economisesc pentru a salva debitorii (inclusiv cei mai mari debitori, guvernele în sine). Monedele aflate sub control guvernamental suferă de pericolul moral de emitere usoară a datoriilor, care poate fi ulterior ștearsă prin devalorizare în detrimentul economisitorilor.

Rămâne de văzut dacă aspectul deflaționist al monedei este o problemă atunci când nu este determinată de contracție economică rapidă sau un avantaj, deoarece protecția împotriva inflației și a devalorizării depășește cu mult riscurile deflației.

## Consens Descentralizat

În capitolul anterior am analizat lanțul-de-blocuri, registrul public global (lista) al tuturor tranzacțiilor, pe care toată lumea din rețeaua bitcoin îl acceptă ca înregistrarea oficială a dreptului de proprietate.

Dar cum pot fi de acord toți cei din rețea cu un singur "adevăr" universal despre cine deține ce, fără a fi nevoie să aibă încredere în nimeni? Toate sistemele tradiționale de plată depind de un model de încredere care are o autoritate centrală care furnizează un serviciu de compensare, verificând practic și compensând toate tranzacțiile. Bitcoin nu are o autoritate centrală, însă, într-un fel, fiecare nod complet are o copie completă a registrului public în care poate avea încredere ca fiind înregistrarea oficială. Lanțul-de-blocuri nu este creat de o autoritate centrală, ci este asamblat independent de fiecare nod din rețea. Cumva, fiecare nod din rețea, care acționează asupra informațiilor transmise prin conexiuni nesigure de rețea, poate ajunge la aceeași concluzie și poate asambla o copie a acelaiași registru public ca toți ceilalți. Acest capitol examinează

procesul prin care rețeaua bitcoin ajunge la un consens global fără o autoritate centrală.

("minerit și consens", "consens emergent") Principala invenție a lui Satoshi Nakamoto este mecanismul descentralizat pentru *consens emergent*. Este emergent, deoarece consensul nu este obținut în mod explicit - nu există alegeri sau un moment fix când consensul are loc. În schimb, consensul este un artefact emergent al interacțiunii asincrone a mii de noduri independente, toate urmând reguli simple. Toate proprietățile bitcoin, inclusiv moneda, tranzacțiile, plățile și modelul de securitate care nu depinde de autoritatea centrală sau de încredere, derivă din această invenție.

Consensul descentralizat al bitcoin rezultă din interacțiunea a patru procese care apar independent pe nodurile din întreaga rețea:

- Verificarea independentă a fiecărei tranzacții, de către fiecare nod complet, pe baza unei liste cuprinzătoare de criterii
- Agregarea independentă a acestor tranzacții în blocuri noi de către noduri de minerit, însotită de calculul demonstrat printr-un algoritm Dovadă-de-Lucru
- Verificarea independentă a noilor blocuri de către fiecare nod și asamblarea într-un lanț
- Selectarea independentă, de către fiecare nod, a lanțului cu cea mai mare cumulare de calcul demonstrată prin Dovadă-de-Lucru

În următoarele secțiuni vom examina aceste procese și modul în care interacționează pentru a crea proprietatea emergentă a unui consens la nivel de rețea care permite oricărui nod bitcoin să își asambleze propria copie a registrului autoritar, de încredere, public, global.

## Verificarea Independentă a Tranzacțiilor

În [Tranzacții](#), am văzut cum software-ul portofel creează tranzacții prin colectarea UTXO-urilor, furnizarea scripturilor de deblocare corespunzătoare și apoi construirea de noi ieșiri alocate unui nou proprietar. Tranzacția rezultată este apoi trimisă nodurilor vecine din rețeaua bitcoin, pentru a putea fi propagată în întreaga rețea bitcoin.

Cu toate acestea, înainte de a transmite tranzacțiile către vecinii săi, fiecare nod bitcoin care primește o tranzacție va verifica mai întâi tranzacția. Acest lucru asigură că numai tranzacțiile valide sunt propagate în rețea, în timp ce tranzacțiile invalide sunt respinse de primul nod care le întâlnește.

Fiecare nod verifică fiecare tranzacție pe baza unei liste de verificare lungi:

- Sintaxa și structura de date a tranzacției trebuie să fie corecte.
- Listele de intrări și de ieșiri nu sunt goale.
- Dimensiunea tranzacției în octeți este mai mică decât `MAX_BLOCK_SIZE`.
- Fiecare valoare de ieșire, precum și totalul, trebuie să se încadreze în intervalul admis de valori (mai puțin de 21 milioane de monede, mai mult decât pragul de *praf*).
- Niciuna dintre intrări nu are rezumat=0, N=-1 (tranzacțiile conibase nu trebuie transmise).
- `nLocktime` este egal cu `INT_MAX`, sau valorile `nLocktime` și `nSequence` sunt satisfăcute în funcție de `MedianTimePast`.

- Dimensiunea tranzacției în octeți este mai mare sau egală cu 100.
- Numărul de operațiuni de semnatură (SIGOPS) conținut în tranzacție este mai mic decât limita de operare a semnăturilor.
- Scriptul de deblocare (`scriptSig`) nu poate decât să împingă numerele pe stivă, iar scriptul de blocare (`scriptPubkey`) trebuie să corespundă formelor `isStandard` (acest lucru respinge tranzacțiile "nonstandard").
- Trebuie să existe o tranzacție similară în bazinul de tranzacții sau într-un bloc din ramura principală.
- Pentru fiecare intrare, dacă ieșirea referențiată există în orice altă tranzacție din bazinul de tranzacții, tranzacția trebuie respinsă.
- Pentru fiecare intrare, caută în ramura principală și în bazinul de tranzacții pentru a găsi tranzacția de ieșire la care se face referire. Dacă tranzacția de ieșire lipsește pentru orice intrare, aceasta va fi o tranzacție orfană. Se adaugă la bazinul de tranzacții orfane, dacă o tranzacție identică nu este deja în bazin.
- Pentru fiecare intrare, dacă tranzacția de ieșire referită este o ieșire coinbase, aceasta trebuie să aibă cel puțin `COINBASE_MATURITY` (100) confirmări.
- Pentru fiecare intrare, ieșirea referențiată trebuie să existe și nu poate fi deja cheltuită.
- Folosind tranzacțiile de ieșire la care se face referire pentru a obține valori de intrare, verifică dacă fiecare valoare de intrare, precum și suma, se află în intervalul admis de valori (mai puțin de 21 milioane bitcoin, mai mult de 0).
- Respinge dacă suma valorilor de intrare este mai mică decât suma valorilor de ieșire.
- Respinge dacă comisionul de tranzacție ar fi prea mic (`minRelayTxFee`) pentru a fi inclus într-un bloc gol.
- fiecare script de deblocare a fiecării intrări trebuie validat față de scripturile corespunzătoare de blocare

Acstea condiții pot fi văzute în detaliu în funcțiile `AcceptToMemoryPool`, `CheckTransaction` și `CheckInputs` în Bitcoin Core. Rețineți că aceste condiții se schimbă de-a lungul timpului, pentru a aborda noi tipuri de atacuri denial-of-service sau uneori pentru a relaxa regulile, astfel încât să includă mai multe tipuri de tranzacții.

Verificând în mod independent fiecare tranzacție aşa cum este primită și înainte de a o propaga, fiecare nod creează un bazin de tranzacții valide (dar neconfirmate) cunoscute sub denumirea de *bazin de tranzacții, memory pool sau mempool*.

## Noduri de Minerit

Unele dintre nodurile din rețeaua bitcoin sunt noduri specializate numite *minerii*. În [Introducere](#) l-am introdus pe Jing, student în inginerie informatică din Shanghai, China, care este miner bitcoin. Jing câștigă bitcoin folosind o "platformă minieră", care este un sistem specializat de hardware proiectat pentru a mina bitcoin. Hardware-ul minier specializat este conectat la un server care rulează un nod bitcoin complet. Spre deosebire de Jing, unii mineri minează fără nod complet, aşa cum vom vedea în [Bazine de Minerit](#). Ca orice alt nod complet, nodul lui Jing primește și propagă tranzacțiile neconfirmate în rețeaua bitcoin. Nodul lui Jing, totuși, agregă aceste tranzacții în

blocuri noi.

Nodul lui Jing ascultă pentru blocuri noi, propagate în rețeaua bitcoin, la fel ca toate nodurile. Cu toate acestea, sosirea unui nou bloc are o semnificație specială pentru un nod miner. Concurența dintre mineri se încheie efectiv cu propagarea unui nou bloc care acționează ca un anunț al căștișilor. Pentru mineri, primirea unui bloc nou valid înseamnă că altcineva a câștigat competiția. Cu toate acestea, sfârșitul unei runde a competiției este și începutul rundei următoare. Noul bloc nu doar marchează sfârșitul cursei; este și semnul de pornire în cursa pentru următorul bloc.

## Agregarea Tranzacțiilor în Blocuri

După validarea tranzacțiilor, un nod bitcoin le va adăuga la *memory pool*, sau *bazinul de tranzacții*, unde tranzacțiile așteaptă până când pot fi incluse (minate) într-un bloc. Nodul lui Jing colectează, validează și transmite noi tranzacții la fel ca orice alt nod. Spre deosebire de alte noduri, totuși, nodul lui Jing va agrega aceste tranzacții într-un bloc *candidat*.

Să urmărim blocurile care au fost create în perioada în care Alice a cumpărat o ceașcă de cafea de la Cafeneaua lui Bob (vezi [Cumpărarea unei Cești de Cafea](#)). Tranzacția lui Alice a fost inclusă în blocul 277.316. În scopul de a demonstra concepțele din acest capitol, să presupunem că blocul a fost extras de sistemul miner al lui Jing și urmărește tranzacția lui Alice, deoarece devine parte a acestui nou bloc.

Nodul miner al lui Jing păstrează o copie locală a lanțului-de-blocuri. Până la momentul când Alice cumpără ceașca de cafea, nodul lui Jing a asamblat un lanț până la blocul 277.314. Nodul lui Jing ascultă pentru tranzacții, încearcă să extragă un bloc nou și ascultă și pentru blocurile descoperite de alte noduri. În timp ce nodul lui Jing minează, acesta primește blocul 277.315 prin rețeaua bitcoin. Sosirea acestui bloc semnifică sfârșitul competiției pentru blocul 277.315 și începutul competiției pentru crearea blocului 277.316.

În decursul ultimelor 10 minute, în timp ce nodul lui Jing căuta o soluție pentru blocul 277.315, acesta colectă de asemenea tranzacții pentru pregătirea următorului bloc. Până acum, a colectat câteva sute de tranzacții în bazinul de memorie. După primirea blocului 277.315 și validarea acestuia, nodul lui Jing îl va compara, de asemenea, cu toate tranzacțiile din bazinul de memorie și va elimina orice tranzacție care a fost inclusă în blocul 277.315. Tranzacțiile rămase în bazinul de memorie nu sunt confirmate și așteaptă să fie înregistrate într-un bloc nou.

Nodul lui Jing construiește imediat un nou bloc gol, un candidat pentru blocul 277.316. Acest bloc se numește *bloc candidat* deoarece nu este încă un bloc valid, deoarece nu conține o Dovadă-de-Lucru validă. Blocul devine valid numai dacă minerul reușește să găsească o soluție la algoritmul Dovadă-de-Lucru.

Când nodul lui Jing agregă toate tranzacțiile din bazinul de memorie, noul bloc candidat are 418 tranzacții cu comisioane totale de tranzacție de 0,09094928 bitcoin. Puteți vedea acest bloc în lanțul-de-blocuri folosind interfața liniei de comandă a clientului Bitcoin Core, așa cum este arătat în [Utilizarea liniei de comandă pentru a prelua blocul 277.316](#).

*Example 24. Utilizarea liniei de comandă pentru a preluă blocul 277.316*

```
$ bitcoin-cli getblockhash 277316
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
$ bitcoin-cli getblock 0000000000000001b6b9a13b095e96db41c4a928b97ef2d9\44a9b31b2cc7bdc4
```

## Tranzacția Coinbase

Prima tranzacție din orice bloc este o tranzacție specială, numită *tranzacție coinbase*. Această tranzacție este construită de nodul lui Jing și conține *recompensa* pentru efortul de minerit.

**NOTE** atunci, s-au scurs două perioade "de înjumătărire". Recompensa blocului s-a schimbat la 12,5 bitcoin în iulie 2016. Aceasta a fost din nou redusă la jumătate peste 210.000 de blocuri, în anul 2020.

Nodul lui Jing creează tranzacția coinbase ca plată către propriul portofel: "Plătește adresa lui Jing 25.09094928 bitcoin". Suma totală de recompensă pe care Jing o încasează pentru minarea unui

bloc este suma recompensei coinbase (25 de bitcoin noi) și comisioanele de tranzacție (0.09094928) din toate tranzacțiile incluse în bloc, să cum se arată în [Tranzacție coinbase](#).

*Example 25. Tranzacție coinbase*

```
$ bitcoin-cli getrawtransaction  
d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f 1
```

Spre deosebire de tranzacțiile obișnuite, tranzacția coinbase nu consumă (cheltuiește) UTXO ca intrări. În schimb, are o singură intrare, numită *coinbase*, care creează bitcoin din nimic. Tranzacția coinbase are o singură ieșire, plătibilă la propria adresă bitcoin a minerului. Rezultatul tranzacției coinbase trimite valoarea de 25.09094928 bitcoin la adresa bitcoin a minerului; în acest caz [1MxTkeEP2PmHSMze5tUZ1hAV3YTJu2Gh1N](https://blockchain.info/tx/1MxTkeEP2PmHSMze5tUZ1hAV3YTJu2Gh1N).

## Recompense și Comisioane Coinbase

Pentru a construi o tranzacție coinbase, nodul lui Jing calculează mai întâi suma totală a comisioanelor de tranzacție prin adăugarea tuturor intrărilor și ieșirilor celor 418 tranzacții care au fost adăugate la bloc. Comisioanele sunt calculate ca:

$$\text{Total Comisioane} = \text{Suma (Intrări)} - \text{Suma (Ieșiri)}$$

În blocul 277.316, comisioanele totale de tranzacție sunt de 0,09094928 bitcoin.

În continuare, nodul lui Jing calculează recompensa corectă pentru noul bloc. Recompensa este calculată pe baza înălțimii blocului, începând de la 50 de bitcoin pe bloc și redusă la jumătate la fiecare 210.000 de blocuri. Deoarece acest bloc are o înălțime de 277.316, recompensa corectă este de 25 de bitcoin.

Calculul poate fi văzut în funcția [GetBlockSubsidy](#) în clientul Bitcoin Core, așa cum se arată în [Calcularea recompensei blocului - Funcția GetBlockSubsidy, Bitcoin Core Client, main.cpp](#).

*Example 26. Calcularea recompensei blocului - Funcția GetBlockSubsidy, Bitcoin Core Client, main.cpp*

```
CAmount GetBlockSubsidy(int nHeight, const Consensus::Params& consensusParams)
{
    int halvings = nHeight / consensusParams.nSubsidyHalvingInterval;
    // Force block reward to zero when right shift is undefined.
    if (halvings >= 64)
        return 0;

    CAmount nSubsidy = 50 * COIN;
    // Subsidy is cut in half every 210,000 blocks which will occur approximately
    // every 4 years.
    nSubsidy >>= halvings;
    return nSubsidy;
}
```

Subvenția inițială este calculată în satoshi înmulțind 50 cu constanta [COIN](#) (100.000.000 satoshi). Aceasta stabilește recompensa inițială ([nSubsidy](#)) la 5 miliarde de satoshi.

În continuare, funcția calculează numărul de "înjumătățiri" care s-au produs prin împărțirea înălțimii curente a blocului la intervalul de înjumătățire ([SubsidyHalvingInterval](#)). În cazul blocului 277.316, cu un interval de înjumătățire la fiecare 210.000 de blocuri, rezultatul este o înjumătățire.

Numărul maxim de jumătăți admis este de 64, deci codul impune o recompensă zero (returnează doar comisioanele) dacă se depășesc cele 64 de înjumătățiri.

În continuare, funcția folosește operatorul de deplasare-binară-dreapta pentru a împărți recompensa ([nSubsidy](#)) în două pentru fiecare rundă de înjumătățire. În cazul blocului 277.316, acest lucru ar deplasa-binar-dreapta recompensa de 5 miliarde de satoshi o dată (la jumătate) și ar

rezulta în 2,5 miliarde de satoshi, sau 25 bitcoin. Operatorul de deplasare-binar-dreapta este utilizat deoarece este mai eficient decât mai multe împărțiri repetate. Pentru a evita o potențială eroare, operația de deplasare este omisă după 63 de înjumătățiri, iar subvenția este setată la 0.

În cele din urmă, recompensa coinbase ([nSubsidy](#)) este adăugată la comisioanele de tranzacție ([nFees](#)), iar suma este returnată.

**TIP** Dacă nodul miner al lui Jing scrie tranzacția coinbase, ce îl oprește pe Jing să se "recompenseze" el însuși cu 100 sau 1000 de bitcoin? Răspunsul este că o recompensă incorectă ar face ca blocul să fie considerat invalid de către toți ceilalți, pierzând energia electrică a lui Jing folosită pentru Dovada-de-Lucru. Jing poate să cheltuiască recompensa numai dacă blocul este acceptat de toată lumea.

## Structura Tranzacției Coinbase

Cu aceste calcule, nodul lui Jing construiește apoi tranzacția coinbase pentru a-și plăti lui însuși 25.09094928 bitcoin.

După cum puteți vedea în [Tranzacție coinbase](#), tranzacția coinbase are un format special. În loc de o intrare de tranzacție care specifică o UTXO anterioară pentru a fi cheltuită, aceasta are o intrare "coinbase". Am examinat intrările tranzacțiilor în [Serializare intrare tranzacție](#). Să comparăm o intrare obișnuită cu o intrare de tranzacție coinbase. [Structura unei intrări "normale" a tranzacției](#) arată structura unei tranzacții obișnuite, în timp ce [Structura unei intrări de tranzacție coinbase](#); arată structura de intrare a tranzacției coinbase.

Table 25. Structura unei intrări "normale" a tranzacției

Dimensiune	Câmp	Descriere
32 octeți	Rezumat (Hash) Tranzacție	Pointer la tranzacția care conține UTXO care trebuie cheltuită
4 octeți	Indicele de ieșire	Numărul de index al UTXO care trebuie cheltuit, primul este 0
1–9 octeți (VarInt)	Dimensiunea Scriptului de Deblocare	Lungimea Scriptului de Deblocare în octeți, care va urma
Variabilă	Script de Deblocare	Un script care îndeplinește condițiile scriptului de blocare al UTXO
4 octeți	Număr de Secvență	De obicei setat pe 0xFFFFFFFF pentru a nu folosi BIP 125 și BIP 68

Table 26. Structura unei intrări de tranzacție coinbase

Dimensiune	Câmp	Descriere
32 octeți	Rezumat (Hash) Tranzacție	Toți biții sunt zero: nu există o referință de rezumat a tranzacției
4 octeți	Indicele de Ieșire	Toți biții sunt același: 0xFFFFFFFF
1–9 octeți (VarInt)	Dimensiunea Coinbase	Lungimea datelor coinbase, de la 2 până la 100 de octeți
Variabilă	Date Coinbase	Date arbitrale utilizate pentru extra nonce și etichete miniere. În blocurile v2; trebuie să înceapă cu înălțimea blocului
4 octeți	Număr de Secvență	Setat la 0xFFFFFFFF

Într-o tranzacție coinbase, primele două câmpuri sunt setate la valori care nu reprezintă o referință UTXO. În loc de un "rezumat de tranzacție", primul câmp este completat cu 32 de octeți setați pe zero. "Indicele de ieșire" este umplut cu 4 octeți, toți setați la 0xFF (255 în zecimal). "Scriptul de deblocare" ([scriptSig](#)) este înlocuit de date coinbase, un câmp de date folosit de mineri, după cum vom vedea în continuare.

## Date Coinbase

(“tranzacții coinbase”, “date coinbase”) Tranzacțiile de tip coinbase nu au un câmp script de deblocare ([scriptSig](#)). În schimb, acest câmp este înlocuit cu date coinbase, care trebuie să aibă între 2 și 100 de octeți. Cu excepția primilor câțiva octeți, restul datelor coinbase pot fi utilizate de mineri în orice fel doresc; este vorba de date arbitrale.

În blocul geneză, de exemplu, Satoshi Nakamoto a adăugat textul "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks" în datele coinbase, folosindu-l ca dovadă a datei și pentru a transmite un mesaj. În prezent, minerii folosesc datele coinbase pentru a include valori nonce și siruri de caractere care identifică bazinul de minerit.

Primii câțiva octeți ai datelor coinbase erau arbitrați, dar nu mai este cazul. Conform BIP-34, blocurile version-2 (blocurile cu câmpul de versiune setat la 2) trebuie să conțină indicele de înălțime al blocului ca operațiune "push" a scriptului la începutul câmpului coinbase.

În blocul 277.316 vedem că datele coinbase (vezi [Tranzacție coinbase](#)), care se află în câmpul de deblocare sau câmpul [scriptSig](#) al intrării tranzacției, conține valoarea hexazecimală `03443b0403858402062f503253482f`. Să decodăm această valoare.

Primul octet, `03`, instruiește motorul de execuție al scriptului să împingă următorii trei octeți pe stivă (vezi [Împinge valoare pe stivă](#)). Următorii trei octeți, `0x443b04`, sunt înălțimea blocului codificată în format little-endian (primul octet cel mai puțin semnificativ). Se inversează ordinea octetilor și rezultatul este `0x043b44`, care este 277.316 în format zecimal.

Următoarele cifre hexazecimale (`0385840206`) sunt folosite pentru a codifica un nonce suplimentar (vezi [Soluția Extra Nonce](#)), sau valoare aleatorie, folosită pentru a găsi o soluție adecvată pentru

Dovada-de-Lucru.

Partea finală a datelor coinbase (*2f503253482f*) este codul ASCII **/P2SH/**, ceea ce indică faptul că nodul care a extras acest bloc acceptă îmbunătățirea P2SH definită în BIP-16. Introducerea funcționalității P2SH a necesitat semnalarea de către mineri pentru a aproba BIP-16 sau BIP-17. Cei care susțin implementarea BIP-16 urmău să includă **/P2SH/** în datele lor coinbase. Cei care susțin implementarea BIP-17 pentru P2SH urmău să includă șirul **p2sh/CHV** în datele lor coinbase. BIP-16 a fost ales câștigător, iar mulți mineri au continuat incluzând șirul **/P2SH/** în datele lor coinbase pentru a indica sprijinul pentru această funcționalitate.

**Extragerea datelor coinbase din blocul geneză** folosește biblioteca libbitcoin introdusă la [Clienți, Biblioteci și Instrumente Alternative](#) pentru a extrage datele coinbase din blocul geneză, afișând mesajul lui Satoshi. Rețineți că biblioteca libbitcoin conține o copie statică a blocului geneză, astfel încât codul poate prelua blocul geneză direct din bibliotecă.

*Example 27. Extragerea datelor coinbase din blocul geneză*

```
/*
  Display the genesis block message by Satoshi.
*/
#include <iostream>
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Create genesis block.
    bc::chain::block block = bc::chain::block::genesis_mainnet();
    // Genesis block contains a single coinbase transaction.
    assert(block.transactions().size() == 1);
    // Get first transaction in block (coinbase).
    const bc::chain::transaction& coinbase_tx = block.transactions()[0];
    // Coinbase tx has a single input.
    assert(coinbase_tx.inputs().size() == 1);
    const bc::chain::input& coinbase_input = coinbase_tx.inputs()[0];
    // Convert the input script to its raw format.
    const auto prefix = false;
    const bc::data_chunk& raw_message = coinbase_input.script().to_data(prefix);
    // Convert this to a std::string.
    std::string message(raw_message.begin(), raw_message.end());
    // Display the genesis block message.
    std::cout << message << std::endl;
    return 0;
}
```

Compilăm codul cu compilatorul GNU C++ și rulăm executabilul rezultat, aşa cum se arată în [Compilarea și executarea codului pentru satoshi-words](#).

Example 28. Compilarea și executarea codului pentru *satoshi-words*

```
$ # Compile the code
$ g++ -o satoshi-words satoshi-words.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Run the executable
$ ./satoshi-words
^D<GS>^A^DEThe Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

## Construirea Antetului Blocului

Pentru a construi antetul blocului, nodul de minerit trebuie să completeze șase câmpuri, aşa cum sunt enumerate în [Structura antetului blocului](#).

Table 27. Structura antetului blocului

Dimensiune	Câmp	Descriere
4 octeți	Versiunea	Un număr de versiune pentru urmărirea actualizărilor de software/protocol
32 octeți	Rezumatul Blocului Anterior	O referință la rezumatul blocului anterior (părinte) din lanț
32 octeți	Rădăcina Merkle	Un rezumat din rădăcina arborelui merkle a tranzacțiilor acestui bloc
4 octeți	Marca de timp	Timpul aproximativ de creare al acestui bloc (secunde de la Unix Epoch)
4 octeți	Țintă	Ținta algoritmului Dovadă-de-Lucru pentru acest bloc
4 octeți	Nonce	Un contor folosit pentru algoritmul Dovadă-de-Lucru

La momentul în care blocul 277.316 a fost minat, numărul de versiune care descrie structura blocului este versiunea 2, care este codată în format little-endian pe 4 octeți ca *0x02000000*.

Apoi, nodul de minerit trebuie să adauge "Rezumatul Blocului Anterior" (cunoscut și ca *prevhash*). Acesta este rezumatul antetului blocului din blocul 277.315, blocul anterior primit de la rețea, pe care nodul lui Jing l-a acceptat și selectat ca *părinte* al blocului candidat 277.316. Rezumatul antetului blocului pentru blocul 277.315 este:

```
0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569
```

**TIP**

Selectând blocul *părinte* specific, indicat de câmpul "Rezumatul Blocului Anterior" din antetul blocului candidat, Jing își pune la bătaie puterea de minerit pentru a extinde lanțul care se termină în acel bloc. În esență, acesta este modul în care Jing "votează" cu puterea sa de minerit pentru lanțul valid cel mai lung.

Următorul pas este să sumarizăm toate tranzacțiile folosind un arbore merkle, pentru a adăuga rădăcina merkle la antetul blocului. Tranzacția coinbase este listată ca prima tranzacție din bloc. Apoi, încă 418 tranzacții sunt adăugate după aceasta, pentru un total de 419 tranzacții în bloc. După cum am văzut la [Arbore Merkle](#), trebuie să existe un număr par de noduri "frunză" în arbore, deci ultima tranzacție este duplicată, creând 420 noduri, fiecare conținând rezumatul unei tranzacții. Rezumatele de tranzacție sunt apoi combinate, în perechi, creând fiecare nivel al arborelui, până când toate tranzacțiile sunt rezumate într-un singur nod la "rădăcina" arborelui. Rădăcina arborelui merkle rezumă toate tranzacțiile într-o singură valoare de 32 de octeți, pe care o puteți vedea listată ca "rădăcină merkle" în [Utilizarea liniei de comandă pentru a prelua blocul 277.316](#), și aici:

```
c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e
```

Nodul miner al lui Jing va adăuga apoi o marcă de timp de 4 octeți, codificată ca o marcă de timp Epocă Unix, care se bazează pe numărul de secunde scurte de la miezul nopții UTC, joi, 1 ianuarie 1970. Timpul *1388185914* este egal cu vineri, 27 decembrie 2013, 23:11:54 UTC.

Nodul lui Jing completează apoi ținta, care definește Dovada-de-Lucru necesară pentru a face acest bloc un bloc valid. Ținta este stocată în bloc sub formă de "biți țintă", care este o codificare exponent-mantisă a țintei. Codificarea are un exponent de 1 octet, urmat de o mantisă (coeficient) pe 3 octeți. În blocul 277.316, de exemplu, valoarea biților țintă este *0x1903a30c*. Prima parte *0x19* este un exponent hexazecimal, în timp ce următoarea parte, *0x03a30c*, este coeficientul. Conceptul de țintă este explicat în [Readaptare pentru Ajustarea Dificultății](#) iar reprezentarea "biți țintă" este explicată în [Reprezentarea Țintei](#).

Câmpul final este nonce-ul, care este inițializat la zero.

Având toate celelalte câmpuri completate, antetul blocului este acum complet și procesul de minerit poate începe. Obiectivul este acum de a găsi o valoare pentru nonce care are ca rezultat un rezumat de antet de bloc care este mai mic decât ținta. Nodul miner va trebui să testeze miliarde sau trilioane de valori nonce înainte de a găsi un nonce care să satisfacă cerința.

## Mineritul Blocului

Acum că un bloc candidat a fost construit de către nodul lui Jing, este timpul ca platforma minieră hardware Jing să "mineze" blocul, pentru a găsi o soluție la algoritmul Dovadă-de-Lucru care face blocul valid. De-a lungul acestei cărți am studiat funcțiile de rezumat criptografic utilizate în diferite aspecte ale sistemului bitcoin. Funcția de rezumare SHA256 este funcția folosită în procesul de minerit bitcoin.

În termenii cei mai simpli, mineritul este procesul de a rezuma în mod repetat antetul blocului, schimbând un parametru, până când rezumatul rezultat se potrivește cu o țintă specifică.

Rezultatul funcției de rezumare nu poate fi determinat în avans și nici nu poate fi creat un tipar care să producă o valoare de rezumat specifică. Această caracteristică a funcțiilor de rezumare înseamnă că singura modalitate de a produce un rezultat de rezumare care se potrivește cu o țintă specifică este să încercați din nou și din nou, modificând aleatoriu intrarea până când rezultatul dorit al rezumatului este descoperit din întâmplare.

## Algoritmul Dovadă-de-Lucru

Un algoritm de rezumare ia o intrare de date cu lungime arbitrară și produce un rezultat determinist de lungime fixă, o amprentă digitală a intrării. Pentru orice intrare specifică, rezumatul rezultat va fi întotdeauna același și poate fi calculat și verificat cu ușurință de către oricine implementează același algoritm de rezumare. Caracteristica cheie a unui algoritm de rezumare criptografic este că este imposibil de calculat găsirea a două intrări diferite care produc aceeași amprentă (cunoscută sub denumirea de *coliziune*). Ca un corolar, este, de asemenea, practic imposibil să selectați o intrare astfel încât să producă o amprentă dorită, în afară de încercarea cu intrări aleatorii.

Cu SHA256, ieșirea are întotdeauna lungimea de 256 biți, indiferent de dimensiunea intrării. În [Exemplu SHA256](#), vom folosi interpretorul Python pentru a calcula rezumatul SHA256 al expresiei "I am Satoshi Nakamoto."

*Example 29. Exemplu SHA256*

```
$ python
Python 2.7.1
>>> import hashlib
>>> print hashlib.sha256("I am Satoshi Nakamoto").hexdigest()
5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e
```

[Exemplu SHA256](#) arată rezultatul calculării rezumatului pentru "I am Satoshi Nakamoto": *5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e*. Acest număr pe 256 biți este *rezumatul* sau *sumarul* frazei și depinde de fiecare parte a frazei. Adăugarea unei singure litere, semn de punctuație sau orice alt caracter va produce un rezumat diferit.

Acum, dacă schimbăm expresia, ar trebui să ne așteptăm să vedem rezumate complet diferite. Să încercăm asta adăugând un număr la sfârșitul frazei noastre, folosind scriptul Python de la [Script SHA256 pentru generarea mai multor rezumate prin iterarea pe un nonce](#).

*Example 30. Script SHA256 pentru generarea mai multor rezumate prin iterarea pe un nonce*

```
# example of iterating a nonce in a hashing algorithm's input

from __future__ import print_function
import hashlib

text = "I am Satoshi Nakamoto"

# iterate nonce from 0 to 19
for nonce in range(20):

    # add the nonce to the end of the text
    input_data = text + str(nonce)

    # calculate the SHA-256 hash of the input (text+nonce)
    hash_data = hashlib.sha256(input_data.encode()).hexdigest()

    # show the input and hash result
    print(input_data, '=>', hash_data)
```

Rularea acestui script va produce rezumatele mai multor fraze, făcute diferite prin adăugarea unui număr la sfârșitul textului. Prin creșterea numărului, putem obține diferite rezumate, aşa cum se arată în [Rezultatul SHA256 a unui script pentru generarea multor rezumate prin iterarea pe un nonce](#).

*Example 31. Rezultatul SHA256 a unui script pentru generarea multor rezumate prin iterarea pe un nonce*

```
$ 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 => 4a2fd48e3be420d0d28e202360cfbaba...  
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...

Fiecare frază produce un rezultat complet diferit al rezumatului. Par complet aleatorii, dar puteți reproduce exact rezultatele din acest exemplu pe orice computer cu Python și puteți vedea exact aceeași rezumate.

Numărul folosit ca variabilă într-un astfel de scenariu se numește *nonce*. Nonce-ul este utilizat pentru a varia ieșirea unei funcții criptografice, în acest caz pentru a varia amprenta SHA256 a frazei.

Pentru a oferi o analogie simplă, imaginăți-vă un joc în care jucătorii aruncă în mod repetat o

pereche de zaruri, încercând să arunce mai puțin decât o țintă specificată. În primul tur, ținta este 12. Dacă nu arunci dublu-șase, câștigi. În runda următoare, ținta este 11. Jucătorii trebuie să arunce 10 sau mai puțin pentru a câștiga, din nou o sarcină ușoară. Să spunem că câteva runde mai târziu ținta va scădea la 5. Acum, mai mult de jumătate din aruncările zarurilor vor depăși ținta și, prin urmare, vor fi invalide. Este nevoie de exponențial mai multe aruncări pentru a câștiga, cu cât ținta este mai mică. În cele din urmă, când ținta este 2 (minimul posibil), o singură aruncare din fiecare 36, sau 2% dintre ele, va produce un rezultat câștigător.

Din perspectiva unui observator care știe că ținta jocului de zaruri este 2, dacă cineva a reușit să aibă o aruncare câștigătoare, se poate presupune că a încercat, în medie, 36 de aruncări. Cu alte cuvinte, se poate estima cantitatea de muncă necesară pentru a reuși din dificultatea impusă de țintă. Când algoritmul se bazează pe o funcție deterministă cum ar fi SHA256, intrarea în sine constituie *dovada* că o anumită cantitate de *lucru* a fost depusă pentru a produce un rezultat sub țintă. Deci, *Dovada-de-Lucru*.

**TIP** Chiar dacă fiecare încercare produce un rezultat aleatoriu, probabilitatea oricăror rezultate posibile poate fi calculată în avans. Prin urmare, un rezultat de dificultate specificată constituie dovada unei cantități specifice de lucru.

În [Rezultatul SHA256 a unui script pentru generarea multor rezumate prin iterarea pe un nonce](#), nonce-ul câștigător este 13 și acest rezultat poate fi confirmat de oricine în mod independent. Oricine poate adăuga numărul 13 ca sufix la expresia "I am Satoshi Nakamoto" și poate calcula rezumatul, verificând că acesta este mai mic decât ținta. Rezultatul este, de asemenea, Dovada-de-Lucru, deoarece dovedește că am făcut munca pentru a găsi acel nonce. În timp ce este nevoie de un singur calcul de rezumate pentru a verifica, ne-a luat 13 calcule de rezumare pentru a găsi un nonce care a funcționat. Dacă am avea o țintă mai mică (dificultate mai mare), ar fi nevoie de multe alte calcule de rezumare pentru a găsi un nonce adecvat, dar un singur calcul de rezumare pe care oricine îl poate verifica. Mai mult, cunoscând ținta, oricine poate estima dificultatea făcând niște calcule statistice și, prin urmare, știe cât de multă muncă a fost necesară pentru a găsi un astfel de nonce.

**TIP** Dovada-de-Lucru trebuie să producă un rezumat care este *mai mic decât* ținta. O țintă mai mare înseamnă că este mai puțin dificil să găsiți un rezumat care este sub țintă. O țintă mai mică înseamnă că este mai dificil să găsiți un rezumat sub țintă. Ținta și dificultatea sunt invers corelate.

Dovada-de-Lucru în bitcoin este foarte similară cu provocarea prezentată în [Rezultatul SHA256 a unui script pentru generarea multor rezumate prin iterarea pe un nonce](#). Minerul construiește un bloc candidat completat cu tranzacții. În continuare, minerul calculează rezumatul din antetul acestui bloc și vede dacă este mai mic decât *ținta* curentă. În cazul în care rezumatul nu este mai mic decât ținta, minerul va modifica nonce-ul (de obicei doar îl incrementează) și va încerca din nou. La dificultatea actuală a rețelei bitcoin, minerii trebuie să încerce de cinciilioane de ori înainte de a găsi un nonce care are ca rezultat un rezumat al antetului de bloc suficient de scăzut.

Un algoritm foarte simplificat de Dovadă-de-Lucru este implementat în Python în [Implementarea simplificată pentru Dovada-de-Lucru](#).

Example 32. Implementarea simplificată pentru Dovada-de-Lucru

```
#!/usr/bin/env python
# example of proof-of-work algorithm

import hashlib
import time

try:
    long      # Python 2
    xrange
except NameError:
    long = int # Python 3
    xrange = range

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)).encode()).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)

```

Rulând acest cod, puteți seta dificultatea dorită (în biți, câți dintre biții de la început trebuie să fie zero) și să vedeți cât durează calculul pentru a găsi o soluție. În [Rularea exemplului Dovada-de-Lucru pentru diferite dificultăți](#), puteți vedea cum funcționează pe un laptop mediu.

*Example 33. Rularea exemplului Dovada-de-Lucru pentru diferite dificultăți*

```
$ 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 029ae6e5004302a120630adcbb808452346ab1cf0b94c5189ba8bac1d47e7903

Elapsed Time: 0.0006 seconds

Hashing Power: 58164 hashes per second

[...]

Difficulty: 4194304 (22 bits)

Starting search...

Success with nonce 1759164

Hash is 0000008bb8f0e731f0496b8e530da984e85fb3cd2bd81882fe8ba3610b6cefc3

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
```

După cum vedeați, creșterea dificultății cu 1 bit provoacă o dublare a timpului necesar pentru a găsi o soluție. Dacă vă gândiți la întreg spațiul numeric de 256 biți, de fiecare dată când restricționați încă un bit la zero, reduceți spațiul de căutare cu jumătate. În [Rularea exemplului Dovada-de-Lucru pentru diferite dificultăți](#), este nevoie de 84 de milioane de încercări de rezumare pentru a găsi un nonce care produce un rezumat care începe cu 26 de biți de zero. Chiar și cu o viteză de peste 120.000 de rezumări pe secundă, este nevoie de 10 minute pe un laptop pentru a găsi această soluție.

La momentul scrierii, rețeaua încearcă să găsească un bloc al cărui rezumat de antet este mai mic decât:

După cum puteți vedea, la începutul acestei ținte există o mulțime de zerouri, ceea ce înseamnă că gama acceptabilă de rezumate este mult mai mică, prin urmare, este mai dificil să găsiți un rezumat valid. Va necesita în medie mai mult de 1,8 zeta-rezumări (mii de miliarde de rezumări) pe secundă pentru ca rețeaua să descopere următorul bloc. Aceasta pare a fi o sarcină imposibilă, dar, din fericire, rețeaua produce 3 exa-rezumate pe secundă (exa-hashes per second - EH/sec) de putere de procesare, care va putea găsi un bloc în medie la aproximativ 10 minute.

## Reprezentarea Tintei

În [Utilizarea liniei de comandă pentru a preluă blocul 277.316](#), am văzut că blocul conține ținta, într-o notație numită "biți țintă" sau doar "biți", care în blocul 277.316 are valoarea de  $0x1903a30c$ . Această notare exprimă obiectivul Dovadă-de-Lucru sub forma coeficient/exponent, cu primele două cifre hexazecimale pentru exponent și următoarele șase cifre hexazecimale pentru coeficient. Prin urmare, în acest bloc, exponentul este  $0x19$  iar coeficientul este  $x03a30c$ .

Formula pentru a calcula tinta de dificultate din această reprezentare este:

i.  $tinta = coeficiente * 2^{(8 * (exponente - 3))}$

Folosind formula respectivă iar valoarea bitilor de dificultate fiind 0x1903a30c, obținem:

i.  $tinta = 0x03a30c * 2^{0x08} * (0x19-0x03)^8$

ii  $\Rightarrow$  tintab  $\equiv 0x03a30c * 2^{(0x08*0x16)}$

iii  $\Rightarrow$  tinta = 0x03a30c \* 2<sup>0xB0</sup>

care în format zecimal este:

$$\text{i. } \Rightarrow \text{tinta} = 238,348 * 2^{176}$$

ii.  $\Rightarrow t_{\text{inta}} = 22,829,202,948,393,929,850,749,706,076,701,368,331,072,452,018,388,575,715,328$

revenind la hexazecimal:

Aceasta înseamnă că un bloc valid pentru înălțimea 277.316 este unul care are un rezumat de antet de bloc care este mai mic decât ţinta. În format binar, acest număr trebuie să înceapă cu peste 60 de biți setați pe zero. Cu acest nivel de dificultate, un singur miner care prelucrează 1 trilion de rezumări pe secundă (1 terahash pe secundă sau 1 TH/sec) ar găsi în medie o soluție doar o dată la 8.496 blocuri sau o dată la 59 de zile.

## Readaptare pentru Ajustarea Dificultății

După cum am văzut, ținta determină dificultatea și, prin urmare, afectează timpul necesar pentru a găsi o soluție la algoritmul Dovadă-de-Lucru. Acest fapt ridică unele întrebări evidente: de ce este ajustabilă dificultatea, cine o ajustează și cum?

Blocurile bitcoin sunt generate în medie la fiecare 10 minute. Acesta este pulsul bitcoin și stă la baza frecvenței emiterii de monedă și vitezei de confirmare a tranzacțiilor. Trebuie să rămână constant nu doar pe termen scurt, ci pe o perioadă de multe decenii. În acest timp, este de așteptat ca puterea de calcul să crească în continuare într-un ritm rapid. Mai mult, numărul de participanți în procesul de minerit și calculatoarele pe care le folosesc se vor schimba în mod constant. Pentru a menține timpul de generare a blocurilor la 10 minute, dificultatea trebuie ajustată pentru a ține cont de aceste modificări. De fapt, ținta Dovadă-de-Lucru este un parametru dinamic care este ajustat periodic pentru a îndeplini obiectivul de un bloc la 10 minute. În termeni simpli, ținta este stabilită astfel încât puterea de minerit curentă să rezulte la un interval de un bloc la 10 minute.

Atunci, cum se face o astfel de ajustare într-o rețea complet descentralizată? Readaptarea se face automat și pe fiecare nod independent. La fiecare 2.016 blocuri, toate nodurile readaptează Dovada-de-Lucru. Ecuația pentru readaptare măsoară timpul necesar pentru a găsi ultimele 2.016 blocuri și le compara cu timpul preconizat de 20.160 minute (2.016 blocuri la intervalul dorit de 10 minute). Se calculează raportul dintre durata efectivă și durata dorită și se face o adaptare proporțională (în sus sau în jos) a țintei. În termeni simpli: dacă rețeaua găsește blocuri mai repede decât la fiecare 10 minute, dificultatea crește (ținta scade). Dacă descoperirea blocurilor este mai lentă decât se aștepta, dificultatea scade (ținta crește).

Ecuăția poate fi sumarizată ca:

Ținta Nouă = Ținta Veche \* (Durata Efectivă a Ultimelor 2016 Blocuri / 20160 minute)

Readaptarea Dovetii-de-Lucru - CalculateNextWorkRequired () în pow.cpp arată codul utilizat în clientul Bitcoin Core.

```
// Limit adjustment step
int64_t nActualTimespan = pindexLast->GetBlockTime() - nFirstBlockTime;
LogPrintf(" nActualTimespan = %d before bounds\n", nActualTimespan);
if (nActualTimespan < params.nPowTargetTimespan/4)
    nActualTimespan = params.nPowTargetTimespan/4;
if (nActualTimespan > params.nPowTargetTimespan*4)
    nActualTimespan = params.nPowTargetTimespan*4;

// Retarget
const arith_uint256 bnPowLimit = UintToArith256(params.powLimit);
arith_uint256 bnNew;
arith_uint256 bnOld;
bnNew.SetCompact(pindexLast->nBits);
bnOld = bnNew;
bnNew *= nActualTimespan;
bnNew /= params.nPowTargetTimespan;

if (bnNew > bnPowLimit)
    bnNew = bnPowLimit;
```

**NOTE**

În timp ce calibrarea țintei are loc la fiecare 2.016 blocuri, din cauza unei erori în clientul Bitcoin Core original, aceasta se bazează pe timpul total al celor 2.015 blocuri anterioare (nu 2.016 aşa cum ar trebui să fie), rezultând o eroare de readaptare către o dificultate mai mare cu 0,05%.

Parametrii **Interval** (2.016 blocuri) și **TargetTimespan** (două săptămâni sub forma de 1.209.600 secunde) sunt definiți în *chainparams.cpp*.

Pentru a evita volatilitatea extremă în dificultate, ajustarea țintei trebuie să fie mai mică decât un multiplu de patru (4) pe ciclu. Dacă ajustarea necesară țintei este mai mare decât un multiplu de patru, aceasta va fi ajustată cu un multiplu de 4 și nu mai mult. Orice ajustare suplimentară se va realiza în următoarea perioadă de readaptare, deoarece dezechilibrul va persista pe durata următoarelor 2.016 blocuri. Prin urmare, discrepanțele mari între puterea de rezumare și dificultate ar putea dura câteva cicluri de 2.016 blocuri pentru a se echilibra.

**TIP**

Dificultatea de a mina un bloc bitcoin este de aproximativ "10 minute de procesare" pentru întreaga rețea, pe baza timpului necesar pentru a mina precedentele 2.016 blocuri. Acest lucru se realizează prin scăderea sau ridicarea țintei.

Rețineți că ținta este independentă de numărul de tranzacții sau de valoarea tranzacțiilor. Aceasta înseamnă că volumul de putere de rezumare și, prin urmare, energia electrică cheltuită pentru a securiza bitcoin este, de asemenea, complet independentă de numărul de tranzacții. Bitcoin poate scala, poate fi adoptat pe scară largă și poate rămâne în siguranță fără o creștere a puterii de rezumare de la nivelul actual. Creșterea puterii de rezumare reprezintă forțele pieței, deoarece noi mineri intră pe piață pentru a concura pentru recompensă. Atâtă timp cât suficientă putere de

rezumare ramâne sub controlul minerilor care acționează cinstit în urmărirea recompensei, este suficient pentru a preveni atacurile de "preluare" și, prin urmare, este suficient pentru a securiza bitcoin.

Dificultatea mineritului este strânsă legată de costul energiei electrice și de cursul de schimb al bitcoin față de moneda utilizată pentru plata energiei electrice. Sistemele miniere de înaltă performanță sunt cât se poate de eficiente cu generația actuală de hardware, transformând energia electrică în calculul de rezumare la cea mai mare rată posibilă. Influența principală pe piața minieră este prețul unui kilowatt-oră de energie electrică, deoarece acest lucru determină rentabilitatea mineritului și, prin urmare, stimulele pentru a intra sau ieși de pe piața de minerit.

## Mineritul cu Succes al Blocului

Așa cum am văzut mai devreme, nodul lui Jing a construit un bloc candidat și l-a pregătit pentru minerit. Jing are mai multe platforme miniere hardware, unde sute de mii de circuite integrate rulează algoritmul SHA256 în paralel la viteze incredibile. Multe dintre aceste mașini specializate sunt conectate la nodul său miner prin USB sau o rețea locală. Apoi, nodul de exploatare care rulează pe desktopul lui Jing transmite antetul blocului la hardware-ul său de minerit, care începe testarea a trilioane de nonce-uri pe secundă. Deoarece nonce-ul are doar 32 de biți, după epuizarea tuturor posibilităților de nonce (aproximativ 4 miliarde), hardware-ul de minerit schimbă antetul blocului (ajustând extra nonce-ul din coinbase sau marca de timp) și resetează contorul nonce, testând noi combinații.

La aproape 11 minute de la începerea minării blocului 277.316, una dintre mașinile de minerit hardware găsește o soluție și o trimite înapoi la nodul miner.

Când este inserat în antetul blocului, nonce-ul 924.591.752 produce un rezumat de bloc de:

0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4

care este mai mic decât ținta:

Imediat, nodul miner al lui Jing transmite blocul tuturor semenilor săi. Ei primesc, validează și apoi propagă noul bloc. Pe măsură ce blocul se propagă prin rețea, fiecare nod îl adaugă la propria sa copie a lanțului-de-blocuri, extinzând lanțul la o nouă înălțime de 277.316 blocuri. Pe măsură ce nodurile minere primesc și validează blocul, își abandonează eforturile pentru a găsi un bloc la aceeași înălțime și încep imediat să calculeze următorul bloc din lanț, folosind blocul lui Jing drept "părinte". Construind peste blocul recent descoperit al lui Jing, ceilalți mineri "votează" în esență cu puterea lor minieră și aprobă blocul lui Jing și lanțul pe care îl extinde.

În următoarea secțiune, vom analiza procesul pe care fiecare nod îl folosește pentru validarea unui bloc și selectarea celui mai lung lanț, creând consensul care formează lanțul-de-blocuri descentralizat.

## Validarea unui Bloc Nou

Al treilea pas în mecanismul de consens al bitcoin este validarea independentă a fiecărui nou bloc de către fiecare nod din rețea. Pe măsură ce blocul nou rezolvat se deplasează prin rețea, fiecare nod efectuează o serie de teste pentru a-l valida înainte de a-l propaga la semenii săi. Acest lucru asigură că numai blocurile valide sunt propagate în rețea. Validarea independentă asigură, de asemenea, că minerii care acționează cinstit vor avea blocurile lor încorporate în lanțul-de-blocuri, obținând astfel recompensa. Acei mineri care acționează necinstit vor avea blocurile respinse și nu numai că pierd recompensa, dar își și risipesc eforturile depuse pentru a găsi o soluție Dovadă-de-Lucru, suportând astfel costul energiei electrice fără compensații.

Când un nod primește un bloc nou, acesta va valida blocul verificând o listă lungă de criterii care trebuie îndeplinite toate; în caz contrar, blocul este respins. Aceste criterii pot fi văzute în clientul Bitcoin Core în funcțiile `CheckBlock` și `CheckBlockHeader` și includ:

- Structura datelor blocului este validă sintactic
- Rezumatul antetului blocului este mai mic decât ţinta (impune Dovada-de-Lucru)
- Marca de timp a blocului este mai puțin de două ore în viitor (permășând erori de timp)
- Dimensiunea blocului este în limite acceptabile
- Prima tranzacție (și numai prima) este o tranzacție coinbase
- Toate tranzacțiile din bloc sunt valide folosind lista de verificare a tranzacțiilor discutată în [Verificarea Independentă a Tranzacțiilor](#)

Validarea independentă a fiecărui nou bloc de către fiecare nod din rețea asigură că minerii nu pot însela. În secțiunile anterioare am văzut cum minerii ajung să scrie o tranzacție care să le acorde bitcoin nou creat în cadrul blocului și să solicite comisioane de tranzacție. De ce minerii nu își scriu singuri o tranzacție pentru o mie de bitcoin în loc de o recompensă corectă? Deoarece fiecare nod validează blocurile conform acelorași reguli. O tranzacție coinbase invalidă ar face ca întregul bloc să fie invalid, ceea ce ar duce la respingerea blocului și, prin urmare, acea tranzacție nu va deveni niciodată parte a registrului. Minerii trebuie să construască un bloc perfect, bazat pe regulile pe care le respectă toate nodurile și să-l mineze cu o soluție corectă pentru Dovada-de-Lucru. Pentru a face acest lucru, ei cheltuiesc multă energie electrică pentru minerit, iar dacă trișează, toată energia electrică și efortul sunt pierdute. Acesta este motivul pentru care validarea independentă este o componentă cheie a consensului descentralizat.

## Asamblarea și Selectarea Lanțurilor-de-Blocuri

Ultimul pas în mecanismul de consens descentralizat al bitcoin este asamblarea blocurilor în lanțuri și selectarea lanțului cu cele mai multe Dovezi-de-Lucru. După ce un nod a validat un nou bloc, va încerca apoi să asambleze un lanț conectând blocul la lanțul-de-blocuri existent.

Nodurile mențin trei seturi de blocuri: cele conectate la lanțul-de-blocuri principal, cele care formează ramuri în afara lanțului-de-blocuri principal (lanțuri secundare) și, în final, blocuri care nu au un părinte cunoscut în lanțurile cunoscute (orfani). Blocurile invalide sunt respinse imediat ce unul dintre criteriile de validare eșuează și, prin urmare, nu sunt incluse în niciun lanț.

În orice moment, "lanțul principal" este lanțul-de-blocuri *valid* care are cea mai multă Dovadă-de-Lucru cumulată. În majoritatea circumstanțelor, acesta este și lanțul cu cele mai multe blocuri, cu excepția cazului în care există două lanțuri de lungime egală și unul are mai multă Dovadă-de-Lucru. Lanțul principal va avea și ramuri cu blocuri care sunt "frați" ai blocurilor din lanțul principal. Aceste blocuri sunt valide, dar nu fac parte din lanțul principal. Acestea sunt păstrate pentru referințe viitoare, în cazul în care unul dintre aceste lanțuri este extins pentru a depăși lanțul principal în lucru. În secțiunea următoare ([Bifurcări ale Lanțului-de-Blocuri](#)), vom vedea cum se produc lanțuri secundare ca urmare a unei minări de blocuri aproape simultane la aceeași înălțime.

Când un nou bloc este primit, un nod va încerca să îl introducă în lanțul-de-blocuri existent. Nodul va verifica câmpul "rezumat bloc anterior", care este referința la părintele blocului. Apoi, nodul va încerca să găsească acel părinte în lanțul-de-blocuri existent. De cele mai multe ori, părintele va fi "vârful" lanțului principal, adică acest nou bloc extinde lanțul principal. De exemplu, noul bloc 277.316 are o referință la rezumatul blocului său părinte 277.315. Majoritatea nodurilor care primesc 277.316 vor avea deja blocul 277.315 ca vârful lanțului lor principal și, prin urmare, vor lega noul bloc și vor extinde acel lanț.

Uneori, aşa cum vom vedea în [Bifurcări ale Lanțului-de-Blocuri](#), noul bloc extinde un lanț care nu este lanțul principal. În acest caz, nodul va atașa noul bloc la lanțul secundar pe care îl extinde și apoi va compara munca lanțului secundar cu lanțul principal. Dacă lanțul secundar are mai multă Dovadă-de-Lucru cumulată decât lanțul principal, nodul va converge la lanțul secundar, ceea ce înseamnă că va selecta lanțul secundar ca noul său lanț principal, făcând din vechiul lanț principal un lanț secundar. Dacă nodul este un miner, acum va construi un bloc care extinde acest nou lanț, mai lung.

Dacă se primește un bloc valid și nu se găsește niciun părinte în lanțurile existente, acel bloc este considerat "orfan". Blocurile orfane sunt salvate în bazinul de blocuri orfane, unde vor rămâne până la primirea părintelui lor. Odată ce părintele este primit și legat în lanțurile existente, orfanul poate fi scos din bazinul de orfani și legat de părinte, făcându-l să facă parte dintr-un lanț. Blocurile orfane apar de obicei atunci când două blocuri minate într-un timp scurt între ele sunt primite în ordine inversă (copil înainte de părinte).

Prin selectarea lanțului valid cea mai mare Dovadă-de-Lucru cumulată, toate nodurile ajung la un consens la nivel de rețea. Discrepanțele temporare între lanțuri sunt soluționate în cele din urmă, pe măsură ce se adaugă mai multă Dovadă-de-Lucru, extinzând unul dintre lanțurile posibile. Nodurile miner "votează" cu puterea lor minieră alegând ce lanț va fi extins prin minarea următorului bloc. Când extrag un nou bloc și extind lanțul, noul bloc reprezintă votul lor.

În secțiunea următoare vom analiza modul în care discrepanțele dintre lanțurile (bifurcările) concurente sunt rezolvate prin selecția independentă a lanțului cu cea mai mare Dovadă-de-Lucru cumulată.

## Bifurcări ale Lanțului-de-Blocuri

Deoarece lanțul-de-blocuri este o structură de date descentralizată, diferitele copii ale acestuia nu sunt întotdeauna consistente. Blocurile ar putea ajunge la noduri diferite în momente diferite, ceea ce face ca nodurile să aibă perspective diferite asupra lanțului-de-blocuri. Pentru a rezolva acest lucru, fiecare nod selectează și încearcă întotdeauna să extindă lanțul-de-blocuri care reprezintă

cea mai mare Dovadă-de-Lucru, cunoscut și ca cel mai lung lanț sau cel mai mare lanț cumulativ de lucru. Prin însumarea Dovezilor-de-Lucru înregistrate în fiecare bloc dintr-un lanț, un nod poate calcula cantitatea totală de muncă care a fost depusă pentru a crea acel lanț. Atâtă timp cât toate nodurile selectează lanțul cu cea mai mare Dovadă-de-Lucru cumulată, rețeaua bitcoin globală ajunge în cele din urmă la o stare consistentă. Bifurcările apar ca neconcordanțe temporare între versiunile lanțului-de-blocuri, care sunt rezolvate prin eventuala reconvergență, pe măsură ce se adaugă mai multe blocuri la una dintre bifurcări.

**TIP** Bifurcările lanțului-de-blocuri descrise în această secțiune apar în mod natural ca urmare a întârzierilor de transmitere în rețeaua globală. De asemenea, vom analiza bifurcările induse în mod deliberat mai târziu în acest capitol.

În următoarele diagrame, urmărim progresul unui eveniment de "bifurcare" în rețea. Diagrama este o reprezentare simplificată a rețelei bitcoin. În scop ilustrativ, blocuri diferite sunt prezentate sub forme diferite (stea, triunghi, triunghi cu susul în jos, romb), răspândite în rețea. Fiecare nod din rețea este reprezentat ca un cerc.

Fiecare nod are propria perspectivă asupra lanțului-de-blocuri global. Pe măsură ce fiecare nod primește blocuri de la vecinii săi, acesta își actualizează propria copie a lanțului-de-blocuri, selectând lanțul cu cea mai mare Dovadă-de-Lucru. În scop ilustrativ, fiecare nod conține o formă geometrică care reprezintă blocul care crede că este în prezent vârful lanțului principal. Deci, dacă vedeați o stea în nod, asta înseamnă că blocul stea este vârful lanțului principal, în ceea ce privește acel nod.

În prima diagramă ([Înainte de bifurcare - toate nodurile au aceeași perspectivă](#)), rețeaua are o perspectivă unificată asupra lanțului-de-blocuri, cu blocul stea drept vârful lanțului principal.

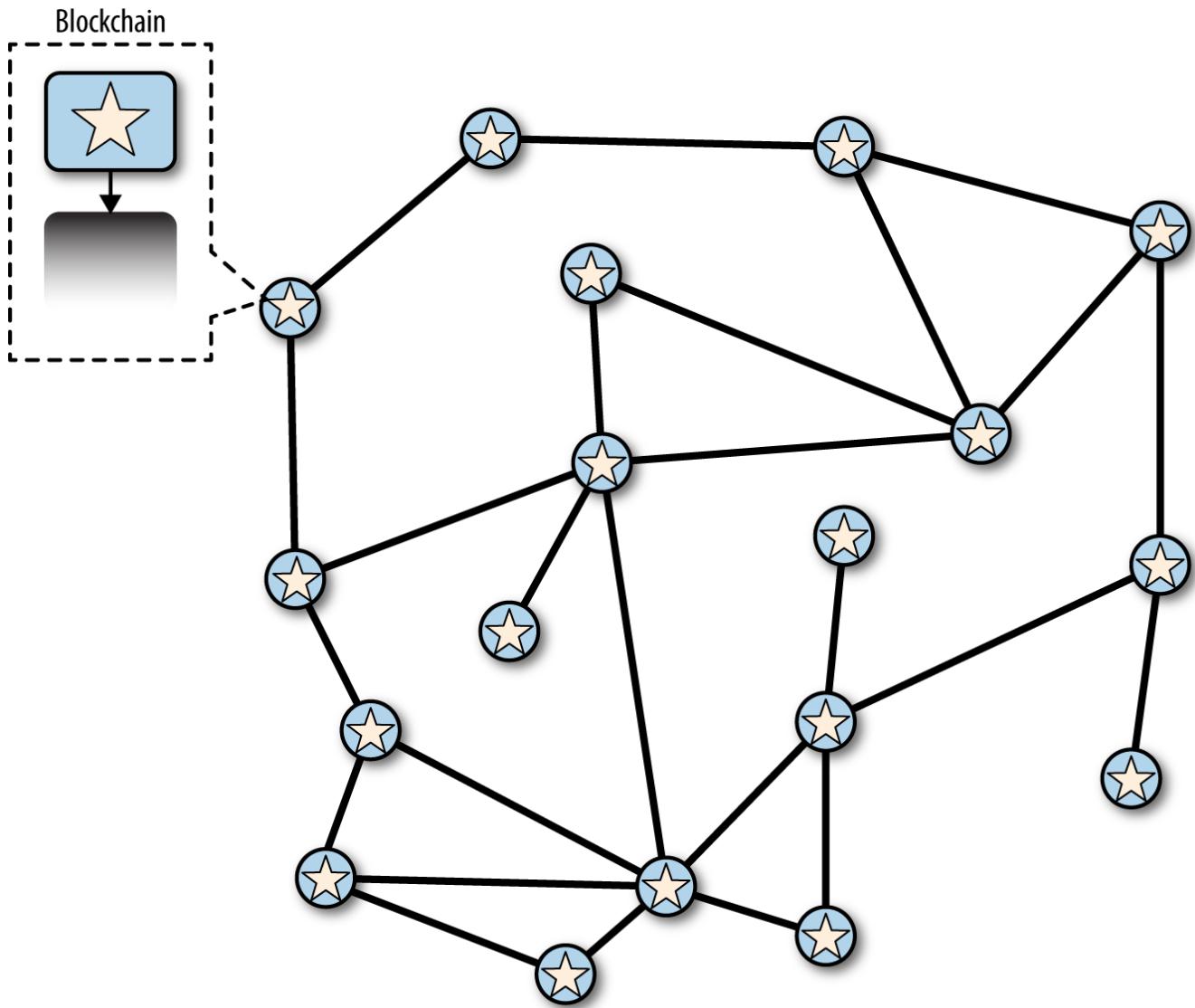


Figure 67. Înainte de bifurcare - toate nodurile au aceeași perspectivă

O "bifurcare" apare ori de câte ori există două blocuri candidat care concurează pentru a forma cel mai lung lanț-de-blocuri. Aceasta se întâmplă în condiții normale ori de câte ori doi mineri rezolvă algoritmul Dovadă-de-Lucru într-o perioadă scurtă de timp unul față de celălalt. Pe măsură ce ambii mineri descoperă o soluție pentru blocurile lor candidat, aceștia își transmit imediat propriul bloc "câștigător" către vecinii lor, iar vecinii încep să propage blocul în rețea. Fiecare nod care primește un bloc valid îl va încorpora în lanțul-de-blocuri, extinzând lanțul-de-blocuri cu un singur bloc. Dacă acel nod vede mai târziu un alt bloc candidat care extinde același părinte, acesta conectează al doilea candidat pe un lanț secundar. Drept urmare, unele noduri vor "vedea" mai întâi un bloc candidat, în timp ce altele vor vedea celălalt bloc candidat și vor apărea două versiuni concurente ale lanțului-de-blocuri.

În [Vizualizarea unui eveniment de bifurcare în lanțul-de-blocuri: două blocuri găsite simultan](#), vedem doi mineri (nodul X și nodul Y) care minează două blocuri diferite aproape simultan. Ambele blocuri sunt copii (urmași) ale blocului stea și extind lanțul construind deasupra blocului stea. Pentru a ne ajuta să le urmărim, unul este prezentat ca un bloc triunghi pornind din nodul X, iar celălalt este prezentat ca un bloc triunghi cu capul în jos pornind din nodul Y.

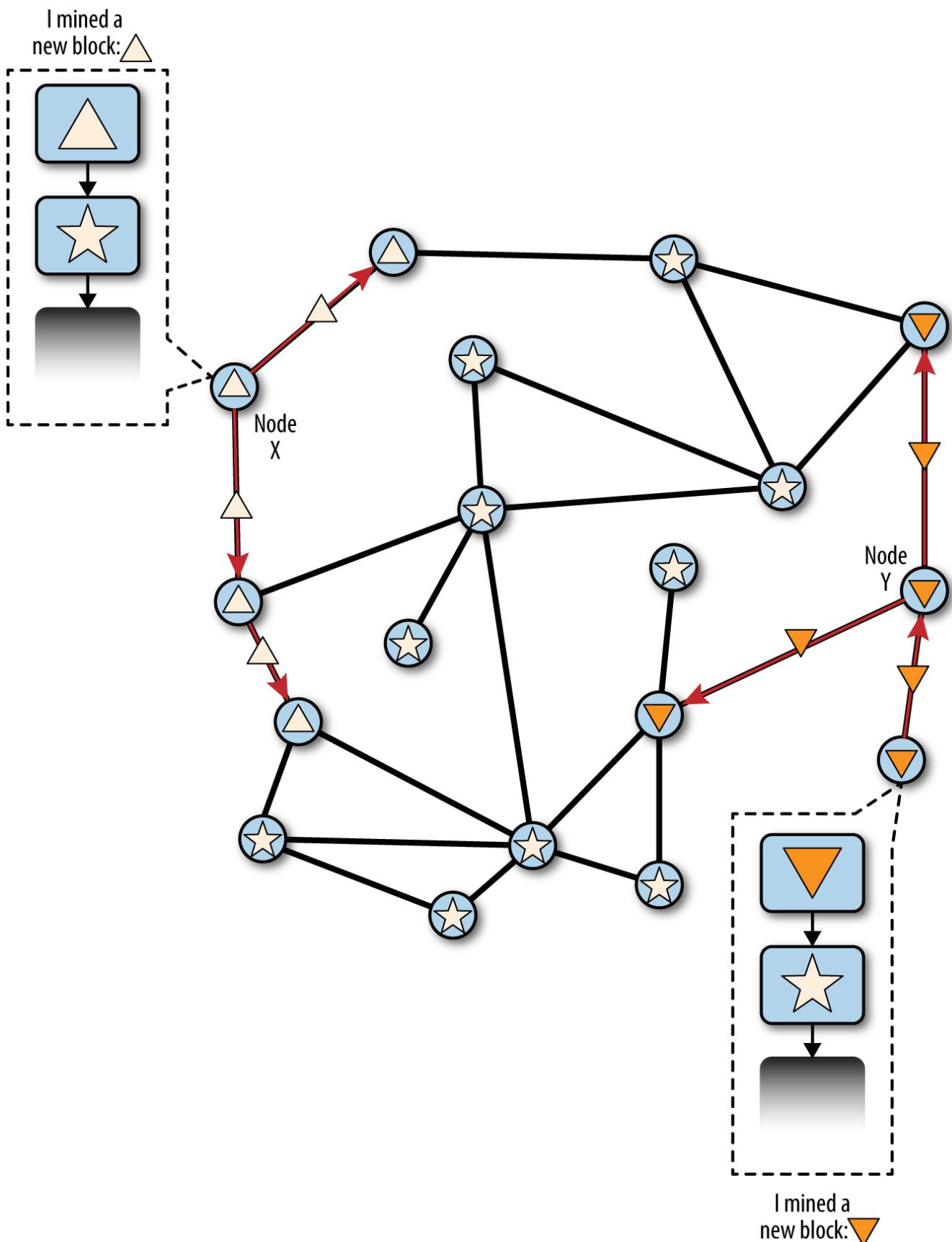


Figure 68. Vizualizarea unui eveniment de bifurcare în lanțul-de-blocuri: două blocuri găsite simultan

Să presupunem, de exemplu, că un nod miner X găsește o soluție Dovadă-de-Lucru pentru un "triunghi" care extinde lanțul-de-blocuri, construind deasupra blocui părinte stea. Aproape simultan, un nod miner Y, care extindea de asemenea lanțul de la blocul "stea", găsește o soluție pentru blocul "triunghi cu susul în jos", blocul său candidat. Acum, există două blocuri posibile; unul pe care îl numim "triunghi", originar din nodul X; și unul pe care îl numim "triunghi cu susul

în jos”, originar din nodul Y. Ambele blocuri sunt valide, ambele blocuri conțin o soluție validă pentru Dovada-de-Lucru și ambele blocuri extind același părinte (blocul ”stea”). Ambele blocuri conțin probabil în cea mai mare parte aceleasi tranzacții, cu poate doar câteva diferențe în ordinea tranzacțiilor.

Pe măsură ce cele două blocuri se propagă, unele noduri primesc mai întâi blocul ”triunghi”, iar altele primesc mai întâi blocul ”triunghiul cu susul în jos”. Așa cum se arată în [Vizualizarea unui eveniment de bifurcare a lanțului-de-blocuri: două blocuri se propagă, împărțind rețeaua, rețeaua se împarte în două perspective diferite asupra lanțului-de-blocuri](#); o parte este acoperită cu un bloc triunghi, cealaltă cu un bloc triunghi cu susul în jos.

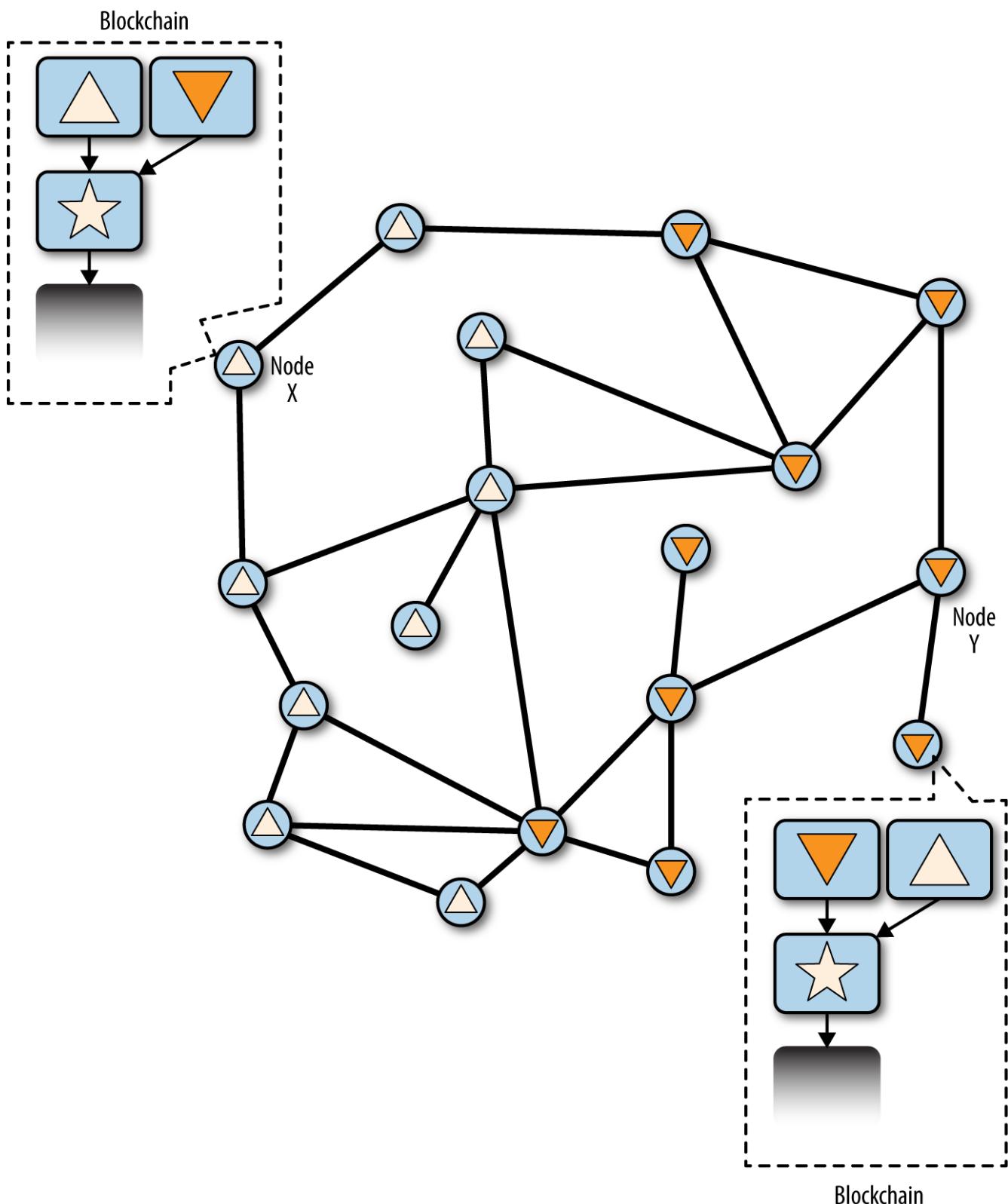


Figure 69. Vizualizarea unui eveniment de bifurcare a lanțului-de-blocuri: două blocuri se propagă, împărțind rețeaua

În diagramă, un nodul X ales la întâmplare a primit mai întâi blocul triunghi și a extins lanțul stea cu acesta. Nodul X a ales lanțul cu blocul "triunghi" ca lanț principal. Mai târziu, nodul X a primit și blocul "triunghi cu susul în jos". Deoarece a fost primit al doilea, se presupune că a "pierdut" cursa. Cu toate acestea, blocul "triunghi cu susul în jos" nu este aruncat. Este legat de părintele bloc "stea" și formează un lanț secundar. Chiar dacă nodul X presupune că a selectat corect lanțul câștigător, acesta păstrează lanțul "pierzător", astfel încât are informațiile necesare pentru a reconverge dacă lanțul "pierzător" ajunge să "câștige".

De cealaltă parte a rețelei, nodul Y construiește un lanț-de-blocuri bazat pe propria perspectivă a secvenței de evenimente. A primit întâi "triunghi cu susul în jos" și a ales acel lanț drept "câștigător". Când ulterior a primit blocul "triunghi", l-a conectat la blocul "stea", ca un lanț secundar.

Nicio parte nu este "corectă" sau "incorectă". Ambele sunt perspective valide ale lanțului-de-blocuri. Doar unul va triumfa, bazat pe modul în care aceste două lanțuri concurente sunt extinse prin eforturi suplimentare.

Nodurile minere a căror perspectivă seamănă cu cea a nodului X vor începe imediat extragerea unui bloc candidat care extinde lanțul cu "triunghi" ca vârf. Prin legarea "triunghi" ca părinte al blocului candidat, aceștia votează cu puterea de calcul. Votul lor susține lanțul pe care l-au ales ca lanț principal.

Orice nod miner a cărui perspectivă seamănă cu cea a nodului Y va începe să construiască un nod candidat având "triunghiul cu susul în jos" ca părinte, extinzând lanțul care cred că este lanțul principal. Și uite aşa, cursa începe din nou.

Bifurcările sunt aproape întotdeauna rezolvate după un singur bloc. În timp ce o parte a puterii de calcul a rețelei este dedicată construirii deasupra "triunghiului" ca părinte, o altă parte a puterii de calcul este concentrată pe construirea deasupra "triunghiului cu susul în jos". Chiar dacă puterea de calcul este divizată aproape uniform, este probabil ca un grup de mineri să găsească o soluție și să o propage înainte ca celălalt grup de mineri să găsească vreo soluție. Să spunem, de exemplu, că minerii care construiesc deasupra "triunghiului" găsesc un nou block "romb" care extinde lanțul (de exemplu, stea-triunghi-romb). Ei propagă imediat acest nou bloc și întreaga rețea îl vede ca o soluție validă, aşa cum se arată în [Vizualizarea unui eveniment de bifurcare a lanțului-de-blocuri: un bloc nou extinde o bifurcare, reconvergența rețelei](#).

Toate nodurile care au ales "triunghiul" drept câștigător în runda precedentă vor extinde pur și simplu lanțul cu încă un bloc. Nodurile care au ales "triunghiul cu susul în jos" drept câștigător, vor vedea acum două lanțuri: stea-triunghi-romb și stea-triunghiul-cu-susul-în-jos. Lanțul stea-triunghi-romb este acum mai lung (mai mult efort cumulat) decât celălalt lanț. Drept urmare, acele noduri vor seta lanțul stea-triunghi-romb ca lanț principal și vor schimba lanțul stea-triunghiul-cu-susul-în-jos într-un lanț secundar, aşa cum se arată în [Vizualizarea unui eveniment de bifurcare a lanțului-de-blocuri: rețeaua reconverge pe un nou lanț mai lung](#). Aceasta este o reconvergență a lanțului, deoarece acele noduri sunt obligate să își revizuiască viziunea asupra lanțului-de-blocuri pentru a încorpora noile dovezi ale unui lanț mai lung. Toți minerii care lucrează la extinderea lanțului stea-triunghiul-cu-susul-în-jos vor opri acum această muncă, deoarece blocul lor candidat este "orfan", întrucât părintele său "triunghiul-cu-susul-în-jos" nu mai este pe cel mai lung lanț. Tranzacțiile din "triunghiul-cu-susul-în-jos" care nu se află în "triunghi" sunt re-inserate în mempool pentru a fi incluse în următorul bloc pentru a deveni parte a lanțului principal. Întreaga rețea reconverge spre un singur lanț-de-blocuri stea-triunghi-romb, având "romb" ca ultimul bloc din lanț. Toți minerii încep imediat să lucreze la blocurile candidat care referențiază "romb" ca părinte pentru a extinde lanțul stea-triunghi-romb.

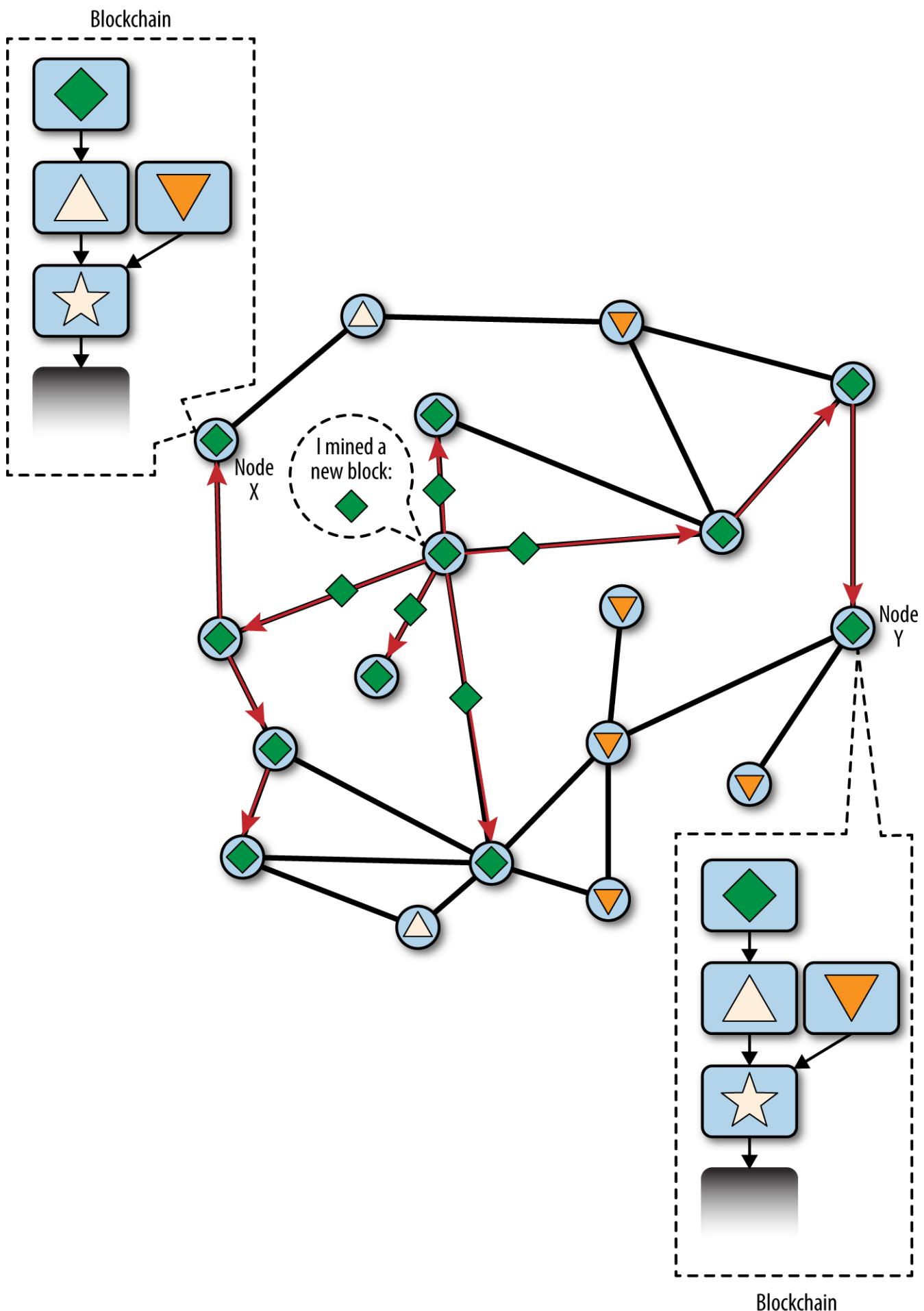


Figure 70. Vizualizarea unui eveniment de bifurcare a lanțului-de-blocuri: un bloc nou extinde o bifurcare, reconvergența rețelei

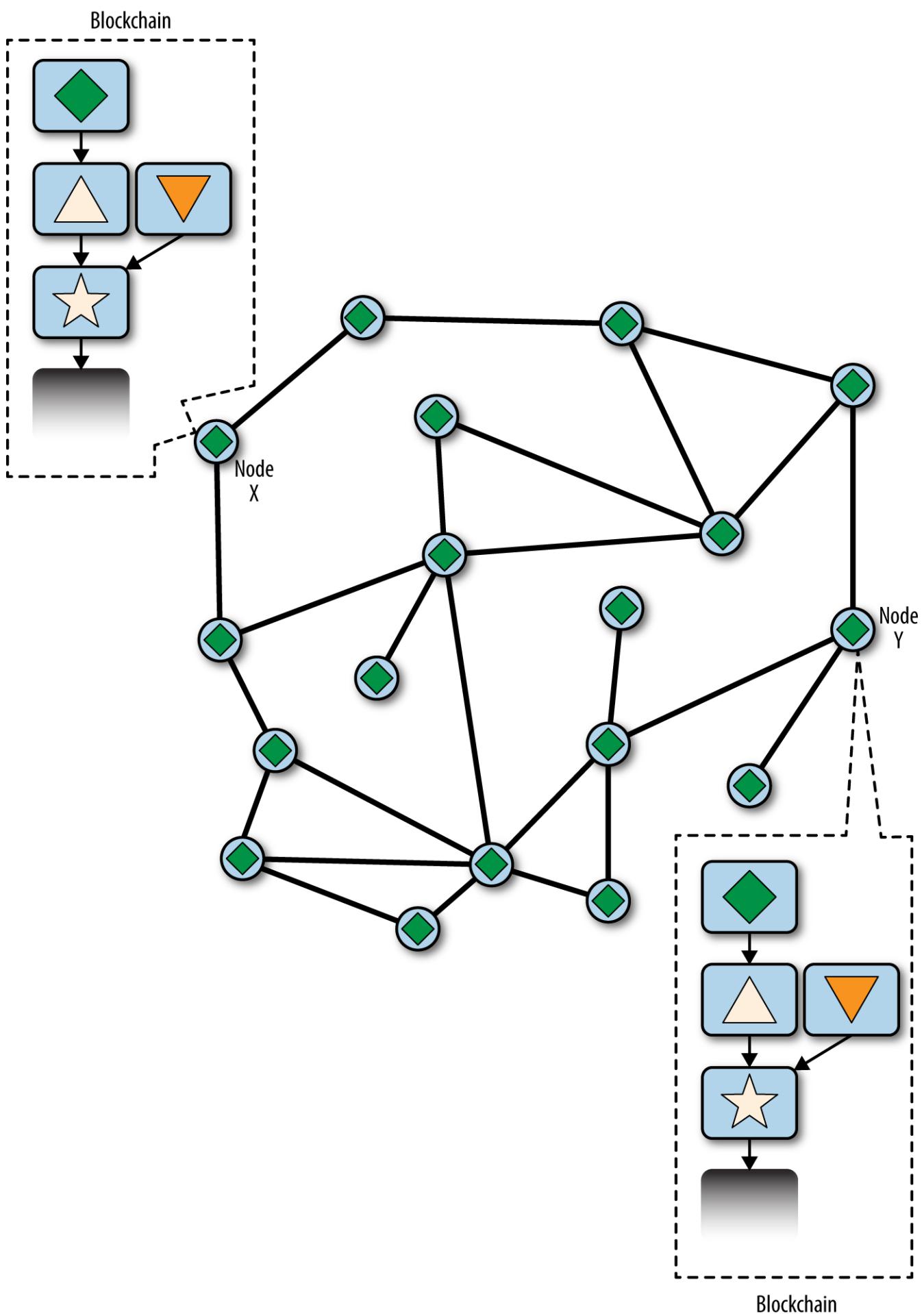


Figure 71. Vizualizarea unui eveniment de bifurcare a lanțului-de-blocuri: rețeaua reconverge pe un nou lanț mai lung

Teoretic este posibil ca o bifurcare să se extindă la două blocuri, dacă două blocuri sunt găsite aproape simultan de minerii de pe ”părțile” opuse ale unei bifurcări anterioare. Cu toate acestea, șansa să se întâpte este foarte mică. În timp ce o bifurcare cu un bloc poate apărea în fiecare zi, o bifurcare cu două blocuri apare cel mult o dată la câteva săptămâni.

Intervalul de 10 minute per bloc al bitcoin este un compromis de proiectare între timpii de confirmare rapidă (validarea tranzacțiilor) și probabilitatea unei bifurcări. Un timp mai rapid ar valida tranzacțiile mai repede, dar ar duce la bifurcări ale lanțului-de-blocuri mai frecvente, în timp ce un timp mai lent ar reduce numărul de bifurcări, dar ar face ca validarea să fie mai lentă.

## Minerit și Cursa de Rezumare

Mineritul bitcoin este o industrie extrem de competitivă. Puterea de rezumare a crescut exponential în fiecare an al existenței bitcoin. Pentru câțiva ani, creșterea a reflectat o schimbare completă a tehnologiei, cum ar fi în 2010 și 2011, când mulți mineri au trecut de la mineritul cu CPU la mineritul cu GPU. În 2013, introducerea mineritului ASIC a dus la un alt salt uriaș în puterea de minerit, prin plasarea funcției SHA256 direct pe cipurile de siliciu specializate în minerit. Primele astfel de cipuri puteau furniza mai multă putere de minerit într-o singură cutie decât întreaga rețea bitcoin în 2010.

Următoarea listă arată puterea totală de rezumare a rețelei bitcoin, în primii opt ani de funcționare:

### 2009

0.5 MH/sec–8 MH/sec (16 × creștere)

### 2010

8 MH/sec–116 GH/sec (14.500 × creștere)

### 2011

116 GH/sec–9 TH/sec (78 × creștere)

### 2012

9 TH/sec–23 TH/sec (2,56 × creștere)

### 2013

23 TH/sec–10 PH/sec (450 × creștere)

### 2014

10 PH/sec–300 PH/sec (30 × creștere)

### 2015

300 PH/sec–800 PH/sec (2,66 × creștere)

### 2016

800 PH/sec–2.5 EH/sec (3,12 × creștere)

În graficul din [Puterea totală de rezumare, terahashes pe secundă \(TH/sec\)](#), putem observa că

puterea de rezumare a rețelei bitcoin a crescut în ultimii doi ani. După cum puteți vedea, concurența dintre mineri și creșterea bitcoin a dus la o creștere exponențială a puterii de rezumare (rezumări totale pe secundă în întreaga rețea).

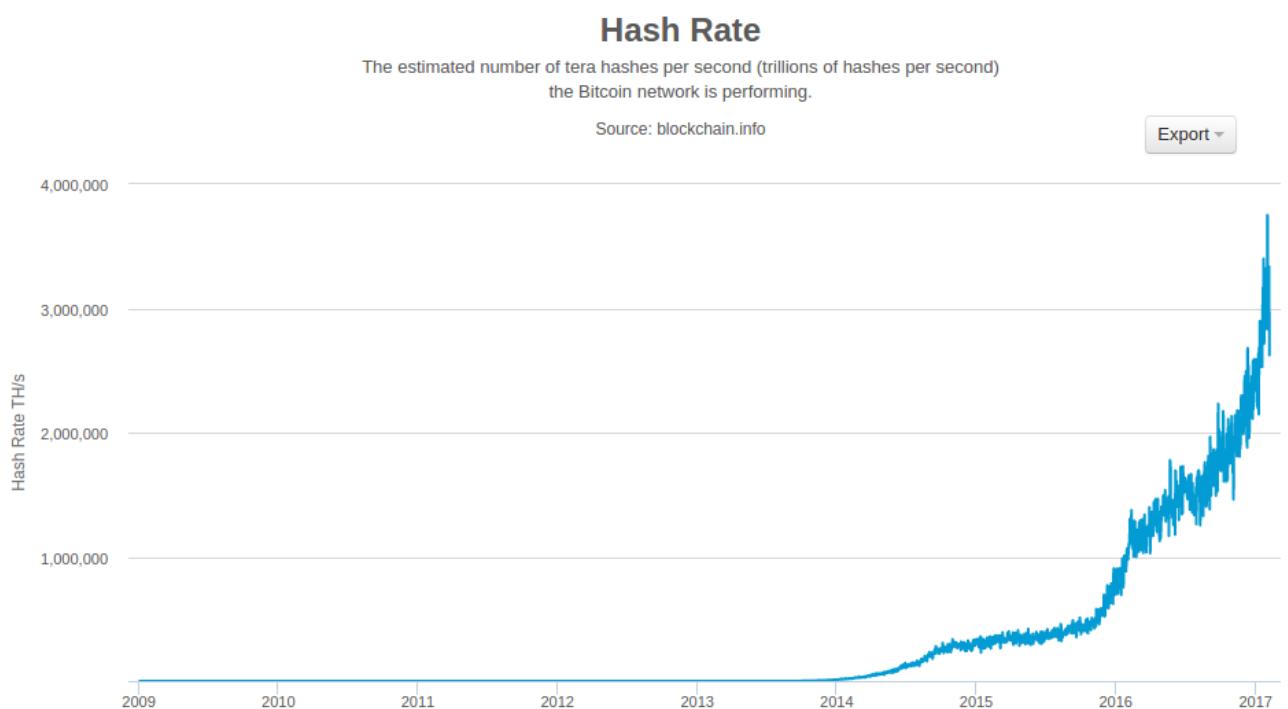


Figure 72. Puterea totală de rezumare, terahashes pe secundă (TH/sec)

Pe măsură ce cantitatea de putere de rezumare folosită pentru a mina bitcoin a explodat, dificultatea a crescut și ea. Metrica de dificultate din graficul prezentat în [Metrica dificultății pentru minerit bitcoin](#) se măsoară ca raportul între dificultatea curentă și dificultatea minimă (dificultatea primului bloc).

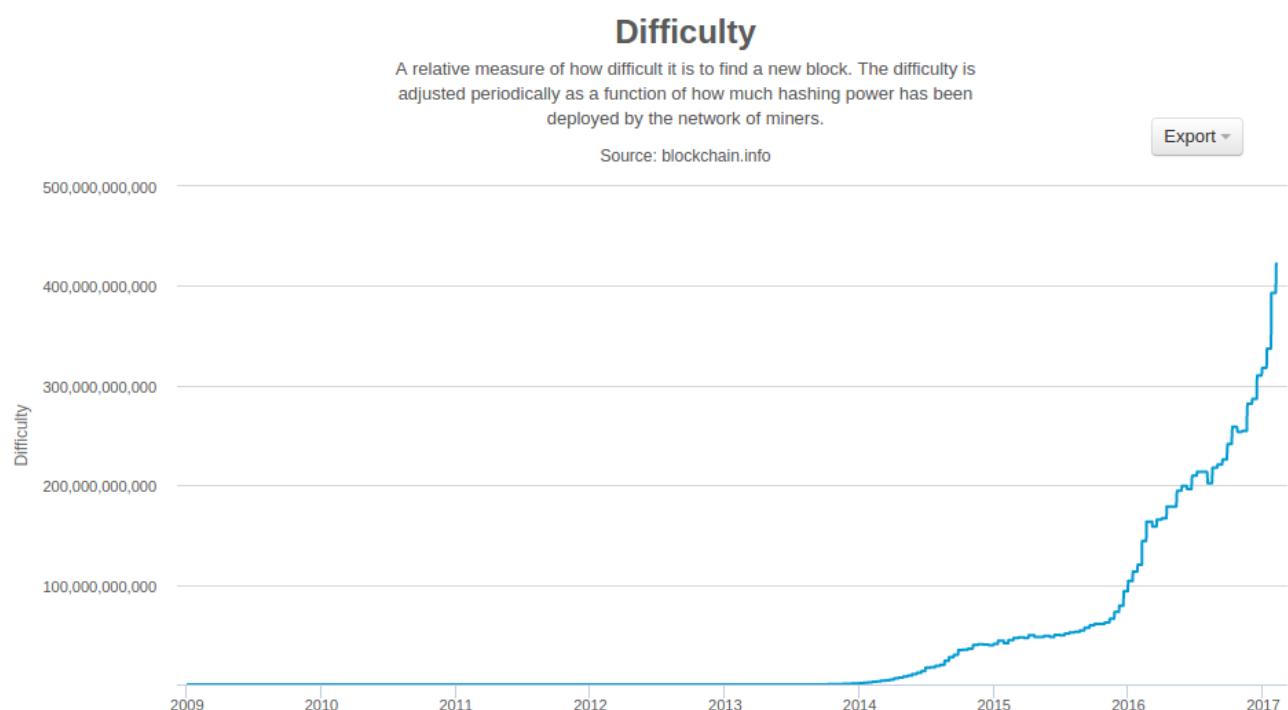


Figure 73. Metrica dificultății pentru minerit bitcoin

În ultimii doi ani, cipurile de minerit ASIC au devenit din ce în ce mai dense, apropiindu-se de

limita producției de siliciu cu o dimensiune (rezoluție) de 16 nanometri (nm). În prezent, producătorii ASIC își propun să depășească producătorii de cipuri de procesare cu scop general, proiectând cipuri cu o dimensiune de 14 nm, deoarece profitabilitatea mineritului împinge această industrie chiar mai repede decât cea a calculatoarelor obișnuite. Nu mai există salturi uriașe în minerit bitcoin, deoarece industria a ajuns în prim-planul Legii lui Moore, care stipulează că densitatea de calcul se va dubla aproximativ la fiecare 18 luni. Totuși, puterea de minerit a rețelei continuă să avanseze într-un ritm exponentional, deoarece cursa pentru cipuri cu densitate mai mare este acompaniată de o cursă pentru centre de date cu densitate mai mare, unde mii de aceste cipuri pot fi folosite. Nu mai este vorba despre cât minerit se poate face cu un cip, ci despre cât de multe cipuri pot fi îngrămadite într-o clădire, asigurând în același timp disiparea căldurii și având suficientă energie electrică.

## Soluția Extra Nonce

(“valori nonce” Din 2012, mineritul bitcoin a evoluat pentru a rezolva o limitare fundamentală în structura antetului blocului. În primele zile ale bitcoin, un miner putea găsi un bloc prin iterarea peste un nonce până când rezumatul rezultat era sub țintă. Pe măsură ce dificultatea a crescut, minerii au trecut adesea prin toate cele 4 miliarde de valori ale nonce-ului fără a găsi un bloc. Cu toate acestea, acest lucru a fost rezolvat cu ușurință prin actualizarea mărcii de timp pentru a ține cont de timpul scurs. Deoarece marca de timp face parte din antet, schimbarea ei le permite minerilor să itereze peste valorile nonce-ului din nou, cu rezultate diferite. Odată ce hardware-ul de minerit a depășit 4GH/sec, această abordare a devenit din ce în ce mai dificilă, deoarece valorile pentru nonce erau epuizate în mai puțin de o secundă. Pe măsură ce echipamentele de minerit ASIC au început să împingă și să depășească rata de rezumare de TH/sec, software-ul de minerit avea nevoie de mai mult spațiu pentru valorile nonce-ului pentru a găsi blocuri valide. Marca de timp putea fi modificată un pic, dar deplasarea ei prea departe în viitor ar determina blocul să nu fie valid. A fost necesară o nouă sursă de “schimbare” în antetul blocului. Soluția a fost utilizarea tranzacției coinbase ca sursă de valori extra pentru nonce. Deoarece script-ul coinbase poate stoca între 2 și 100 de octeți de date, minerii au început să folosească acel spațiu ca spațiu suplimentar pentru nonce, permitându-le să exploreze o gamă mult mai mare de valori ale antetului blocului pentru a găsi blocuri valide. Tranzacția coinbase este inclusă în arborele merkle, ceea ce înseamnă că orice modificare a scriptului coinbase determină modificarea rădăcinii merkle. Opt octeți de nonce în plus, plus cei 4 octeți de nonce “standard” le permit minerilor să exploreze un total de  $2^{96}$  (8 următoare de 28 zeroare) posibilități pe secundă fără a fi nevoie să modifice marca de timp. Dacă, în viitor, minerii ar putea parcurge toate aceste posibilități, atunci ar putea modifica marca de timp. Există, de asemenea, mai mult spațiu în script-ul coinbase pentru extinderea viitoare a spațiului pentru nonce.

## Bazine de Minerit

În acest mediu extrem de competitiv, minerii individuali care lucrează singuri (cunoscuți și ca mineri solo) nu au nicio șansă. Probabilitatea ca aceștia să găsească un bloc care să le compenseze energia electrică și costurile hardware sunt atât de mici încât reprezintă un joc de noroc, precum jocul la loterie. Nici cel mai rapid sistem de minerit ASIC de larg consum nu poate ține pasul cu sistemele comerciale care folosesc zeci de mii de aceste cipuri în depozite gigant, în apropierea centralelor hidroelectrice. Minerii colaborează acum pentru formarea bazinelor de minerit, care își combină puterea de rezumare și împărtășesc recompensa între mii de participanți. Prin participarea la un bazin, minerii obțin o pondere mai mică din recompensa totală, dar, de obicei,

sunt recompensați în fiecare zi, reducând incertitudinea.

Să ne uităm la un exemplu specific. Presupunem că un miner a achiziționat echipamente de minerit cu o rată combinată de rezumare de 14.000 gigahashes pe secundă (GH/s) sau 14 TH/s. În 2017, acest echipament costa aproximativ 2.500 USD. Hardware-ul consumă 1375 wați (1,3 kW) de energie electrică atunci când funcționează, 33 kW-oră pe zi, cu un cost între 1 și 2 USD pe zi, la tarife foarte mici de electricitate. În cazul dificultății actuale bitcoin, minerul va putea mina singur un bloc aproximativ o dată la 4 ani. Cum am calculat această probabilitate? Se bazează pe o rată de rezumare în rețea de 3 EH/sec (în 2017), iar rata minerului este de 14 TH/sec:

$$i. P = (14 * 10^{12} / 3 * 10^{18}) * 210240 = 0.98$$

Unde 210240 este numărul de blocuri în patru ani. Minerul are o probabilitate de 98% de a găsi un bloc pe parcursul a patru ani, pe baza ratei globale de rezumare la începutul perioadei.

Dacă minerul găsește un singur bloc în acest interval de timp, răsplata de 6,25 bitcoin, la aproximativ 1.000 de dolari pe bitcoin, va duce la o singură recompensă de 6250 de dolari, ceea ce va produce un profit net de aproximativ 750 de dolari. Cu toate acestea, șansa de a găsi un bloc într-o perioadă de 4 ani depinde de norocul minerului. S-ar putea să găsească două blocuri în 4 ani și să facă un profit foarte mare. Sau s-ar putea să nu găsească nici un bloc timp de 5 ani și să sufere o pierdere financiară mai mare. Și mai rău, dificultatea algoritmului Dovadă-de-Lucru este probabil să crească semnificativ în acea perioadă, la ritmul actual de creștere a puterii de rezumare, ceea ce înseamnă că minerul are cel mult un an pentru a ajunge pe zero chiar înainte ca hardware-ul să fie în mod efectiv învechit și să trebuiască înlocuit cu un hardware miner mai puternic. Din punct de vedere financiar, acest lucru are sens doar la costuri reduse de energie electrică (mai puțin de 1 cent per kW-oră) și doar la o scară foarte mare.

Bazinele de minerit coordonează sute sau mii de mineri, pe baza unor protocoale specializate pentru bazinele de minerit. Minerii individuali își configurează echipamentele miniere pentru a se conecta la un server de bazin, după crearea unui cont cu bazinul. Hardware-ul lor de exploatare rămâne conectat la serverul de bazin în timp ce minează, sincronizând eforturile lor cu ceilalți mineri. Astfel, minerii din bazin împărtășesc efortul de a mina un bloc și apoi împart recompensele.

Blocurile minate cu succes plătesc recompensa către o adresă bitcoin a bazinului, mai degrabă decât minerilor individuali. Serverul de bazin va efectua periodic plăți către adresele bitcoin ale minerilor, odată ce ponderea lor din recompense a atins un anumit prag. De obicei, serverul de bazin percepă o taxă procentuală din recompense pentru furnizarea serviciului de bazin de minare.

Minerii care participă într-un bazin împart munca de căutare a unei soluții la un bloc candidat, obținând "părți" pentru contribuția lor minieră. Bazinul de minerit stabilește o țintă mai mare (dificultate mai mică) pentru câștigarea unei părți, de obicei de peste 1.000 de ori mai ușor decât ținta rețelei bitcoin. Când cineva din bazin minează cu succes un bloc, recompensa este câștigată de către bazin și apoi împărțită tuturor minerilor proporțional cu numărul de părți cu care au contribuit la efort.

Bazinele sunt deschise oricărui miner, mare sau mic, profesionist sau amator. Prin urmare, un bazin va avea unii participanți cu o singură mașină de minerit, iar alții cu un garaj plin cu hardware de minerit de înaltă performanță. Unii vor extrage cu câteva zeci de kilowati de

electricitate, alții vor opera un centru de date consumând un megawatt de energie. Cum măsoară un bazin minier contribuțiile individuale, pentru a distribui în mod corect recompensele, fără posibilitatea de a însela? Răspunsul este folosirea algoritmului Dovadă-de-Lucru pentru a măsura contribuția fiecărui miner din bazin, dar stabilind o dificultate mai mică, astfel încât chiar și cei mai mici mineri din bazin câștigă o cotă suficient de frecvent pentru fi rentabilă participarea la bazin. Prin stabilirea unei dificultăți mai mici pentru a câștiga părți, grupul măsoară cantitatea de muncă depusă de fiecare miner. De fiecare dată când un miner din bazin găsește un rezumat din antetul blocului care este mai mic decât ținta bazinului, el dovedește că a depus munca de rezumare pentru a găsi rezultatul respectiv. Mai important, munca de a găsi părți contribuie, într-un mod măsurabil statistic, la efortul general de a găsi un rezumat mai mic decât ținta rețelei bitcoin. Mii de mineri care încearcă să găsească rezumate cu valoare scăzută vor găsi în cele din urmă unul suficient de scăzut pentru a satisface ținta rețelei bitcoin.

Să revenim la analogia unui joc de zaruri. Dacă jucătorii aruncă zaruri cu un obiectiv de a arunca mai puțin de patru (dificultatea generală a rețelei), un bazin ar stabili o țintă mai usoară, numărând de câte ori jucătorii din bazin au reușit să arunce mai puțin de opt. Atunci când jucătorii din bazin aruncă mai puțin de opt (ținta bazinului), câștigă părți, dar nu câștigă jocul, deoarece nu ating obiectivul de joc (mai puțin de patru). Jucătorii din bazin vor atinge mult mai des ținta mai usoară a bazinului, câștigând părți foarte regulat, chiar și atunci când nu ating ținta mai grea de a câștiga jocul. Din când în când, unul dintre jucătorii din bazin va arunca o aruncare combinată de zar mai mică de patru, iar bazinul câștigă. Apoi, câștigurile pot fi distribuite participaților la bazin pe baza părților pe care le-au obținut. Chiar dacă ținta de opt sau mai puțin nu a fost câștigătoare, a fost o modalitate corectă de a măsura aruncările de zar pentru jucători și produce ocazional o aruncare mai mică de patru.

În mod similar, un bazin de minerit va stabili o țintă (mai mare și mai usoară) pentru bazin, care va asigura că un miner individual din bazin poate găsi rezumate cu antetul blocului care sunt mai mici decât ținta de bazin, obținând părți. Din când în când, una dintre aceste încercări va produce un rezumat de antet de bloc care este mai mic decât ținta rețelei bitcoin, ceea ce îl face un bloc valid și întregul bazin câștigă.

## Bazine Gestionate

Majoritatea bazinelor de minerit sunt "gestionate", ceea ce înseamnă că există o companie sau o persoană care rulează un server de bazin. Proprietarul serverului de bazin se numește *operator de bazin*, iar acesta percepă minerilor din bazin o taxă procentuală din câștiguri.

Serverul de bazin rulează software specializat și un protocol pentru bazine de minare care coordonează activitățile minerilor din bazin. Serverul de bazin este, de asemenea, conectat la unul sau mai multe noduri-complete și are acces direct la o copie completă a bazei de date lanț-de-blocuri. Acest lucru permite serverului de bazin să valideze blocurile și tranzacțiile în numele minerilor din bazin, scutindu-i de povara de a rula un nod complet. Pentru minerii din bazin, aceasta este o considerație importantă, deoarece un nod complet necesită un computer dedicat cu cel puțin 100 până la 150 GB stocare persistentă (disc) și cel puțin 2 până la 4 GB memorie (RAM). În plus, software-ul bitcoin care rulează pe nodul complet trebuie monitorizat, întreținut și actualizat frecvent. Orice timp de oprire cauzat de lipsa de întreținere sau de lipsa resurselor va afecta profitabilitatea minerului. Pentru mulți mineri, posibilitatea de a mina fără a rula un nod complet este un alt mare beneficiu al aderării la un bazin gestionat.

Minerii din bazin se conectează la serverul de bazin utilizând un protocol minier, cum ar fi Stratum (STM) sau GetBlockTemplate (GBT). Un standard mai vechi numit GetWork (GWK) a fost în mare parte scos din uz încă de la sfârșitul anului 2012, deoarece nu acceptă cu ușurință mineritul la rate de rezumare de peste 4GH/s. Atât protocoalele STM, cât și GBT creează *șabloane* de bloc care conțin un șablon de antet de bloc candidat. Serverul de bazin construiește un bloc candidat prin agregarea tranzacțiilor, adăugând o tranzacție de tip coinbase (cu spațiu suplimentar pentru nonce), calculând rădăcina merkle și legând rezumatul blocului anterior. Antetul blocului candidat este apoi trimis fiecărui miner din bazin ca șablon. Fiecare miner din bazin apoi minează folosind șablonul de bloc, la o țintă mai mare (mai ușoară) decât ținta rețelei bitcoin și trimite rezultatele de succes înapoi serverului de bazin pentru a obține părți.

### **Bazin de Minerit de-la-egal-la-egal (P2Pool)**

Bazinele gestionate creează posibilitatea înșelării de către operatorul bazinului, care ar putea direcționa efortul bazinului către tranzacții de cheltuiere dublă sau de invalidare blocurilor (a se vedea [Atacuri de Consens](#)). Mai mult, serverele de bazin centralizate reprezintă un singur punct de eșec. Dacă serverul de bazin este dezactivat sau este încetinit de un atac de tip denial-of-service, minerii bazinului nu pot să mineze. În 2011, pentru a rezolva aceste probleme de centralizare, a fost propusă și pusă în aplicare o nouă metodă de minerit în bazine: P2Pool, un bazin minier de-la-egal-la-egal (peer-to-peer) fără operator central.

P2Pool funcționează descentralizând funcțiile serverului de bazin, implementând un sistem paralel de tip lanț-de-blocuri numit *lanț de acțiuni*. Un lanț de acțiuni este un lanț-de-blocuri care se confruntă cu o dificultate mai mică decât lanțul-de-blocuri bitcoin. Lanțul de acțiuni permite minerilor din bazin să colaboreze într-un bazin descentralizat prin minarea acțiunilor din lanțul de acțiuni, la o rată de un bloc de acțiuni la fiecare 30 de secunde. Fiecare dintre blocurile din lanțul de acțiuni înregistrează o recompensă proporțională a acțiunilor pentru minerii din bazin care contribuie la efort, transportând acțiunile înainte de la blocul de acțiuni precedent. Când unul dintre blocurile de acțiuni atinge și ținta rețelei bitcoin, acesta este propagat și inclus pe lanțul-de-blocuri bitcoin, răsplătind toți minerii din bazin care au contribuit la toate acțiunile care au precedat blocul de acțiuni câștigător. În esență, în loc de un server de bazin care să țină evidența acțiunilor și a recompenselor minerilor din bazin, lanțul de acțiuni permite tuturor minerilor din bazin să țină evidența tuturor acțiunilor folosind un mecanism de consens descentralizat, similar cu mecanismul de consens al lanțului-de-blocuri bitcoin.

Mineritul P2Pool este mai complex decât mineritul gestionat de bazin, deoarece necesită ca minerii din bazin să ruleze un calculator dedicat cu suficient spațiu pe disc, memorie și lățime de bandă pentru a susține un nod bitcoin și software-ul P2Pool. Minerii P2Pool conectează hardware-ul de minare la nodul lor local P2Pool, care simulează funcțiile unui server de bazin prin trimiterea de șabloane de blocuri la hardware-ul miner. Pe P2Pool, minerii individuali de bazin își construiesc propriile blocuri candidat, agregând tranzacții la fel ca minerii solo, dar apoi minează colaborativ pe lanțul de acțiuni. P2Pool este o abordare hibridă care are avantajul unor plăți mult mai granulare decât mineritul solo, dar fără a da prea mult control unui operator de bazin, cum ar fi cazul pentru bazinele gestionate.

Chiar dacă P2Pool reduce concentrația de putere a operatorilor de bazine de minerit, este posibil ca acesta să fie vulnerabil la atacurile de tip 51% împotriva lanțului de acțiuni în sine. O adoptie mult mai amplă a P2Pool nu rezolvă problema atacului de 51% pentru bitcoin în sine. Mai degrabă, P2Pool face bitcoin mai robust în ansamblu, ca parte a unui ecosistem minier diversificat.

## Atacuri de Consens

Mecanismul de consens al bitcoin este cel puțin teoretic, vulnerabil la atacul minerilor (sau bazinelor) care încearcă să-și folosească puterea de calcul pentru scopuri necinstituite sau distructive. După cum am văzut, mecanismul de consens depinde de faptul că majoritatea minerilor acționează cîndin din interes propriu. Cu toate acestea, dacă un miner sau un grup de mineri pot obține o pondere semnificativă din puterea de minerit, ei pot ataca mecanismul de consens, astfel încât să perturbe securitatea și disponibilitatea rețelei bitcoin.

Este important de menționat că atacurile de consens pot afecta doar consensul viitor sau, în cel mai bun caz, trecutul cel mai recent (zeci de blocuri). Registrul bitcoin devine din ce în ce mai imutabil pe măsură ce trece timpul. În timp ce în teorie, o bifurcare poate fi obținută la orice adâncime, în practică, puterea de calcul necesară pentru a forța o bifurcare foarte adâncă este imensă, făcând blocurile vechi practic imutabile. Atacurile de consens nu afectează nici securitatea cheilor private sau a algoritmului de semnare (ECDSA). Un atac de consens nu poate fura bitcoin, nu poate cheltui bitcoin fără semnaturi, redirecționa bitcoin sau schimba tranzacțiile sau înregistrările de proprietate din trecut. Atacurile de consens nu pot afecta decât cele mai recente blocuri și pot provoca perturbări de tipul denial-of-service pentru crearea de blocuri viitoare.

Un scenariu de atac împotriva mecanismului de consens se numește "atac 51%". În acest scenariu, un grup de mineri, care controlează o majoritate (51%) din puterea totală de calcul a rețelei, pun la cale atacul împotriva bitcoin. Cu capacitatea de a mina majoritatea blocurilor, minerii atacatori pot provoca "bifurcări" deliberate în lanțul-de-blocuri și pot efectua cheltuiiri duble sau să execute atacuri de tip denial-of-service împotriva unor tranzacții sau adrese specifice. Un atac bifurcare/cheltuire-dublă este acela în care atacatorul face ca blocurile confirmate anterior să fie invalidate, bifurcând sub ele și re-convergând pe un lanț alternativ. Cu o putere suficientă, un atacator poate invalida șase sau mai multe blocuri la rând, determinând invalidarea tranzacțiilor considerate imutabile (șase confirmări). Rețineți că o cheltuire dublă se poate efectua numai pe tranzacțiile proprii ale atacatorului, pentru care atacatorul poate produce o semnatură validă. Cheltuirea dublă a propriilor tranzacții este profitabilă dacă prin invalidarea unei tranzacții, atacatorul poate face un schimb ireversibil pentru o plată sau un produs fără să le plătească.

Să examinăm un exemplu practic de atac 51%. În primul capitol, am analizat o tranzacție între Alice și Bob pentru o ceașcă de cafea. Bob, proprietarul cafenelei, este dispus să accepte plata pentru cafea fără să aștepte confirmarea (minare într-un bloc), deoarece riscul unei cheltuiiri duble pentru o ceașcă de cafea este scăzut, în comparație cu comoditatea unui serviciu rapid pentru clienți. Acest lucru este similar cu practicile cafenelelor care acceptă plăți cu cardul de credit fără o semnatură pentru sume sub 25\$, deoarece riscul unei rambursări cu card de credit este scăzut, în timp ce costul întârzierii tranzacției pentru obținerea unei semnaturi este relativ mai mare. În schimb, vânzarea unui articol mai scump pentru bitcoin riscă un atac de cheltuire-dublă, în care cumpărătorul difuzează o tranzacție concurrentă care cheltuiește aceleași intrări (UTXO) și anulează plata către comerciant. Un atac de cheltuire-dublă se poate întâmpla în două moduri: fie înainte de confirmarea unei tranzacții, fie dacă atacatorul profită de o bifurcare în lanțul-de-blocuri pentru a anula mai multe blocuri. Un atac de 51% le permite atacatorilor să-și cheltuiască dublu propriile tranzacții în noul lanț, anulând astfel tranzacția corespunzătoare din vechiul lanț.

În exemplul nostru, atacatorul rău intenționat Mallory merge la Galeria lui Carol și achiziționează un frumos tablou care îl înfățișează pe Satoshi Nakamoto drept Prometeu. Carol vinde tabloul

”Marele Foc” pentru 250.000 de dolari în bitcoin către Mallory. În loc să aștepte șase sau mai multe confirmări cu privire la tranzacție, Carol împacheteaza și inmâneaza tabloul lui Mallory după o singură confirmare. Mallory lucrează cu un complice, Paul, care operează un bazin de minerit mare, iar complicele lansează un atac de 51% imediat ce tranzacția lui Mallory este inclusă într-un bloc. Paul îndrumă bazinul de minerit să remineze aceeași înălțime a blocului ca și blocul care conține tranzacția lui Mallory, înlocuind plata lui Mallory către Carol cu o tranzacție care cheltuie dublu aceeași intrare ca și plata lui Mallory. Tranzacția de cheltuire-dublă consumă același UTXO și plătește înapoi portofelul lui Mallory, în loc să îi plătească lui Carol, permitându-i, în esență, lui Mallory să păstreze bitcoinul. Paul apoi îndrumă bazinul de minerit să mineze un bloc suplimentar, astfel încât lanțul care conține tranzacția cu cheltuire dublă să fie mai lung decât lanțul inițial (provocând o bifurcare sub blocul care conține tranzacția lui Mallory). Când bifurarea lanțului-de-blocuri se rezolvă în favoarea noului lanț (mai lung), tranzacția cu cheltuire-dublă înlocuiește plata inițială către Carol. Lui Carol îi lipsește acum tabloul și, de asemenea, nu are nicio plată bitcoin. Pe toată durata acestei activități, participanții la bazinul de minerit a lui Paul ar putea să nu aibă nici o idee de încercarea de cheltuire-dublă, deoarece ei minează cu mineri automatizați și nu pot monitoriza fiecare tranzacție sau bloc.

Pentru a se proteja împotriva acestui tip de atac, un comerciant care vinde articole de valoare mare trebuie să aștepte cel puțin șase confirmări înainte de a da produsul cumpărătorului. În mod alternativ, comerciantul ar trebui să utilizeze un cont multisemnatură cu împăternicire, așteptând din nou mai multe confirmări după finanțarea contului cu împăternicire. Cu cât trec mai multe confirmări, cu atât devine mai greu să invalidezi o tranzacție cu un atac de 51%. Pentru articole cu valoare ridicată, plata cu bitcoin va fi în continuare convenabilă și eficientă chiar dacă cumpărătorul trebuie să aștepte 24 de ore pentru livrare, ceea ce ar corespunde cu aproximativ 144 confirmări.

Pe lângă un atac de cheltuire-dublă, celălalt scenariu pentru un atac de consens este să refuze serviciul anumitor participanți bitcoin (adrese bitcoin specifice). Un atacator cu majoritatea puterii de minare poate pur și simplu să ignore anumite tranzacții. Dacă sunt incluse într-un bloc minat de un alt miner, atacatorul poate în mod deliberat să bifurce și să remineze blocul, excluzând din nou tranzacțiile specifice. Acest tip de atac poate duce la o negare a serviciului împotriva unei adrese specifice sau a unui set de adrese atâtă timp cât atacatorul controlează majoritatea puterii de minerit.

În ciuda numelui său, scenariul de atac de 51% nu necesită de fapt 51% din puterea de calcul. De fapt, un astfel de atac poate fi încercat cu un procent mai mic din puterea de rezumare. Pragul de 51% este pur și simplu nivelul la care un astfel de atac este aproape garantat să reușească. Un atac de consens este în esență un remorcher de război pentru următorul bloc, iar grupul ”mai puternic” are mai multe șanse de câștig. Cu o putere mai puțin sporită, probabilitatea de reușită este redusă, deoarece alți mineri controlează generarea unor blocuri cu puterea lor de minerit ”cinstită”. O modalitate de a privi problema este că cu cât un atacator are mai multă putere de calcul, cu atât mai multe bifurcări poate crea în mod deliberat, cu atât mai multe blocuri din trecutul recent poate invalida, sau cu atât mai multe blocuri pe viitor poate controla. Grupurile de cercetare în domeniul securității au folosit modelarea statistică pentru a susține că sunt posibile diferite tipuri de atacuri de consens cu doar 30% din puterea de calcul.

Creșterea masivă a puterii totale de calcul a făcut, în mod cert, bitcoin insensibil atacurilor unui singur miner. Nu este posibil ca un miner solo să controleze mai mult decât un procent mic din puterea de minerit totală. Cu toate acestea, centralizarea controlului cauzată de bazinele de minerit

a introdus riscul atacurilor unui operator de bazin de minerit cu scopul de a obține profit. Operatorul de bazin dintr-un bazin gestionat controlează construcția blocurilor candidat și, de asemenea, controlează tranzacțiile incluse. Acest lucru oferă operatorului de bazin posibilitatea de a exclude tranzacții sau de a introduce tranzacții cu cheltuire-dublă. Dacă un astfel de abuz de putere se face într-un mod limitat și subtil, un operator de bazin ar putea obține profit dintr-un atac de consens, fără a fi observat.

Cu toate acestea, nu toți atacatorii vor fi motivați de profit. Un scenariu de atac potențial este acela în care un atacator intenționează să perturbe rețeaua bitcoin fără posibilitatea de a profita de o asemenea perturbare. Un atac rău intenționat gândit să paralizeze bitcoin ar necesita investiții enorme și planificare ascunsă, dar poate fi lansat de un atacator bine finanțat, cel mai probabil sponsorizat de stat. În mod alternativ, un atacator bine finanțat ar putea ataca consensul bitcoin, acumulând simultan hardware-ul minier, compromitând operatorii de bazine și atacând alte bazine cu atacuri denial-of-service. Toate aceste scenarii sunt teoretic posibile, dar din ce în ce mai nepractice, întrucât puterea totală de calcul a rețelei bitcoin continuă să crească exponențial.

Fără îndoială, un atac de consens serios ar deteriora încrederea în bitcoin pe termen scurt, ceea ce ar putea duce la o scădere semnificativă a prețului. Cu toate acestea, rețeaua și software-ul bitcoin sunt în continuă evoluție, astfel încât atacurile de consens ar fi întâmpinate cu măsuri imediate de către comunitatea bitcoin, ceea ce face ca bitcoin să fie mai robust.

## Modificarea Regulilor de Consens

Regulile de consens determină validitatea tranzacțiilor și a blocurilor. Aceste reguli stau la baza colaborării dintre toate nodurile bitcoin și sunt responsabile pentru convergența tuturor perspectivelor locale într-un singur lanț-de-blocuri consecvent în întreaga rețea.

Deși regulile de consens sunt invariabile pe termen scurt și trebuie să fie consecvente pe toate nodurile, acestea nu sunt invariabile pe termen lung. Pentru a permite evoluția și dezvoltarea sistemului bitcoin, regulile trebuie să se schimbe din când în când pentru a acomoda noi funcționalități, îmbunătățiri sau corecții de erori. Spre deosebire de dezvoltarea software-traditional, totuși, upgrade-urile unui sistem de consens sunt mult mai dificile și necesită coordonare între toți participanții.

### Bifurcări Hard

În [Bifurcări ale Lanțului-de-Blocuri](#) am analizat modul în care rețeaua bitcoin poate diverge pentru scurt timp, două părți ale rețelei urmând două ramuri diferite ale lanțului-de-blocuri pentru o perioadă scurtă de timp. Am văzut cum acest proces se produce în mod natural, ca parte a funcționării normale a rețelei și modul în care rețeaua reconverge pe un lanț-de-blocuri comun după ce unul sau mai multe blocuri sunt minate.

Există un alt scenariu în care rețeaua poate devia în două lanțuri: o schimbare a regulilor de consens. Acest tip de bifurcare se numește *bifurcare hard*, deoarece după bifurcare rețeaua nu mai converge într-un singur lanț. În schimb, cele două lanțuri evoluează independent. Bifurcările hard apar atunci când o parte a rețelei funcționează sub un set diferit de reguli de consens decât restul rețelei. Acest lucru poate apărea din cauza unei erori sau din cauza unei schimbări deliberate în implementarea regulilor de consens.

Bifurcările hard pot fi folosite pentru a schimba regulile de consens, dar necesită coordonare între toți participanții la sistem. Orice noduri care nu se actualizează la noile reguli de consens nu sunt în măsură să participe la mecanismul de consens și sunt forțate pe un lanț separat în momentul bifurcării. Astfel, o schimbare introdusă de o bifurcare hard poate fi considerată "incompatibilă cu versiunile precedente", în condițiile în care sistemele neactualizate nu mai pot procesa noile reguli de consens.

Să examinăm mecanismele unei bifurcări hard cu un exemplu specific.

**Un lanț-de-blocuri cu bifurcări** prezintă un lanț-de-blocuri cu două bifurcări. La înălțimea blocului 4, apare o bifurcare de un bloc. Acesta este tipul de bifurcare spontană pe care am văzut-o în [Bifurcări ale Lanțului-de-Blocuri](#). Odată cu minarea blocului 5, rețeaua reconverge pe un singur lanț și bifurcarea este rezolvată.

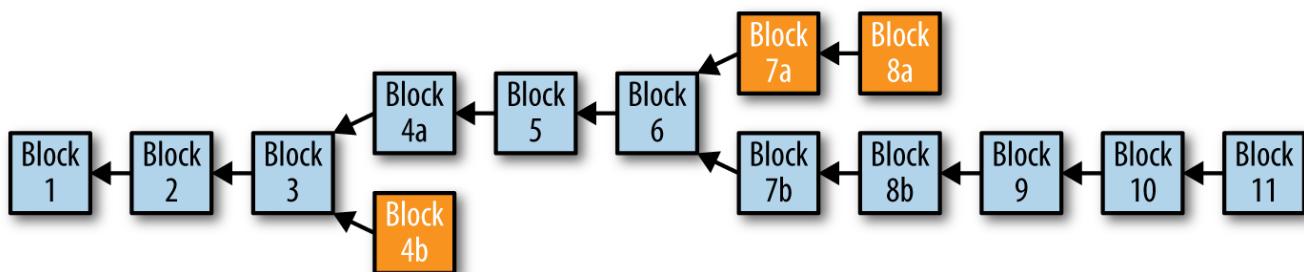


Figure 74. Un lanț-de-blocuri cu bifurcări

Mai târziu, însă, la înălțimea de bloc 6, apare o bifurcare hard. Să presupunem că o nouă implementare a clientului este lansată cu o modificare a regulilor de consens. Începând de la înălțimea blocului 7, minerii care execută această nouă implementare vor accepta un nou tip de semnătură digitală, să o numim o semnătură "Smores", care nu este bazată pe ECDSA. Imediat după, un nod care rulează noua implementare creează o tranzacție care conține o semnătură Smores și un miner cu software-ul actualizat minează blocul 7b care conține această tranzacție.

Orice nod sau miner care nu a actualizat software-ul pentru validarea semnăturilor Smores nu poate prelucra acum blocul 7b. Din perspectiva lor, atât tranzacția care conține o semnătură Smores, cât și blocul 7b care conține acea tranzacție nu sunt valide, deoarece le evaluatează pe baza vechilor reguli de consens. Aceste noduri vor respinge tranzacția împreună cu blocul și nu le vor propaga. Orice miner care utilizează regulile vechi nu vor accepta blocul 7b și vor continua să mineze un bloc candidat al cărui părinte este blocul 6. De fapt, minerii care folosesc regulile vechi pot să nu primească nici măcar blocul 7b dacă toate nodurile cu care sunt conectate respectă de asemenea vechile reguli și, prin urmare, nu propagă blocul. În cele din urmă, vor reuși să mineze blocul 7a, care este valid în conformitate cu vechile reguli și nu conține tranzacții cu semnături Smores.

Cele două lanțuri continuă să diverge din acest punct. Minerii din lanțul "b" vor continua să accepte și să mineze tranzacțiile care conțin semnături Smores, în timp ce minerii din lanțul "a" vor continua să ignore aceste tranzacții. Chiar dacă blocul 8b nu conține nicio tranzacție semnată Smores, minerii din lanțul "a" nu îl pot prelucra. Pentru ei pare a fi un bloc orfan, deoarece părintele său "7b" nu este recunoscut ca un bloc valid.

## Bifurcări Hard: Software, Rețea, Minerit și Lanț

Pentru programatori, termenul "bifurcare" are o altă semnificație, adăugând confuzie la termenul "bifurcare hard". În software-ul open source, o bifurcare apare atunci când un grup de dezvoltatori aleg să urmeze un roadmap software diferit și să înceapă o implementare concurrentă a unui proiect open source. Am discutat deja despre două circumstanțe care vor duce la o bifurcare hard: o eroare a regulilor de consens și o modificare deliberată a regulilor de consens. În cazul unei modificări deliberate a regulilor de consens, o bifurcare software precede bifurcarea hard. Cu toate acestea, pentru ca acest tip de bifurcare să aibă loc, trebuie dezvoltată, adoptată și lansată o nouă implementare software a regulilor de consens.

Exemple de bifurcări software care au încercat să schimbe regulile de consens includ Bitcoin XT, Bitcoin Classic și cel mai recent Bitcoin Unlimited. Cu toate acestea, niciuna dintre aceste bifurcări software nu a rezultat într-o bifurcare hard. Deși o bifurcare software este o precondiție necesară, nu este în sine suficientă pentru apariția unei bifurcări hard. Pentru ca o bifurcare să aibă loc, implementarea concurrentă trebuie adoptată și noile reguli activate, de către mineri, portofele și noduri intermediare. În schimb, există numeroase implementări alternative ale Bitcoin Core, și chiar bifurcări software, care nu schimbă regulile de consens și corectează o eroare, pot coexista în rețea și pot interopera fără a provoca o bifurcare hard.

Regulile de consens pot dифeri în mod evident și explicit, în validarea tranzacțiilor sau a blocurilor. Regulile pot dифeri în moduri mai subtile, în implementarea regulilor de consens, deoarece acestea se aplică scripturilor bitcoin sau primitivelor criptografice, cum ar fi semnăturile digitale. În cele din urmă, regulile de consens pot dифeri în moduri neanticipate din cauza restricțiilor implicate de consens impuse de limitările sistemului sau de către detaliile de implementare. Un exemplu al situației din urmă a fost observat la bifurcarea hard neanticipată în timpul actualizării Bitcoin Core 0.7 la 0.8, care a fost cauzată de o limitare a implementării Berkeley DB folosită pentru stocarea blocurilor.

În mod conceptual, ne putem gândi la o bifurcare hard ca fiind dezvoltată în patru etape: o bifurcare software, o bifurcare de rețea, o bifurcare de minerit și o bifurcare a lanțului.

Procesul începe atunci când programatorii creează o implementare alternativă a clientului, cu reguli de consens modificate.

Atunci când această implementare a bifurcării este lansată în rețea, un anumit procent de mineri, utilizatori de portofel și noduri intermediare pot adopta și executa această implementare. Apariția unei bifurcări va depinde dacă noile reguli de consens se aplică blocurilor, tranzacțiilor sau a unui alt aspect al sistemului. Dacă noile reguli de consens se referă la tranzacții, atunci un portofel care creează o tranzacție în conformitate cu noile reguli poate precipita o bifurcare de rețea, urmată de o bifurcare hard când tranzacția este minată într-un bloc. Dacă noile reguli se referă la blocuri, atunci procesul de bifurcare hard va începe atunci când un bloc este minat în conformitate cu noile reguli.

În primul rând, rețeaua va bifurca. Nodurile bazate pe implementarea inițială a regulilor de consens vor respinge orice tranzacții și blocuri create după noile reguli. Mai mult, nodurile care urmează regulile de consens originale vor interzice și se vor deconecta temporar de la nodurile care le transmit aceste tranzacții și blocuri invalide. Drept urmare, rețeaua se va împărți în două: nodurile vechi vor rămâne conectate numai la nodurile vechi și nodurile noi vor fi conectate numai

la nodurile noi. O singură tranzacție sau bloc bazat pe noile reguli se va propaga prin rețea și va avea ca rezultat partaționarea în două rețele.

Odată ce un miner, care utilizează noile reguli, va mina un bloc, puterea de minerit și lanțul vor bifurca de asemenea. Noii mineri vor mina deasupra noului bloc, în timp ce minerii vechi vor mina un lanț separat pe baza vechilor reguli. Rețeaua partaționată va face astfel încât minerii care operează pe reguli de consens diferite să nu primească blocuri unii de la alții, deoarece sunt conectați la două rețele separate.

## Mineri Divergenți și Dificultate

Pe măsură ce minerii diverg în minarea a două lanțuri diferite, puterea de calcul este împărțită între lanțuri. Puterea de minerit poate fi împărțită în orice proporție între cele două lanțuri. Noile reguli pot fi urmate doar de o minoritate sau de marea majoritate a puterii de minerit.

Să presupunem, de exemplu, o scindare de 80% -20%, majoritatea puterii de minerit folosind noile reguli de consens. Să presupunem, de asemenea, că bifurcarea apare imediat după o perioadă de ajustarea a întei de dificultate.

Cele două lanțuri ar moșteni fiecare dificultatea din perioada de ajustare. Noile reguli de consens ar avea 80% din puterea de minerit disponibilă anterior. Din perspectiva acestui lanț, puterea de minerit a scăzut brusc cu 20% față de perioada anterioară. Blocurile vor fi găsite în medie la fiecare 12,5 minute, reprezentând scăderea cu 20% a puterii de minerit disponibile pentru extinderea acestui lanț. Această rată a emiterii blocurilor va continua (exceptând orice schimbări în puterea de rezumare) până la extragerea a 2016 blocuri, ceea ce va dura aproximativ 25.200 de minute (12.5 minute pe bloc) sau 17.5 zile. După 17,5 zile, va avea loc o ajustare și dificultatea se va regla (redusă cu 20%) pentru a produce din nou blocuri la 10 minute.

Lanțul minoritar, minând în conformitate cu vechile reguli cu doar 20% din puterea de rezumare, se va confrunta cu o sarcină mult mai dificilă. Pe acest lanț, acum blocurile vor fi minate în medie la fiecare 50 de minute. Dificultatea nu va fi ajustată pentru următoarele 2016 blocuri, care vor necesita 100.800 de minute sau aproximativ 10 săptămâni pentru a fi minate. Presupunând o capacitate fixă pe bloc, aceasta va rezulta, de asemenea, la reducerea capacitatei de tranzacție cu un factor de 5, deoarece există mai puține blocuri pe oră disponibile pentru înregistrarea tranzacțiilor.

## Bifurcări Hard Contencioase

Acum sunt începuturile dezvoltării de software de consens. La fel cum dezvoltarea open source a schimbat atât metodele și produsele software-ului, cât și crearea de noi metodologii, noi instrumente și noi comunități în urma sa, dezvoltarea software-ului consensual reprezintă, de asemenea, o nouă frontieră în informatică. Din dezbatările, experimentele și încercările dezvoltării bitcoin, vom vedea că apar noi instrumente de dezvoltare, practici, metodologii și comunități.

Bifurcările hard sunt văzute ca riscante, deoarece forțează o minoritate să facă upgrade sau să rămână pe un lanț minoritar. Mulți consideră că riscul de a împărți întregul sistem în două sisteme concurente este un risc inacceptabil. Drept urmare, mulți programatori sunt reticenți să utilizeze mecanismul de bifurcare hard pentru a implementa actualizări la regulile de consens, cu excepția cazului în care există o susținere aproape unanimă din partea întregii rețele. Orice

propuneri de bifurcare hard care nu au un suport aproape unanim sunt considerate prea "controversate" pentru fi încercate fără a risca o partajare a sistemului.

Problema bifurcărilor hard este extrem de controversată în comunitatea de dezvoltare bitcoin, mai ales când se referă la orice modificări propuse la regulile de consens care controlează limita maximă a blocului. Unii dezvoltatori se opun oricărei forme de bifurcare hard, considerând-o prea riscantă. Alții consideră mecanismul bifurcării hard ca un instrument esențial pentru modernizarea regulilor de consens într-un mod care evită "datorile tehnice" și oferă o despărțire curată de trecut. În cele din urmă, unii dezvoltatori văd bifurcările hard ca un mecanism care ar trebui utilizat rar, cu multă planificare în avans și doar sub un consens aproape unanim.

Deja am văzut apariția de noi metodologii care să abordeze riscurile create de bifurcări hard. În secțiunea următoare, vom analiza bifurcările soft și metodele BIP-34 și BIP-9 pentru semnalizarea și activarea modificărilor consensului.

## Bifurcări Soft

Nu toate modificările regulilor de consens provoacă o bifurcare hard. Doar schimbările de consens care sunt incompatibile cu versiunea veche determină o bifurcare. Modificarea poate avea loc fără o bifurcare, dacă este implementată în aşa fel încât un client nemodificat încă vede tranzacția sau blocul ca fiind valide în conformitate cu regulile anterioare.,

Termenul *bifurcare soft* a fost introdus pentru a distinge această metodă de actualizare de o "bifurcare hard". În practică, o bifurcare soft nu este deloc o bifurcare. O bifurcare soft este o modificare compatibilă cu versiunile anterioare a regulilor de consens, care permite clienților neactualizați să continue să funcționeze în consens cu noile reguli.

Un aspect al bifurcărilor soft care nu este imediat evident este faptul că actualizările bifurcării soft pot fi folosite doar pentru a constrângi regulile de consens, nu pentru a le relaxa. Pentru a fi compatibile cu versiunile anterioare, tranzacțiile și blocurile create în conformitate cu noile reguli trebuie să fie valide și în conformitate cu vechile reguli, dar nu invers. Noile reguli pot limita numai ceea ce este valid; altfel, vor declanșa o bifurcare hard atunci când sunt respinse în conformitate cu vechile reguli.

Bifurcările soft pot fi implementate în mai multe moduri - termenul nu specifică o anumită metodă, ci mai degrabă un set de metode care au un lucru în comun: nu necesită ca toate nodurile să se actualizeze sau să forțeze în afara consensului nodurile neactualizate.

## Bifurcări soft care redefinesc operatorii NOP

O serie de bifurcări soft au fost implementate în bitcoin, pe baza reinterpretării operatorilor NOP. Bitcoin Script avea zece operatori rezervați pentru utilizări viitoare, NOP1 până la NOP10. Conform regulilor de consens, prezența acestor operatori într-un script este interpretată ca un operator nul, ceea ce înseamnă că nu au niciun efect. Execuția continuă după operatorul NOP, ca și cum nu ar fi fost acolo.

Prin urmare, o bifurcare soft poate modifica semantica unui operator NOP pentru a-i da un sens nou. De exemplu, BIP-65 ([CHECKLOCKTIMEVERIFY](#)) a reinterpretat operatorul NOP2. Clienții care implementează BIP-65 interpretează NOP2 ca [OP\\_CHECKLOCKTIMEVERIFY](#) și impun o regulă absolută de consens de timp de blocare pentru UTXO-uri care conțin acest operator în scripturile de blocare.

Această modificare este o bifurcare soft, deoarece o tranzacție valabilă în conformitate cu BIP-65 este valabilă și pentru orice client care nu implementează (ignorând) BIP-65. Pentru clienții vechi, scriptul conține un cod NOP, care este ignorat.

### Alte modalități de actualizare folosind bifurcări soft

Reinterpretarea operatorilor NOP a fost și planificată și un mecanism evident pentru actualizarea consensului. Recent, însă, a fost introdus un alt mecanism de bifurcare soft, care nu se bazează pe operatorii NOP pentru un tip foarte specific de schimbare a consensului. Acest lucru este examinat mai în detaliu în [Martor Segregat](#). Segwit este o modificare arhitecturală în structura unei tranzacții, care mută scriptul de deblocare (martor) din interiorul tranzacției într-o structură externă de date (segregarea acesteia). Segwit a fost inițial conceput ca o actualizare hard fork, deoarece modifica o structură fundamentală (tranzacția). În noiembrie 2015, un programator care lucrează la Bitcoin Core a propus un mecanism prin care segwit ar putea fi introdus ca o bifurcare soft. Mecanismul folosit este o modificare a scriptului de blocare a UTXO-ului creat în conformitate cu regulile segwit, astfel încât clienții nemodificați să văd scriptul de blocare drept răscumpărabil cu orice script de deblocare. Drept urmare, segwit poate fi introdus fără a fi nevoie ca fiecare nod să fie actualizat sau separat de lanț: o bifurcare soft.

Este probabil să existe și alte mecanisme, încă de descoperit, prin care actualizările pot fi realizate ca bifurcare soft într-un mod compatibil cu versiunile precedente.

## Critici ale Bifurcărilor Soft

Bifurcările soft bazate pe operatorii NOP sunt relativ necontroverse. Operatorii NOP au fost plasăți în Bitcoin Script cu scopul explicit de a permite actualizări care nu perturbă.

Cu toate acestea, mulți programatori sunt îngrijorați de faptul că alte metode de actualizare folosind bifurcări soft fac compromisuri inacceptabile. Criticile obișnuite cu privire la modificările bifurcărilor soft includ:

### Datoria tehnică

Deoarece bifurcările soft sunt mai complexe din punct de vedere tehnic decât o actualizare folosind o bifurcare hard, acestea introduc *datorii tehnice*, termen care se referă la creșterea costurilor viitoare de întreținere a codului din cauza compromisurilor de proiectare făcute în trecut. La rândul său, complexitatea codului crește probabilitatea de erori și vulnerabilități de securitate.

### Relaxarea validării

Clienții nemodificați să văd tranzacțiile ca fiind valide, fără a evalua regulile de consens modificate. De fapt, clienții nemodificați nu validează folosind gama completă de reguli de consens, deoarece sunt orbi la noile reguli. Aceasta se aplică actualizărilor bazate pe NOP, precum și altor actualizări de bifurcare soft.

### Actualizări ireversibile

Deoarece bifurcările soft creează tranzacții cu străngeri suplimentare de consens, ele devin în practică actualizări ireversibile. Dacă o actualizare a bifurcării soft ar fi inversată după activare, orice tranzacție creată în conformitate cu noile reguli ar putea duce la pierderea fondurilor în conformitate cu vechile reguli. De exemplu, dacă o tranzacție CLTV este evaluată

conform vechilor reguli, nu există nicio restricție de timp și ea poate fi cheltuită în orice moment. Prin urmare, criticii susțin că o bifurcare soft eşuată care a trebuit să fie inversată din cauza unei erori ar duce cu siguranță la pierderea de fonduri.

## Semnalizare Bifurcare Soft folosind Versiunea Blocului

Deoarece bifurcările soft permit clienților nemodificați să continue să funcționeze în cadrul consensului, mecanismul pentru "activarea" unei bifurcări este semnalizarea disponibilității de către mineri: majoritatea minerilor trebuie să fie de acord că sunt pregătiți și dispuși să aplique noile reguli de consens. Pentru a-și coordona acțiunile, există un mecanism de semnalizare care le permite să își arate sprijinul pentru o schimbare a regulilor de consens. Acest mecanism a fost introdus odată cu activarea BIP-34 în martie 2013 și înlocuit cu activarea BIP-9 în iulie 2016.

### Semnalizare și Activare BIP-34

Prima implementare, în BIP-34, a folosit câmpul de versiune a blocului pentru a permite minerilor să semnalizeze disponibilitatea pentru a schimba o regulă specifică de consens. Înainte de BIP-34, versiunea blocului a fost setată la "1" prin *convenție* nu impusă de *consens*.

BIP-34 a definit o modificare a regulilor de consens care a impus câmpul coinbase (de intrare) al tranzacției coinbase pentru a conține înălțimea blocului. Înainte de BIP-34, coinbase putea conține orice date arbitrară pe care minerii au ales să le includă. După activarea BIP-34, blocurile valide au trebuit să conțină o înălțime specifică a blocului la începutul coinbase și să fie identificate cu un număr de versiune mai mare sau egal cu "2."

Pentru a semnaliza schimbarea și activarea BIP-34, minerii au setat versiunea de bloc pe "2", în loc de "1." Acest lucru nu a făcut ca blocurile cu versiunea "1" să devină imediat invalide. Odată activat, blocurile cu versiunea "1" vor deveni invalide și toate blocurile cu versiunea "2" ar trebui să conțină înălțimea blocului în coinbase pentru a fi valabile.

BIP-34 a definit un mecanism de activare în două etape, bazat pe o fereastră de 1000 de blocuri. Un miner va semnala disponibilitatea sa individuală pentru BIP-34 prin construirea de blocuri cu "2" ca număr de versiune. În mod strict, aceste blocuri nu trebuiau încă să respecte noua regulă de consens privind includerea înălțimii blocului în tranzacția coinbase, deoarece regula de consens nu a fost încă activată. Regulile de consens sunt activate în două etape:

- Dacă 75% (750 din ultimele 1000 de blocuri) sunt marcate cu versiunea "2", atunci blocurile de versiune "2" trebuie să conțină înălțimea blocului în tranzacția coinbase sau sunt respinse ca invalide. Blocurile de versiune "1" sunt încă acceptate de rețea și nu trebuie să conțină înălțimea blocului. Vechile și noile reguli de consens coexistă în această perioadă.
- Când 95% (950 din cele mai recente 1000 de blocuri) au versiunea "2", blocurile cu "versiunea 1" nu mai sunt considerate valide. Blocurile cu versiunea "2" sunt valide numai dacă conțin înălțimea blocului în coinbase (conform pragului anterior). După aceea, toate blocurile trebuie să respecte noile reguli de consens, iar toate blocurile valide trebuie să conțină înălțimea blocului în tranzacția coinbase.

După semnalizarea și activarea cu succes în conformitate cu regulile BIP-34, acest mecanism a fost

utilizat încă de două ori pentru a activa bifurcări soft:

- **BIP-66** Codificarea DER strictă a semnăturilor a fost activată prin semnalizarea de tip BIP-34 cu o versiune de bloc ”3”, invalidând versiunea ”2” a blocurilor.
- **BIP-65 CHECKLOCKTIMEVERIFY** a fost activat prin semnalizarea de tip BIP-34 cu versiunea de bloc ”4” invalidând versiunea ”3” a blocurilor.

După activarea BIP-65, mecanismul de semnalizare și activare BIP-34 a fost retras și înlocuit cu mecanismul de semnalizare BIP-9 descris în continuare.

Standardul este definit în [BIP-34 \(Block v2, Height in Coinbase\)](#).

## Semnalizare și Activare BIP-9

Mecanismul folosit de BIP-34, BIP-66 și BIP-65 a avut succes în activarea a trei bifurcări soft. Cu toate acestea, a fost înlocuit pentru că avea mai multe limitări:

- Folosind valori întregi pentru versiunea de bloc, doar o bifurcare soft ar putea fi activată la un moment dat, astfel încât a fost necesară coordonarea între propunerile de bifurcare soft și acordul asupra priorității și ordinii acestora.
- Mai mult, deoarece versiunea de bloc a fost incrementată, mecanismul nu a oferit o modalitate simplă de a respinge o modificare și apoi de a propune o altă variantă. Dacă clienții vechi încă rulau, ei puteau greși semnalizarea pentru o nouă modificare ca semnalizare pentru modificarea respinsă anterior.
- Fiecare nouă modificare a redus irevocabil versiunile de bloc disponibile pentru modificările viitoare.

BIP-9 a fost propus pentru a depăși aceste provocări și a îmbunătăți rata și ușurința implementării viitoarelor modificări.

BIP-9 interpretează versiunea blocului ca un câmp de biți în loc de un număr întreg. Deoarece versiunea blocului a fost utilizată inițial ca un număr întreg, versiunile 1 până la 4, acum rămân disponibili doar 29 de biți pentru a fi utilizati pentru câmpul de biți. Acest lucru lasă 29 de biți care pot fi folosiți pentru a semnaliza în mod independent și simultan disponibilitatea pentru 29 de propunerii diferite.

De asemenea, BIP-9 stabilește un timp maxim pentru semnalizare și activare. În acest fel, minerii nu trebuie să semnalizeze pentru totdeauna. Dacă o propunere nu este activată în perioada **TIMEOUT** (definită în propunere), propunerea este considerată respinsă. Propunerea poate fi retrimisă pentru semnalizare cu un bit diferit, reînnoind perioada de activare.

Mai mult, după ce **TIMEOUT** a trecut și o funcționalitate a fost activată sau respinsă, bitul de semnalizare poate fi reutilizat pentru o altă funcționalitate fără confuzie. Prin urmare, până la 29 de modificări pot fi semnalizate în paralel și după **TIMEOUT** biții pot fi ”reciclați” pentru a propune noi modificări.

## NOTE

În timp ce biții de semnalizare pot fi reutilizați sau reciclați, atâtă timp cât perioada de votare nu se suprapune, autorii BIP-9 recomandă ca biții să fie reutilizați numai atunci când este necesar; un comportament neașteptat ar putea apărea din cauza erorilor din software-ul mai vechi. Pe scurt, nu ar trebui să ne așteptăm să vedem reutilizarea până când nu s-au folosit toți cei 29 de biți o singură dată.

Modificările propuse sunt identificate de o structură de date care conține următoarele câmpuri:

### nume

O scurtă descriere utilizată pentru a distinge propunerile. Cel mai adesea BIP-ul care descrie propunerea, ca "bipN", unde N este numărul BIP.

### bit

de la 0 la 28, bitul din versiunea de bloc pe care minerii îl folosesc pentru a semnaliza aprobarea pentru această propunere.

### starttime

Timpul (bazat pe Timpul Mediat Trecut) la care începe semnalizarea după care valoarea bitului este interpretată ca semnalizarea disponibilității pentru propunere.

### endtime

Timpul (bazat pe Timpul Median Trecut) după care modificarea este considerată respinsă dacă nu a atins pragul de activare.

Spre deosebire de BIP-34, BIP-9 contorizează semnalizarea de activare în intervale întregi pe baza perioadei de ajustare a ținței de dificultate de 2016 blocuri. Pentru fiecare perioadă de reajustare, dacă suma blocurilor care semnalizează pentru o propunere depășește 95% (1916 din 2016), propunerea va fi activată după o încă o perioadă de reajustare.

BIP-9 oferă o diagramă de stare a propunerii în care ilustrează diferitele etape și tranziții pentru o propunere, așa cum se arată în [Diagrama BIP-9 de tranziție a stării](#).

Propunerile încep în starea **DEFINED** (definit), odată ce parametrii lor sunt cunoscuți (definiți) în software-ul bitcoin. Pentru blocurile cu Timp Median Trecut după data de pornire, propunerea tranzitează la starea **STARTED**. Dacă pragul de vot este depășit într-o perioadă de reajustare și nu a fost depășită limita de timp, starea propunerii trece la **LOCKED\_IN**. O perioadă de reajustare mai târziu, propunerea devine **ACTIVE**. Propunerile rămân în starea **ACTIVE** pentru totdeauna odată ce ajung la acea stare. Dacă termenul expiră înainte de atingerea pragului de vot, starea propunerii se schimbă în **FAILED**, indicând o propunere respinsă. Propunerile **FAILED** rămân permanent în acea stare.

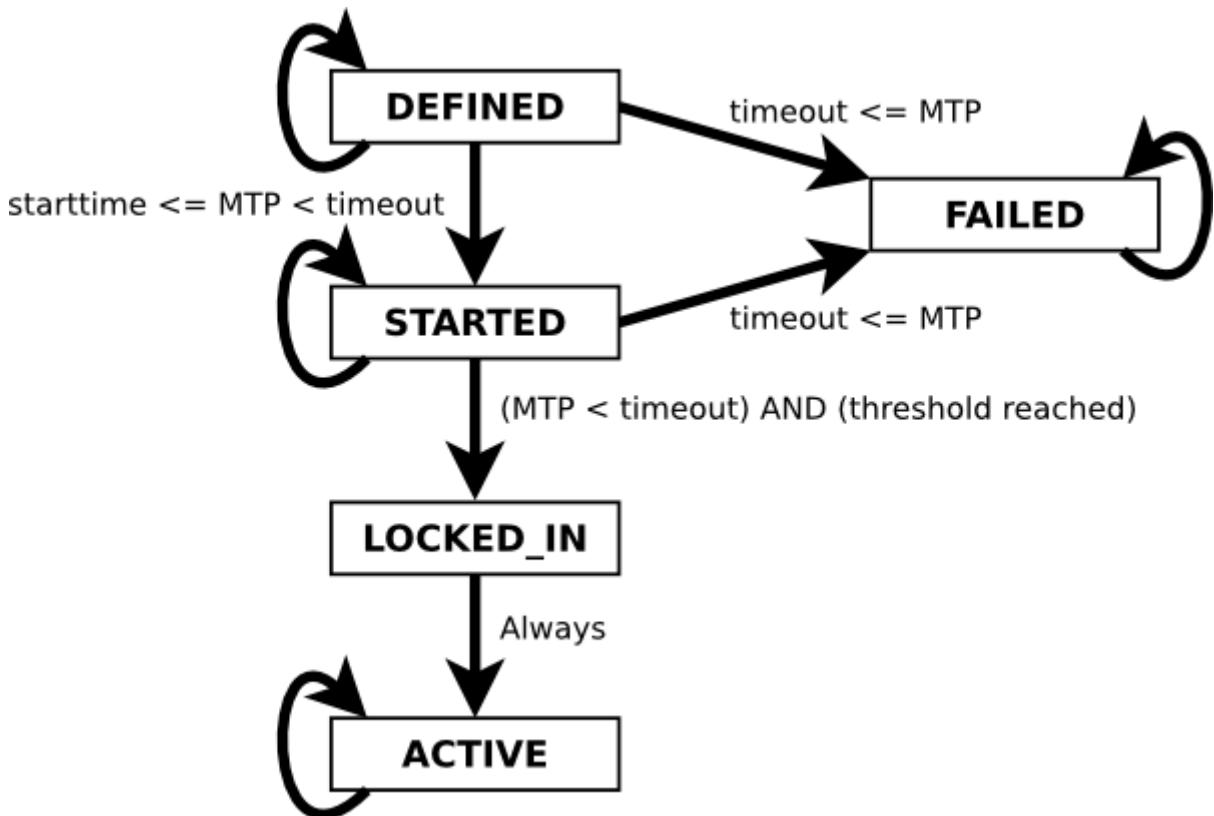


Figure 75. Diagrama BIP-9 de tranziție a stării

BIP-9 a fost implementat pentru prima oară pentru activarea [CHECKSEQUENCEVERIFY](#) și a BIP-urilor asociate (68, 112, 113). Propunerea denumită "csv" a fost activată cu succes în iulie 2016.

Standardul este definit în [BIP-9 \(Biți de versiune cu timp și întârziere\)](#).

## Dezvoltarea de Software de Consens

Software-ul de consens continuă să evolueze și există multe discuții cu privire la diferitele mecanisme de schimbare a regulilor de consens. Prin natura sa, bitcoin stabilește un standard foarte ridicat pentru coordonare și consens pentru schimbări. Ca sistem descentralizat, nu are nicio "autoritate" care să-și poată impune voința participanților rețelei. Puterea este dispersată între mai multe circumscriptii, cum ar fi minerii, programatorii bitcoin core, dezvoltatorii de portofele, bursele, comercianții și utilizatorii finali. Decizii nu pot fi luate unilateral de niciuna dintre aceste circumscriptii. De exemplu, în timp ce minerii pot modifica teoretic regulile cu majoritate simplă (51%), acestea sunt constrâns de acordul celorlalte circumscriptii. Dacă acționează unilateral, restul participanților pot refuza pur și simplu să le urmeze, păstrând activitatea economică pe un lanț minoritar. Fără activitate economică (tranzacții, comercianți, portofele, schimburi), minerii vor mina o monedă fără valoare cu blocuri goale. Această dispersie a puterii înseamnă că toți participanții trebuie să se coordoneze sau nu se pot face modificări. Status quo este starea stabilă a acestui sistem cu doar câteva modificări posibile dacă există o majoritate foarte mare de consens. Pragul de 95% pentru bifurcările soft reflectă această realitate.

Este important să recunoaștem că nu există o soluție perfectă pentru dezvoltarea consensului. Atât bifurcările hard, cât și bifurcările soft implică compromisuri. Pentru unele tipuri de schimbări, bifurcările soft pot fi o alegere mai bună; pentru altele, bifurcările hard pot fi o alegere mai bună. Nu există o alegere perfectă; ambele prezintă riscuri. Singura caracteristică constantă a dezvoltării

software-ului de consens este că schimbarea este dificilă și că consensul obligă la compromisuri.

Unii văd acest lucru ca o slăbiciune a sistemelor de consens. În timp, este posibil să ajungeți să îl vedeți aşa cum fac eu, ca cea mai mare forță a sistemului.

## Securitatea Bitcoin

Securizarea bitcoin este o provocare, deoarece bitcoin nu este o referință abstractă la valoare, precum un sold într-un cont bancar. Bitcoin seamănă foarte mult cu numerarul digital sau cu aurul. Probabil ați auzit expresia: "Posesia este nouă zecimi din lege". Ei bine, în bitcoin, posesia este zece zecimi din lege. Posesia cheilor pentru deblocarea bitcoin este echivalentă cu a deține numerar sau o bucată de metal prețios. Puteți să-l pierdeți, să-l rătăciți, să vi-l fure sau să dați accidental o sumă greșită cuiva. În fiecare dintre aceste cazuri, utilizatorii nu au recurs, la fel ca și cum ar fi scăpat bani pe un trotuar public.

Cu toate acestea, bitcoin are capacitatea pe care nu le au numerarul, aurul și conturile bancare. Un portofel bitcoin, care conține cheile dumneavoastră, poate fi salvat ca orice fișier obișnuit. Poate fi păstrat în mai multe copii, chiar imprimat pe hârtie pentru o copie de rezervă. Nu puteți "face copie de rezervă" de numerar, aur sau conturi bancare. Bitcoin este suficient de diferit de orice a venit înainte că trebuie să ne gândim și la securitatea bitcoin într-un mod inedit.

## Principii de Securitate

Principiul principal al bitcoin este descentralizarea și are implicații importante pentru securitate. Un model centralizat, cum ar fi o bancă tradițională sau o rețea de plată, depinde de controlul accesului și de verificare pentru a-i ține pe actori rău facători în afara sistemului. Prin comparație, un sistem descentralizat precum bitcoin împinge responsabilitatea și controlul către utilizatori. Deoarece securitatea rețelei se bazează pe Dovada-de-Lucru, nu pe controlul accesului, rețeaua poate fi deschisă și nu este necesară o criptare pentru traficul bitcoin.

Într-o rețea tradițională de plată, cum ar fi un sistem de carduri de credit, plata este deschisă deoarece conține datele private al utilizatorului (numărul cardului de credit). După plata inițială, oricine are acces la datele private poate "trage" fonduri și poate taxa proprietarul din nou și din nou. Astfel, rețeaua de plată trebuie securizată end-to-end cu criptare și trebuie să se asigure că oricine trage cu urechea sau orice intermediari nu pot compromite traficul de plată, în tranzit sau când este stocat (în repaus). Dacă un actor rău-făcător obține acces la sistem, el poate compromite tranzacțiile curente și datele de plată care pot fi utilizate pentru a crea noi tranzacții. Mai rău, atunci când datele clientului sunt compromise, clienții sunt expuși la furtul de identitate și trebuie să ia măsuri pentru a preveni utilizarea frauduloasă a conturilor compromise.

Bitcoin este cu totul diferit. O tranzacție bitcoin autorizează doar o valoare specifică pentru un destinatar specific și nu poate fi falsificată sau modificată. Nu dezvăluie nicio informație privată, precum identitatea părților și nu poate fi utilizată pentru a autoriza plăți suplimentare. Prin urmare, o rețea de plată bitcoin nu trebuie să fie criptată sau protejată. De fapt, puteți trimite tranzacții bitcoin pe un canal public deschis, cum ar fi WiFi sau Bluetooth nesigur, fără pierderea securității.

Modelul de securitate descentralizat al bitcoin pune multă putere în mâinile utilizatorilor. Cu

această putere vine responsabilitatea de a menține secretul cheilor. Pentru majoritatea utilizatorilor, acest lucru nu este ușor de făcut, în special pe dispozitive de calcul cu scop general, cum ar fi smartphone-urile sau laptopurile conectate la internet. Deși modelul descentralizat al bitcoin împiedică tipul de compromis în masă văzut cu cardurile de credit, mulți utilizatori nu sunt capabili să-și securizeze în mod adecvat cheile și sunt compromiși, unul câte unul.

## Dezvoltarea Sistemelor Bitcoin în Siguranță

Cel mai important principiu pentru dezvoltatorii bitcoin este descentralizarea. Majoritatea programatorilor vor fi familiarizați cu modelele de securitate centralizate și ar putea fi tentați să aplique aceste modele aplicațiilor lor bitcoin, cu rezultate dezastruoase.

Securitatea bitcoin se bazează pe controlul descentralizat asupra cheilor și pe validarea tranzacțiilor independent de către mineri. Dacă doriți să beneficiați de securitatea bitcoin, trebuie să vă asigurați că rămâneți în cadrul modelului de securitate bitcoin. În termeni simpli: nu luați controlul cheilor de la utilizatori și nu scoateți tranzacțiile de pe lanțul-de-blocuri.

De exemplu, multe burse-de-schimb bitcoin timpurii au concentrat toate fondurile utilizatorilor într-un singur portofel "fierbinte" cu cheile stocate pe un singur server. Un astfel de design elimină controlul utilizatorilor și centralizează controlul asupra cheilor dintr-un singur sistem. Multe astfel de sisteme au fost sparte, cu consecințe dezastruoase pentru clienții lor.

O altă greșală comună este de a lua tranzacțiile "în afara lanțului-de-blocuri" într-un efort greșit pentru a reduce comisioanele tranzacției sau pentru a accelera procesarea tranzacțiilor. Un sistem "în afara lanțului-de-blocuri" va înregistra tranzacțiile pe un registru intern, centralizat și le va sincroniza doar ocazional cu lanțul-de-blocuri bitcoin. Această practică, din nou, înlocuiește securitatea bitcoin descentralizată cu o abordare proprietară și centralizată. Atunci când tranzacțiile sunt în afara lanțului-de-blocuri, evidențele centralizate securizate în mod necorespunzător pot fi falsificate, devalorizând fondurile și golind rezervele în mod neobservat.

Dacă nu sunteți pregătiți să investiți puternic în securitatea operațională, în mai multe niveluri de control al accesului și în audituri (așa cum fac băncile tradiționale), ar trebui să vă gândiți foarte atent înainte de a scoate fonduri în afara contextului de securitate descentralizat al bitcoin. Chiar dacă aveți fonduri și disciplina pentru a implementa un model de securitate robust, un astfel de design nu face decât să reproducă modelul fragil al rețelelor financiare tradiționale, afectat de furtul de identitate, corupție și delapidare. Pentru a profita de modelul unic de securitate descentralizat al bitcoin, trebuie să evitați tentația arhitecturilor centralizate cu care puteți fi familiar, dar care în cele din urmă subminează securitatea bitcoin.

## Rădăcină de Încredere

Arhitectura tradițională de securitate se bazează pe un concept numit *rădăcină de încredere*, care este un nucleu de încredere folosit ca bază pentru securitatea sistemului general sau al aplicației. Arhitectura de securitate este dezvoltată în jurul rădăcinii de încredere ca o serie de cercuri concentrice, precum straturile dintr-o ceapă, extinzând încrederea spre exterior din centru. Fiecare strat se bazează pe stratul interior mai de încredere folosind controale de acces, semnături digitale, criptare și alte primitive de securitate. Deoarece sistemele software devin mai complexe, acestea sunt mai susceptibile în a conține erori, ceea ce le face vulnerabile la compromisurile de securitate. Ca urmare, cu cât un sistem software este mai complex, cu atât este mai greu să îl asigurați.

Conceptul de rădăcină de încredere asigură că cea mai mare parte a încredерii este plasată în partea cea mai puțin complexă a sistemului și, prin urmare, cea mai puțin vulnerabilă, în timp ce software-ul mai complex este plasat în jurul ei. Această arhitectură de securitate se repetă la diferite scări, stabilindu-se mai întâi o rădăcină de încredere în hardware-ul unui singur sistem, apoi extinzând acea rădăcină de încredere prin sistemul de operare la serviciile de nivel superior, iar în final pe multe servere stratificate în cercuri concentrice de încredere diminuată.

Arhitectura de securitate bitcoin este diferită. În bitcoin, sistemul de consens creează un registru public de încredere care este complet descentralizat. Un lanț-de-blocuri validat corect utilizează blocul geneză ca rădăcină de încredere, construind un lanț de încredere până la blocul actual. Sistemele bitcoin pot și trebui să utilizeze lanțul-de-blocuri ca rădăcină de încredere. Atunci când proiectați o aplicație bitcoin complexă care constă în servicii pe mai multe sisteme diferite, ar trebui să examinați cu atenție arhitectura de securitate pentru a afla unde se plasează încrederea. În cele din urmă, singurul lucru care ar trebui să fie în mod explicit de încredere este un lanț-de-blocuri complet validat. Dacă aplicația dumneavoastră implică în mod explicit sau implicit încredere în orice altceva decât în lanțul-de-blocuri, aceasta ar trebui să fie o sursă de îngrijorare, deoarece introduce vulnerabilitate. O metodă bună de evaluare a arhitecturii de securitate a aplicației dumneavoastră este de a lua în considerare fiecare componentă individuală și de a evalua un scenariu ipotetic în care componenta respectivă este complet compromisă și sub controlul unui actor rău intenționat. Luați pe rând fiecare componentă a aplicației dumneavoastră și evaluați impactul asupra securității generale dacă acea componentă este compromisă. Dacă aplicația dumneavoastră nu mai este sigură atunci când componente sunt compromise, acest lucru arată că aveți încredere nepotrivită în acele componente. O aplicație bitcoin fără vulnerabilități ar trebui să fie vulnerabilă doar la un compromis al mecanismului de consens bitcoin, ceea ce înseamnă că rădăcina sa de încredere se bazează pe cea mai puternică parte a arhitecturii de securitate bitcoin.

Numeroasele exemple de burse-de-schimb bitcoin sparte servesc pentru a sublinia acest punct, deoarece arhitectura și designul lor de securitate nu trece nici chiar de cel mai ocazional control. Aceste implementări centralizate au investit încredere în mod explicit în numeroase componente din afara lanțului-de-blocuri bitcoin, cum ar fi portofelele fierbinți, bazele de date centralizate pentru registru, cheile de criptare vulnerabile și scheme similare.

## Cele mai Bune Practici pentru Securitatea Utilizatorului

Oamenii au folosit controale pentru securitatea fizică de mii de ani. Prin comparație, experiența noastră cu securitatea digitală este mai mică de 50 de ani. Sistemele de operare moderne cu scop general nu sunt foarte sigure și nu sunt adecvate în special pentru stocarea de bani digitali. Calculatoarele noastre sunt expuse constant la amenințări externe prin intermediul conexiunilor de internet permanente. Rulează mii de componente software de la sute de autori, adesea cu acces neconstrâns la fișierele utilizatorului. O singură componentă de software necinstit, dintre numeroasele mii instalate pe calculator, vă poate compromite tastatura și fișierele, furând orice bitcoin stocat în aplicații portofel. Nivelul necesar de menenanță pentru a menține un calculator fără virus și troian depășește nivelul de calificare al multora, în afară de o mică minoritate de utilizatori de calculatoare.

În ciuda a zeci de ani de cercetare și avansări în securitatea informațiilor, activele digitale sunt încă foarte vulnerabile la un adversar hotărât. Chiar și cele mai protejate și restricționate sisteme, în companii de servicii financiare, agenții de informații și contractori de apărare, au frecvență mare. Bitcoin creează active digitale care au o valoare intrinsecă și pot fi furate și redirecționate către noi proprietari instantaneu și irevocabil. Acest lucru creează un stimulent masiv pentru hackeri. Până acum, hackerii trebuiau să transforme informațiile de identitate sau datele contului - cum ar fi cărțile de credit și conturile bancare - în valoare după ce le-au compromis. În ciuda dificultăților de îngădare și de spălare a informațiilor financiare, am văzut furturi în continuă ascensiune. Bitcoin crește această problemă, deoarece nu trebuie să fie îngăduit sau spălat; este o valoare intrinsecă în cadrul unui activ digital.

Din fericire, bitcoin creează, de asemenea, stimulente pentru îmbunătățirea securității calculatorului. Deși anterior riscul de compromitere a calculatorului era vag și indirect, bitcoin face aceste riscuri clare și evidente. Înținerea bitcoin pe un computer servește pentru a concentra mintea utilizatorului asupra necesității îmbunătățirii securității computerului. Ca rezultat direct al proliferării și adoptării crescute a bitcoin și a altor monede digitale, am observat o escaladare atât în tehnici de hacking, cât și în soluțiile de securitate. În termeni simpli, hackerii au acum o țintă foarte succulentă, iar utilizatorii au un stimulent clar să se apere.

În ultimii trei ani, ca rezultat direct al adoptării bitcoin, am observat o inovație extraordinară în domeniul securității informațiilor sub formă de criptare hardware, stocare a cheilor și portofele hardware, tehnologie cu mai multe semnături și împăternicire legală digitală. În secțiunile următoare vom examina diverse practici recomandate pentru securitatea utilizatorului.

## Stocare fizică Bitcoin

Deoarece majoritatea utilizatorilor sunt mult mai în largul lor cu securitatea fizică decât securitatea informațiilor, o metodă foarte eficientă pentru protejarea bitcoin este transformarea lor în formă fizică. Cheile bitcoin nu sunt altceva decât numere lungi. Aceasta înseamnă că pot fi stocate într-o formă fizică, cum ar fi tipărite pe hârtie sau gravate pe o monedă metalică. Securizarea cheilor devine apoi la fel de simplă ca securizarea fizică a copiei tipărite a cheilor bitcoin. Un set de chei bitcoin care sunt tipărite pe hârtie se numește "portofel de hârtie" și există multe instrumente gratuite care pot fi utilizate pentru a le crea. Eu personal păstrează mareala majoritate a bitcoinului meu (99% sau mai mult) pe portofele de hârtie, criptate cu BIP-38, cu mai multe copii blocate în seifuri. Păstrarea bitcoin offline se numește *stocare la rece* și este una dintre cele mai eficiente tehnici de securitate. Un sistem de stocare la rece este unul în care cheile sunt generate pe un sistem offline (unul niciodată conectat la internet) și stocate offline fie pe hârtie, fie pe suport digital, cum ar fi un stick de memorie USB.

## Portofele Hardware

Pe termen lung, securitatea bitcoin va lua din ce în ce mai mult forma portofelelor anti-manipulare hardware. Spre deosebire de un smartphone sau un calculator desktop, un portofel hardware bitcoin are doar un singur scop: păstrarea bitcoin în siguranță. Fără un software cu scop general pentru a face compromisuri și cu interfețe limitate, portofelele hardware pot oferi un nivel de securitate aproape infailabil utilizatorilor care nu sunt experți. Mă aștept ca portofelele hardware să devină metoda predominantă de stocare bitcoin. Pentru un exemplu de astfel de portofel hardware, consultați [Trezor](#).

## Echilibrarea Riscului

Deși majoritatea utilizatorilor sunt îngrijorați pe bună dreptate de furtul bitcoin, există un risc și mai mare. Fișierele de date se pierd tot timpul. Dacă conțin bitcoin, pierderea este mult mai dureroasă. În efortul de a-și asigura portofelele bitcoin, utilizatorii trebuie să fie foarte atenți să nu meargă prea departe și să ajungă să piardă bitcoin. În iulie 2011, un cunoscut proiect de conștientizare și educație bitcoin a pierdut aproape 7.000 de bitcoin. În efortul lor de a preveni furtul, proprietarii au implementat o serie complexă de copii de rezervă criptate. În cele din urmă, au pierdut din greșală cheile de criptare, făcând copii de rezervă fără valoare și pierzând o avere. Ca și cum ai ascunde bani îngropându-i în deșert, dacă îți asiguri prea bine bitcoin-ul, s-ar putea să nu-l mai găsești.

## Diversificarea Riscului

Ați transporta în portofel întreaga avere în numerar? Cei mai mulți oameni ar considera necugetat, totuși de obicei, utilizatorii bitcoin își păstrează adesea tot bitcoinul într-un singur portofel. În schimb, utilizatorii ar trebui să răspândească riscul printre portofele bitcoin multiple și diverse. Utilizatorii prudenți vor păstra doar o fracție mică, poate mai mică de 5%, din bitcoinul lor într-un portofel online sau mobil ca "marunțis". Restul trebuie împărțit între câteva mecanisme de stocare diferite, cum ar fi un portofel pentru desktop și offline (stocare la rece).

## Multisig și Guvernanță

Ori de câte ori o companie sau o persoană individuală stochează cantități mari de bitcoin, ar trebui să ia în considerare utilizarea unei adrese bitcoin multisemnătură. Adresele multisemnătură securizează fondurile, necesitând mai multe semnături pentru a efectua o plată. Cheile de semnare ar trebui să fie stocate într-o serie de locații diferite și sub controlul diferitelor persoane. Într-un mediu corporativ, de exemplu, cheile ar trebui să fie generate independent și deținute de mai mulți directori ai companiei, pentru a se asigura că nicio persoană nu poate compromite fondurile. Adresele multisemnătură pot oferi, de asemenea, redundanță, atunci când o singură persoană deține mai multe chei care sunt stocate în diferite locații.

## Supraviețuirea

O considerentă de securitate importantă care este adesea trecută cu vederea este disponibilitatea, în special în contextul incapacității sau decesului titularului de cheie. Utilizatorilor bitcoin li se spune să folosească parole complexe și să-și păstreze cheile în siguranță și în mod privat, fără să le împărtășească cu nimeni. Din păcate, această practică face aproape imposibilă recuperarea de fonduri de către familia utilizatorului, dacă utilizatorul nu este disponibil pentru a le debloca. În cele mai multe cazuri, familiile utilizatorilor de bitcoin ar putea să nu știe de loc de existență fondurilor bitcoin.

Dacă aveți mult bitcoin, ar trebui să luați în considerare distribuirea detaliilor de acces cu o rudă sau un avocat de încredere. O schemă de supraviețuire mai complexă poate fi configurată cu acces multi-semnătură și planificare a proprietății prin intermediul unui avocat specializat în calitate de "executord de active digitale".

# Concluzie

Bitcoin este o tehnologie complet nouă, fără precedent și complexă. În timp, vom dezvolta instrumente și practici de securitate mai bune, care sunt mai ușor de utilizat de către non-experti. Deocamdată, utilizatorii bitcoin pot folosi multe dintre sfaturile discutate aici pentru a se bucura de o experiență bitcoin sigură și fără probleme.

# Aplicații Lanț-de-Blocuri

Să construim pe cunoștințele noastre despre bitcoin din perspectiva unei platforme pentru aplicații. În zilele noastre, multe persoane folosesc termenul "blockchain" pentru a face referire la orice platformă de aplicații care împărtășește principiile de design ale bitcoin. Termenul este adesea folosit greșit și aplicat la multe lucruri care nu reușesc să furnizeze funcționalitățile principale pe care le oferă lanțul-de-blocuri (blockchain-ul) bitcoin.

În acest capitol vom analiza caracteristicile oferite de lanțul-de-blocuri bitcoin, ca platformă de aplicații. Vom lua în considerare dezvoltarea de aplicații *primitive*, care formează elementele de bază ale oricărei aplicații blockchain. Vom analiza mai multe aplicații importante care utilizează aceste elemente primitive, precum canale de plată (cu stare) și canale de plată rutate (Lightning Network).

## Introducere

Sistemul bitcoin a fost conceput ca un sistem de plată și monedă descentralizată. Cu toate acestea, cea mai mare parte a funcționalității sale este derivată din componente de nivel mult mai scăzut, care pot fi utilizate pentru aplicații mult mai largi. Bitcoin nu a fost construit cu componente precum conturi, utilizatori, solduri și plăți. În schimb, folosește un limbaj de scriptare tranzacțional cu funcții criptografice de nivel scăzut, așa cum am văzut în [Tranzacții](#). La fel cum conceptele de nivel superior precum conturi, solduri și plăți pot fi derivate din aceste elemente de bază, la fel și multe alte aplicații complexe. Astfel, lanțul-de-blocuri bitcoin poate deveni o platformă de aplicații care oferă servicii de încredere pentru aplicații, cum ar fi contractele inteligente, depășind cu mult scopul inițial al monedei digitale și al plăților.

## Pietre de Temelie (primitive)

Când funcționează corect și pe termen lung, sistemul bitcoin oferă anumite garanții, care pot fi utilizate ca pietre de temelie pentru a crea aplicații. Acestea includ:

### Fără Cheltuire-Dublă

Cea mai fundamentală garanție a algoritmului de consens descentralizat al bitcoin asigură că nici un UTXO nu poate fi cheltuit de două ori.

### Imutabilitate

Odată ce o tranzacție este înregistrată în lanțul-de-blocuri și s-au adăugat suficiente blocuri ulterioare, datele tranzacției devin imutabile. Imutabilitatea este subscrisă de energie, deoarece rescrierea lanțului-de-blocuri necesită cheltuirea de energie pentru a produce Dovadă-de-Lucru.

Energia necesară și, prin urmare, gradul de imutabilitate crește odată cu cantitatea de muncă depusă deasupra blocului care conține o tranzacție.

## Neutralitate

Rețeaua descentralizată bitcoin propagă tranzacții valide indiferent de originea sau conținutul tranzacțiilor respective. Acest lucru înseamnă că oricine poate crea o tranzacție validă cu suficient comision poate avea încredere că va putea transmite acea tranzacție și că va fi inclusă în lanțul-de-blocuri la orice moment.

## Marcă de timp securizată (timestamp)

Regulile de consens resping orice bloc al cărui marcă de timp este prea departe în trecut sau în viitor. Acest lucru asigură că mărcile de timp de pe blocuri pot fi de încredere. Marca de timp de pe un bloc implică o garanție necheltuită-înainte pentru intrările tuturor tranzacțiilor incluse.

## Autorizare

Semnăturile digitale, validate într-o rețea descentralizată, oferă garanții de autorizare. Scripturile care conțin o cerință pentru o semnătură digitală nu pot fi executate fără autorizarea titularului cheii private implicate în script.

## Auditabilitate

Toate tranzacțiile sunt publice și pot fi auditate. Toate tranzacțiile și blocurile pot fi legate într-un lanț neîntrerupt până la blocul geneză.

## Contabilitate

În orice tranzacție (cu excepția tranzacției coinbase), valoarea intrărilor este egală cu valoarea ieșirilor plus a comisioanelor. Nu este posibilă crearea sau distrugerea valorii bitcoin într-o tranzacție. Ieșirile nu pot depăși intrările.

## Nonexpirare

O tranzacție validă nu expiră. Dacă este validă astăzi, va fi validă în viitorul apropiat, atâtă timp cât intrările rămân necheltuite iar regulile de consens nu se modifică.

## Integritate

O tranzacție bitcoin semnată cu **SIGHASH\_ALL** sau părți ale unei tranzacții semnate de un alt tip **SIGHASH** nu pot fi modificate fără a invalida semnătura, invalidând astfel tranzacția în sine.

## Atomicitate tranzacție

Tranzacțiile Bitcoin sunt atomice. Sunt valide și confirmate (minate) sau nu. Tranzacțiile parțiale nu pot fi minate și nu există o stare provizorie pentru o tranzacție. În orice moment, o tranzacție este minată sau nu.

## Unități de Valoare Discrete (indivizibile)

Ieșirile tranzacției sunt unități de valoare discrete și indivizibile. Pot fi fie cheltuite, fie necheltuite integral. Nu pot fi divizate sau cheltuite parțial.

## Cvorul de Control

Constrângerile *multisig* în scripturi impun un cvorum de autorizare, predefinit în schema multisemnătură. Cerința M-din-N este impusă de regulile de consens.

## **Timpi de Blocare/ Îmbătrânire**

Orice clauză de script care conține un timp de blocare (timelock) relativ sau absolut poate fi executată numai după ce vârsta sa depășește timpul specificat.

## **Replicare**

Stocarea descentralizată a lanțului-de-blocuri asigură că atunci când o tranzacție este minată, după confirmări suficiente, aceasta este replicată în rețea și devine durabilă și rezistentă la pierderi de energie, pierderi de date etc.

## **Pretecție Anti-Falsificare**

O tranzacție poate cheltui numai ieșiri existente, validate. Nu este posibilă crearea sau contrafacerea valorii.

## **Consistență**

În absența partaționării minerilor, blocurile care sunt înregistrate în lanțul-de-blocuri sunt supuse reorganizării sau dezacordului cu o probabilitate în scădere exponențială, pe baza adâncimii la care sunt înregistrate. Odată înregistrat în profunzime, calculul și energia necesară pentru schimbare fac ca schimbarea să fie practic imposibilă.

## **Înregistrarea Stării Externe**

O tranzacție poate comite o valoare a datelor, prin **OP\_RETURN**, reprezentând o tranziție de stare într-o mașină de stare externă.

## **Emitere Previzibilă**

Vor fi emise mai puțin de 21 de milioane de bitcoin, într-un ritm previzibil.

Lista pietrelor de temelie nu este completă și sunt adăugate mai multe cu fiecare funcționalitate nou-introdusă în bitcoin.

# **Aplicații Construite pe baza Pietrelor de Temelie**

Pietrele de temelie oferite de bitcoin sunt elemente ale unei platforme de încredere care pot fi utilizate pentru compunerea aplicațiilor. Iată câteva exemple de aplicații care există astăzi și pietrele de temelie pe care le folosesc:

## **Dovada-de-Existență (notar digital)**

Imutabilitate + Marcă de Timp + Durabilitate. O amprentă digitală poate fi consemnată cu o tranzacție în lanțul-de-blocuri, dovedind că un document a existat (marca-de-timp) la momentul înregistrării. Amprenta nu poate fi modificată ex-post-facto (Imutabilitate) și dovada va fi păstrată permanent (Durabilitate).

## **Kickstarter (Lighthouse)**

Consistență + Atomicitate + Integritate. Dacă semnați o intrare și ieșirea (Integritatea) unei tranzacții de strângere de fonduri, alții pot contribui la strângerea de fonduri, dar fondurile nu pot fi cheltuite (Atomicitate) până când obiectivul (valoarea de ieșire) este finanțat (consecvență).

## Canale de plată

Cvorum de Control + Timpi de Blocare + Fără Cheltuiuri Duble + Nonexpirare + Rezistență la Cenzură + Autorizare. O multisemnătură 2-din-2 (Cvorum) cu un timp-de-blocare (Timelock) folosită ca tranzacție de "decontare" a unui canal de plată poate fi păstrată (Nonexpirare) și cheltuită în orice moment (Rezistență de Cenzură) de către oricare parte (Autorizare). Cele două părți pot apoi să creeze tranzacții de angajament care să cheltuiască-dublu (Fără Cheltuiuri-Duble) decontarea într-un interval mai scurt (Timelock).

## Contrapartida

Contrapartida este un strat de protocol construit peste bitcoin. Protocolul de contrapartidă oferă posibilitatea de a crea și de a tranzacționa active și tokeni virtuali. În plus, contrapartida oferă un schimb descentralizat pentru active. Contrapartida implementează, de asemenea, contracte inteligente, bazate pe Mașina Virtuală Ethereum (EVM).

Contrapartida include metadate în tranzacțiile bitcoin, folosind operatorul **OP\_RETURN** sau adresele multisemnătură 1-din-N care codifică metadatele în locul cheilor publice. Folosind aceste mecanisme, Contrapartida implementează un strat de protocol codat în tranzacții bitcoin. Stratul suplimentar de protocol poate fi interpretat de aplicații care sunt conștiente de contrapartidă, cum ar fi portofelele și exploratorii lanț-de-blocuri, sau orice aplicație construită folosind bibliotecile contrapartidei.

Contrapartida poate fi folosită ca o platformă pentru alte aplicații și servicii. De exemplu, Tokenly este o platformă construită peste Contrapartidă care permite creatorilor de conținut, artiștilor și companiilor să emită tokeni care exprimă proprietatea digitală și poate fi folosită pentru închiriere, acces, comerț sau cumpărături pentru conținut, produse și servicii. Alte aplicații care folosesc contrapartida includ jocuri (Spells of Genesis) și proiecte de calcul grilă (Folding Coin).

Mai multe detalii despre Contrapartidă pot fi găsite la <https://counterparty.io>. Proiectul open source poate fi găsit la <https://github.com/CounterpartyXCP>.

## Canale de Plată și Canale de Stare

*Canalele de plată* sunt un mecanism fără încredere (în terți) pentru schimbul de tranzacții bitcoin între două părți, în afara lanțului-de-blocuri bitcoin. Aceste tranzacții, care ar fi valide dacă ar fi decontate pe lanțul-de-blocuri bitcoin, sunt în schimb păstrate în afara lanțului, acționând ca *bilete la ordin* pentru eventuala decontare. Deoarece tranzacții nu sunt decontate, ele pot fi trimise fără latență de decontare (settlement latency) obișnuită, permitând o viteză extrem de ridicată a tranzacțiilor, latență scăzută (submillisecunde) și granularitate fină (la nivel de satoshi).

De fapt, termenul *canal* este o metaforă. Canalele cu stare sunt construcții virtuale reprezentate de schimbul de stare între două părți, în afara lanțului-de-blocuri. Nu există "canale" în sine, iar mecanismul de transport al datelor de bază nu este canalul. Folosim termenul *canal* pentru a reprezenta relația și starea comună între două părți, în afara lanțului-de-blocuri.

Pentru a explica în continuare acest concept, gândiți-vă la un flux TCP. Din perspectiva protocolelor la nivel superior, este o "mufă" care conectează două aplicații pe internet. Dar dacă te uiți la traficul de rețea, un flux TCP este doar un canal virtual peste pachetele IP. Fiecare

extremitate a unui flux TCP pune în ordine și asamblează pachete IP pentru a crea iluzia unui flux de octeți. Dedeșubt, toate pachetele sunt deconectate. În mod similar, un canal de plată este doar o serie de tranzacții. Dacă sunt ordonate și conectate corespunzător, acestea creează obligații de răscumpărare în care puteți avea încredere, chiar dacă nu aveți încredere în cealaltă parte a canalului.

În această secțiune vom analiza diverse forme de canale de plată. În primul rând, vom examina mecanismele utilizate pentru a construi un canal de plată unidirectional pentru un serviciu de microplată contorizat, cum ar fi streaming video. Apoi, vom extinde acest mecanism și vom introduce canale de plată bidirectionale. În cele din urmă, vom analiza modul în care canalele bidirectionale pot fi conectate de la un capăt la altul pentru a forma canale multihop într-o rețea rutată, propusă prima dată sub denumirea de *Lightning Network*.

Canalele de plată fac parte din conceptul mai larg de *canal cu stare*, care reprezintă o modificare a stării în-afara-lanțului, securizată prin eventuala decontare într-un lanț-de-blocuri. Un canal de plată este un canal cu stare în care starea modificată este balanța unei monede virtuale.

## Canale cu Stare - Concepte de Bază și Terminologie

Un canal cu stare se stabilește între două părți, printr-o tranzacție care blochează o stare comună pe lanțul-de-blocuri. Aceasta se numește *tranzacție de finanțare* (funding transaction) sau *tranzacție ancoră*. Această tranzacție unică trebuie transmisă în rețea și minată pentru a înființa canalul. În exemplul unui canal de plată, starea blocată este soldul inițial (în monedă virtuală) al canalului.

Cele două părți schimbă apoi tranzacții semnate, numite *tranzacții de angajament* (commitment transactions), care modifică starea inițială. Aceste tranzacții sunt tranzacții valide, prin faptul că ele pot fi trimise spre soluționare de către oricare dintre părți, dar mai degrabă sunt ținute în afara lanțului de către fiecare parte în așteptarea închiderii canalului. Actualizările de stare pot fi create cât de repede fiecare parte poate crea, semnă și transmite o tranzacție celeilalte părți. În practică, aceasta înseamnă că se pot schimba mii de tranzacții pe secundă.

Atunci când fac schimb de tranzacții de angajament, cele două părți invalidează de asemenea stările anterioare, astfel încât cea mai curentă tranzacție de angajament este întotdeauna singura care poate fi răscumpărată. Acest lucru împiedică oricare dintre părți să înceleze prin închiderea unilaterală a canalului cu o stare anterioară expirată care le este mai favorabilă decât starea curentă. Vom examina diferențele mecanisme care pot fi utilizate pentru a invalida starea anterioară în restul acestui capitol.

În cele din urmă, canalul poate fi închis fie în mod cooperant, prin trimiterea unei tranzacții de decontare finală către lanțul-de-blocuri, sau unilateral, de oricare dintre părți care transmite ultima tranzacție de angajament către lanțul-de-blocuri. O opțiune de închidere unilaterală este necesară în cazul în care una dintre părți se deconectează în mod neașteptat. Tranzacția de decontare reprezintă starea finală a canalului și este decontată pe lanțul-de-blocuri.

Pe întreaga durată de viață a canalului, doar două tranzacții trebuie depuse pentru minare pe lanțul-de-blocuri: tranzacțiile de finanțare (funding) și decontare (settlement). Între aceste două stări, cele două părți pot schimba orice număr de tranzacții de angajament care nu sunt niciodată văzute de nimeni altcineva și nici transmise către lanțul-de-blocuri.

**Un canal de plată între Bob și Alice, care arată tranzacțiile de finanțare, angajament și decontare** ilustrează un canal de plată între Bob și Alice, arătând tranzacțiile de finanțare, angajament și decontare.

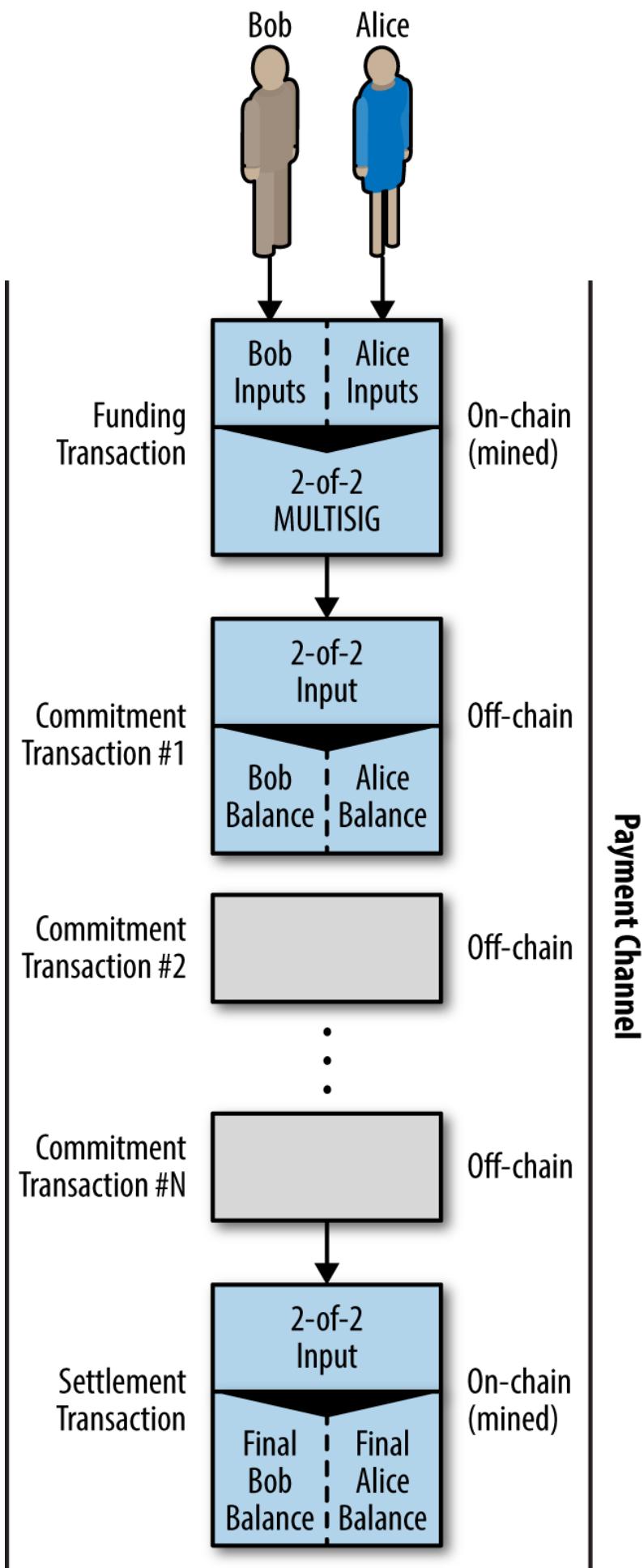


Figure 76. Un canal de plată între Bob și Alice, care arată tranzacțiile de finanțare, angajament și

## Exemplu de Canal de Plată Simplu

Pentru a explica canalele cu stare, începem cu un exemplu foarte simplu. Demonstrăm un canal unidirecțional, ceea ce înseamnă că valoarea circulă într-o singură direcție. De asemenea, vom începe cu presupunerea naivă că nimeni nu încearcă să îñsele, pentru a menține lucrurile simple. După ce am explicitat ideea de bază a canalului, vom analiza de ce anume este nevoie pentru a-l face să nu necesite încredere în terți, astfel încât niciuna dintre părți nu poate îñsela, chiar dacă încearcă.

Pentru acest exemplu vom presupune doi participanți: Emma și Fabian. Fabian oferă un serviciu de streaming video care este facturat la secundă folosind un canal de microplată. Fabian taxează 0,01 milibit (0,00001 BTC) pe secundă de videoclip, echivalentul a 36 milibiți (0,036 BTC) pe oră video. Emma este un utilizator care achiziționează acest serviciu de video streaming de la Fabian. **Emma achiziționează streaming video de la Fabian folosind un canal de plată, plătind pentru fiecare secundă a videoclipului** o arată pe Emma care cumpără serviciul de streaming video de la Fabian folosind un canal de plată.

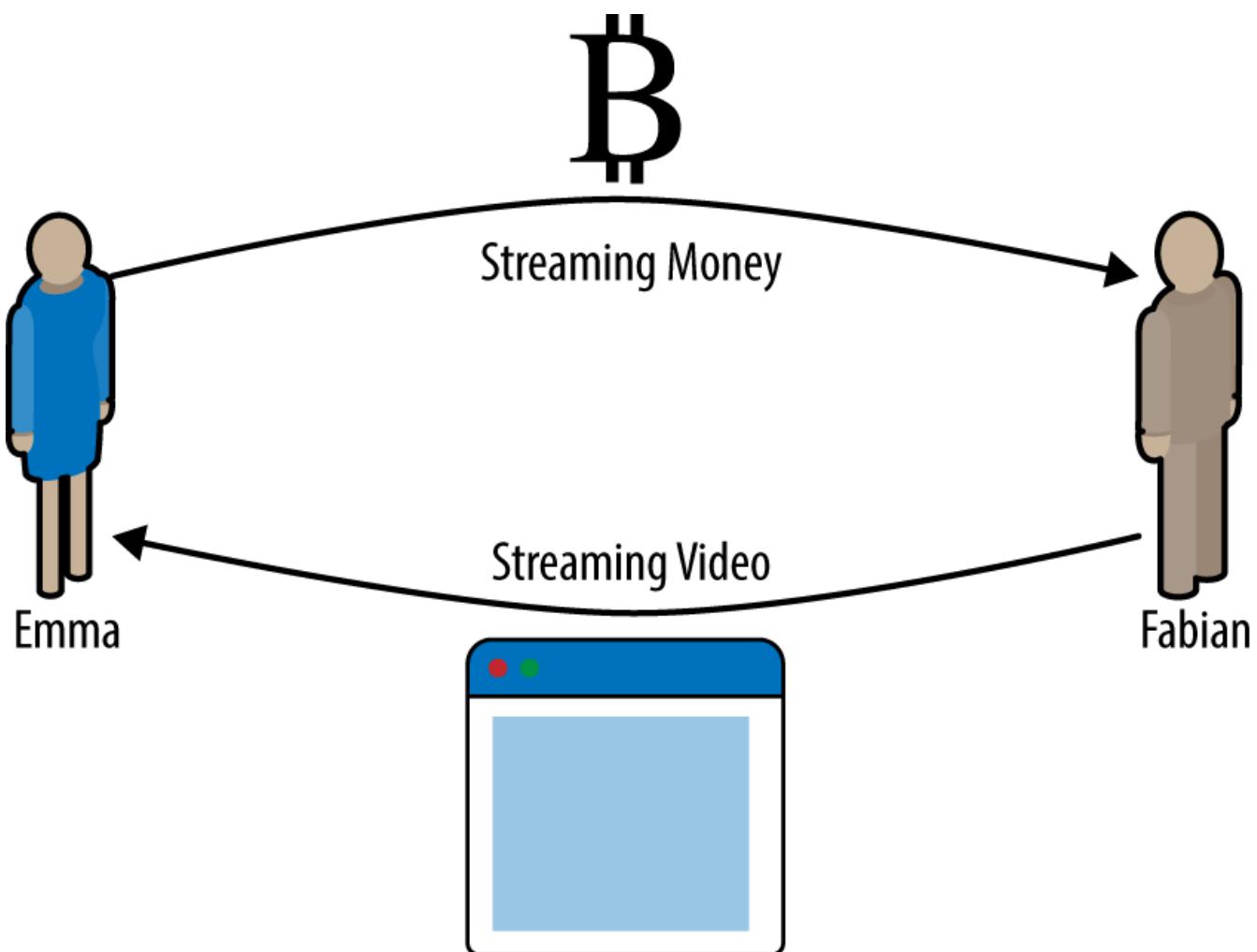


Figure 77. Emma achiziționează streaming video de la Fabian folosind un canal de plată, plătind pentru fiecare secundă a videoclipului

În acest exemplu, Fabian și Emma folosesc un software special care se ocupă atât de canalul de plată, cât și de transmisia video. Emma rulează software-ul în browserul său, Fabian îl rulează pe

un server. Software-ul include funcționalitatea de bază unui portofel bitcoin și poate crea și semnează tranzacții. Atât conceptul, cât și termenul "canal de plată" sunt ascunse complet utilizatorilor. Ceea ce văd ei este un videoclip pentru care este plătit la secundă.

Pentru a configura canalul de plată, Emma și Fabian stabilesc o adresă multisemnătură 2-din-2, fiecare dintre ei deținând una dintre chei. Din perspectiva Emmei, software-ul din browserul său afișează un cod QR cu o adresă P2SH (care începe cu "3") și îi cere să depună un "depozit de garantare" pentru maxim o oră de videoclip. Adresa este apoi finanțată de Emma. Tranzacția Emmei, plătită la adresa multisemnătură, este tranzacția de finanțare sau ancoră pentru canalul de plată.

Pentru acest exemplu, să spunem că Emma finanțează canalul cu 36 de milibit (0,036 BTC). Acest lucru îi va permite Emmei să consume *până la* o oră de transmisie video. În acest caz, tranzacția de finanțare stabilește suma maximă care poate fi transmisă în acest canal, setând *capacitatea canalului*.

Tranzacția de finanțare consumă una sau mai multe intrări din portofelul Emmei, furnizând fonduri. Creează o ieșire cu o valoare de 36 de milibit plătită adresei multisemnătură 2-din-2 controlată în comun de Emma și Fabian. Este posibil să aibă ieșiri suplimentare pentru rest înapoi în portofelul Emmei.

Odată confirmată tranzacția de finanțare, Emma poate începe să streaming-ul video. Software-ul Emmei creează și semnează o tranzacție de angajament care schimbă soldul canalului pentru a credita 0,01 milibit la adresa lui Fabian și a restitu 35,99 milibit înapoi către Emma. Tranzacția semnată de Emma consumă ieșirea de 36 de milibit creată de tranzacția de finanțare și creează două ieșiri: una pentru rambursarea către ea, cealaltă pentru plata către Fabian. Tranzacția este doar parțial semnată - necesită două semnături (2-din-2), dar are doar semnatura Emmei. Când serverul lui Fabian primește această tranzacție, acesta adaugă a doua semnătură (pentru intrarea 2-din-2) și o returnează Emmei, împreună cu o secundă de videoclip. Acum, ambele părți au o tranzacție de angajament complet semnată pe care oricare o poate răscumpăra, reprezentând soldul corect actualizat al canalului. Niciuna dintre părți nu a transmis această tranzacție în rețea bitcoin.

În următoarea rundă, software-ul Emmei creează și semnează o altă tranzacție de angajament (angajamentul #2) care consumă *aceeași* ieșire 2-din-2 din tranzacția de finanțare. Cea de-a doua tranzacție de angajament alocă o ieșire de 0,02 milibit la adresa lui Fabian și o ieșire de 35,98 milibit înapoi la adresa Emmei. Această nouă tranzacție este plata pentru două secunde cumulate de videoclip. Software-ul lui Fabian semnează și returnează a doua tranzacție de angajament, împreună cu o altă secundă a videoclipului.

În acest fel, software-ul Emmei continuă să trimită tranzacții de angajament la serverul lui Fabian în schimbul streamingului video. Bilanțul canalului se acumulează treptat în favoarea lui Fabian, deoarece Emma consumă mai multe secunde de videoclip. Să zicem că Emma vizionează 600 de secunde (10 minute) de videoclip, creând și semnând 600 de tranzacții de angajament. Ultima tranzacție de angajament (#600) va avea două ieșiri, împărțind balanța canalului, 6 milibit la Fabian și 30 milibit la Emma.

În cele din urmă, Emma dă clic pe "Stop" pentru a opri transmiterea videoclipului. Fie Fabian, fie Emma pot transmite acum starea finală a tranzacției pentru decontare. Această ultimă tranzacție

este "tranzacția de decontare" și îi plătește lui Fabian pentru tot ce a consumat Emma, restituind restul tranzacției de finanțare către Emma.

[Canalul de plată al Emmei cu Fabian, care arată tranzacțiile de angajament care actualizează soldul canalului](#) arată canalul dintre Emma și Fabian și tranzacțiile de angajament care actualizează balanța canalului.

La final, doar două tranzacții sunt înregistrate pe lanțul-de-blocuri: tranzacția de finanțare care a stabilit canalul și o tranzacție de decontare care a alocat corect soldul final între cei doi participanți.

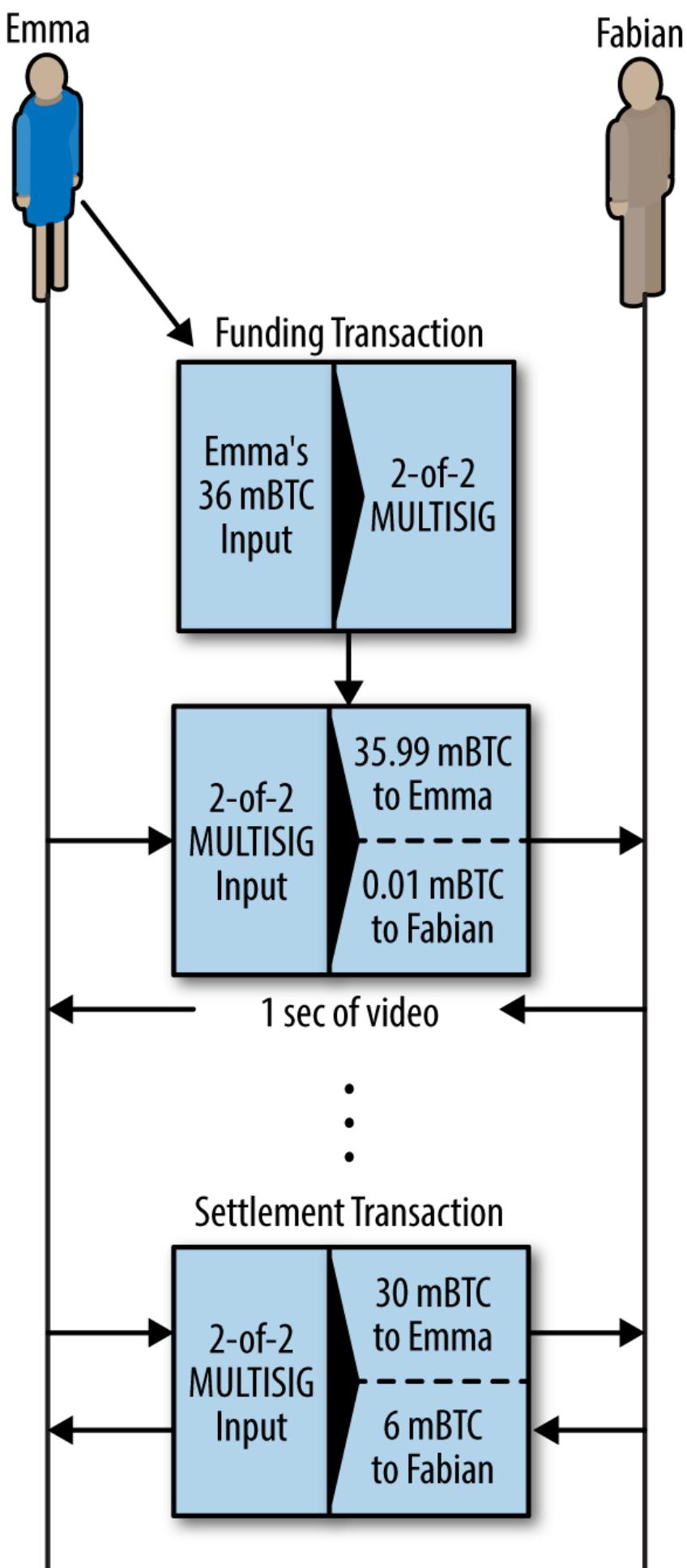


Figure 78. Canalul de plată al Emmei cu Fabian, care arată tranzacțiile de angajament care actualizează

## Realizarea Canalelor fără Încredere

Canalul pe care tocmai l-am descris funcționează, dar numai dacă ambele părți cooperează, fără erori sau încercări de a însela. Să ne uităm la unele dintre scenariile care pot strica acest canal și să vedem ce este necesar pentru a le remedia:

- Odată ce tranzacția de finanțare are loc, Emma are nevoie de semnătura lui Fabian pentru a obține bani înapoi. Dacă Fabian dispără, fondurile Emmei sunt blocate într-un contract 2-din-2 și se pierd efectiv. Acest canal, în felul în care este construit, duce la o pierdere de fonduri dacă una dintre părți se deconectează înainte de a exista cel puțin o tranzacție de angajament semnată de ambele părți.
- În timp ce canalul rulează, Emma poate să ia oricare dintre tranzacțiile de angajament pe care Fabian le-a contrasemnat și să transmită una dintre ele către lanțul-de-blocuri. De ce să plătească 600 de secunde de videoclip, dacă ea poate transmite tranzacția de angajament #1 și să plătească doar 1 secundă de videoclip? Canalul eșuează, deoarece Emma poate însela prin difuzarea unui angajament anterior care este în favoarea ei.

Ambele probleme pot fi rezolvate folosind **timpi-de-blocare** (timelocks) - să ne uităm la modul în care am putea utiliza **timpii-de-blocare** la nivel de tranzacție (**nLocktime**).

Emma nu poate risca să finanțeze o multisemnătură 2-din-2 decât dacă are o rambursare garantată. Pentru a rezolva această problemă, Emma construiește în același timp tranzacția de finanțare și rambursare. Ea semnează tranzacția de finanțare, dar nu o transmite nimănui. Emma transmite numai tranzacția de rambursare lui Fabian și obține semnătura sa.

Tranzacția de rambursare (refund) acționează ca prima tranzacție de angajament (commitment), iar timpul-de-blocare al acesteia stabilește limita superioară pentru viața canalului. În acest caz, Emma ar putea seta **nLocktime** la 30 de zile sau 4320 blocuri în viitor. Toate tranzacțiile de angajament ulterioare trebuie să aibă un interval de timp mai scurt, pentru a putea fi răscumpărate înainte de tranzacția de rambursare.

Acum, când Emma are o tranzacție de rambursare complet semnată, poate transmite cu încredere tranzacția de finanțare semnată știind că, în cele din urmă, după expirarea perioadei de timp-de-blocare, poate răscumpăra tranzacția de rambursare chiar dacă Fabian dispără.

Fiecare tranzacție de angajament pe care părțile o schimbă pe parcursul vieții canalului va avea timp-de-blocare în viitor. Dar întârzierea va fi puțin mai scurtă pentru fiecare angajament, astfel încât angajamentul cel mai recent poate fi răscumpărat înaintea angajamentului anterior pe care îl invalidează. Datorită **nLockTime**, niciuna dintre părți nu poate propaga cu succes vreuna din tranzacțiile de angajament până la expirarea timpului-de-blocare. Dacă totul merge bine, vor coopera și vor închide canalul în pace cu o tranzacție de decontare, ceea ce face inutilă transmiterea unei tranzacții de angajament intermediu. Dacă nu, cea mai recentă tranzacție de angajament poate fi propagată pentru a deconta contul și a invalida toate tranzacțiile de angajament anterioare.

De exemplu, dacă tranzacția de angajament #1 are timp-de-blocare peste 4320 de blocuri în viitor, atunci tranzacția de angajament #2 are timp de blocare 4319 blocuri în viitor. Tranzacția de

angajament #600 poate fi cheltuită cu 600 de blocuri înainte ca tranzacția de angajament #1 să devină validă.

Fiecare angajament stabilăște un **temp-de-blocare mai scurt**, ceea ce îi permite să fie cheltuită înainte ca angajamentele anterioare să devină valide arată fiecare tranzacție de angajament care stabilăște un temp-de-blocare mai scurt, permitând să fie cheltuită înainte ca angajamentele anterioare să devină valide.

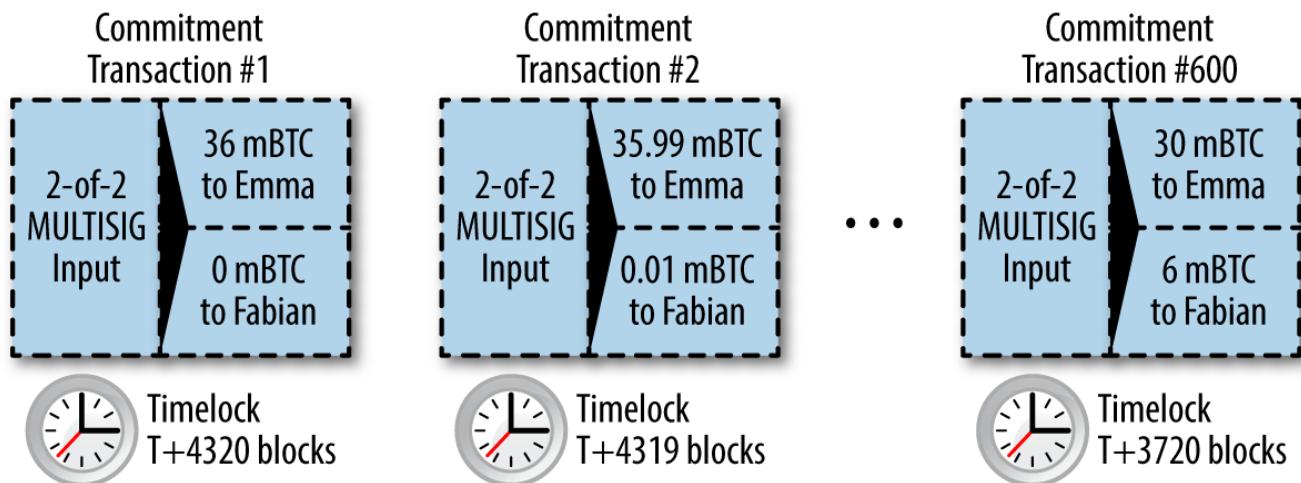


Figure 79. Fiecare angajament stabilăște un **temp-de-blocare mai scurt**, ceea ce îi permite să fie cheltuită înainte ca angajamentele anterioare să devină valide

Fiecare tranzacție de angajament ulterioară trebuie să aibă un temp-de-blocare mai scurt, astfel încât să poată fi difuzată (broadcast) înaintea predecesorilor săi și înaintea tranzacției de rambursare. Capacitatea de a difuza un angajament mai devreme asigură că va putea cheltui ieșirea de finanțare și va exclude ca orice altă tranzacție de angajament să fie răscumpărată prin cheltuirea ieșirii. Garanțiile oferite de lanțul-de-blocuri bitcoin (prevenind cheltuirea-dublă și impunând tempi-de-blocare) permit în mod efectiv fiecărei tranzacții de angajament să-și invalideze predecesorii.

Canalele cu stare folosesc tempi-de-blocare pentru a aplica contractele inteligente pe o anumită dimensiune. În acest exemplu am văzut cum dimensiunea timpului garantează că cea mai recentă tranzacție de angajament devine validă înainte de orice angajamente anterioare. Astfel, cea mai recentă tranzacție de angajament poate fi transmisă, cheltuind intrările și invalidând tranzacțiile anterioare de angajament. Executarea contractelor inteligente cu tempi-de-blocare absoluiți protejează împotriva înșelăciunii uneia dintre părți. Această implementare nu are nevoie decât de tempi-de-blocare absoluiți la nivel de tranzacție (**nLocktime**). În continuare, vom vedea cum pot fi utilizați tempi-de-blocare la nivel de script, **CHECKLOCKTIMEVERIFY** și **CHECKSEQUENCEVERIFY** pentru a construi canale de stare mai flexibile, utile și mai sofisticate.

Prima formă de canal de plată unidirectională a fost demonstrată ca o aplicație prototip de streaming video în 2015 de o echipă de dezvoltatori argentinieni.

Tempii-de-blocare nu sunt singura cale de a invalida tranzacțiile anterioare de angajament. În secțiunile următoare vom vedea cum poate fi folosită o cheie de revocare pentru a obține același rezultat. Tempii-de-blocare sunt eficienți, dar au două dezavantaje distințe. Stabilind un interval de timp maxim atunci când canalul este deschis pentru prima dată, acestea limitează durata de viață a canalului. Mai rău, tempii-de-blocare forțează implementările canalelor să ajungă la un

echilibru între permiterea canalelor cu durată lungă de viață și obligarea unuia dintre participanți să aștepte foarte mult timp pentru o rambursare în cazul închiderii premature. De exemplu, dacă permiteți ca un canal să rămână deschis timp de 30 de zile, setând termenul de restituire la 30 de zile, dacă una dintre părți dispare imediat, cealaltă parte trebuie să aștepte 30 de zile pentru o rambursare. Cu cât închiderea canalului este mai îndepărtată, cu atât restituirea este mai îndepărtată.

A doua problemă este că, deoarece fiecare tranzacție de angajament ulterioară trebuie să diminueze timpul-de-blocare, există o limită explicită a numărului de tranzacții de angajament care pot fi schimbate între părți. De exemplu, un canal de 30 de zile, care stabilește o un timp-de-blocare de 4320 de blocuri în viitor, poate găzdui doar 4320 de tranzacții de angajament intermediar înainte de a fi închis. Există un pericol în setarea timpului-de-blocare al tranzacției de angajamentului la 1 bloc. Prin setarea timpului-de-blocare dintre tranzacțiile de angajament la 1 bloc, un dezvoltator creează o povară foarte mare pentru participanții la canal, care trebuie să fie vigilenți, să rămână online să monitorizeze și să fie gata să transmită în orice moment tranzacția de angajament potrivită.

Acum că am înțeles cum pot fi utilizați timpii-de-blocare pentru a invalida angajamentele anterioare, putem vedea diferența dintre închiderea canalului în mod cooperativ și închiderea sa unilaterală prin difuzarea unei tranzacții de angajament. Toate tranzacțiile de angajament au timp-de-blocare, prin urmare, difuzarea unei tranzacții de angajament va implica întotdeauna așteptarea până la expirarea termenului. Însă, dacă cele două părți sunt de acord cu privire la soldul final și știu că ambele dețin tranzacții de angajament care vor face ca acel sold să devină realitate, ele pot construi o tranzacție de decontare (settlement) fără timp-de-blocare reprezentând același sold. Într-o încheiere de comun acord, oricare dintre părți ia cea mai recentă tranzacție de angajament și construiește o tranzacție de decontare care este identică în toate felurile, cu excepția faptului că omite timpul-de-blocare. Ambele părți pot semna această tranzacție de decontare știind că nu există nicio modalitate de a însela și de a obține un sold mai favorabil. Prin semnarea și transmiterea în mod cooperativ a tranzacției de decontare pot închide canalul și își pot răscumpăra soldul imediat. În cel mai rău caz, una dintre părți poate fi meschină, să refuze să coopereze și să forțeze cealaltă parte să facă o închidere unilaterală cu cea mai recentă tranzacție de angajament. Dar dacă fac asta, trebuie să aștepte și ca fondurile lor să fie deblocate.

## Angajamente Revocabile Asimetric

O modalitate mai bună de a gestiona stările angajamentelor anterioare este de a le revoca în mod explicit. Totuși, acest lucru nu este ușor de realizat. O caracteristică cheie a bitcoin este că, odată ce o tranzacție este validă, aceasta rămâne validă și nu expiră. Singura modalitate de a anula o tranzacție este de a cheltui dublu intrările sale cu o altă tranzacție înainte de a fi minată. De aceea, am folosit timpi-de-blocare în exemplul canalului de plată simplu de mai sus pentru a ne asigura că angajamentele mai recente ar putea fi cheltuite înainte ca angajamentele mai vechi să fie valide. Cu toate acestea, ordonarea angajamentelor în timp creează o serie de constrângeri care fac dificilă utilizarea canalelor de plată.

Chiar dacă o tranzacție nu poate fi anulată, ea poate fi construită astfel încât să fie indezirabilă de folosit. Modul în care facem acest lucru este oferind fiecărei părți o *cheie de revocare* care poate fi folosită pentru pedepsirea celeilalte părți dacă încearcă să trișeze. Acest mecanism de revocare a tranzacțiilor anterioare de angajament a fost propus pentru prima dată ca parte a Lightning

Network.

Pentru a explica cheile de revocare, vom construi un canal de plată mai complex între două burse-de-chimb administrate de Hitesh și Irene. Hitesh și Irene gestionează burse de bitcoin în India și, respectiv, în SUA. Clienții bursei din India trimit adesea plăți către clienții bursei din SUA și invers. În prezent, aceste tranzacții apar pe lanțul-de-blocuri bitcoin, dar asta înseamnă că trebuie să plătească comisioane și să aștepte mai multe blocuri pentru confirmare. Configurarea unui canal de plată între burse va reduce semnificativ costurile și va accelera fluxul tranzacțiilor.

Hitesh și Irene inițiază canalul prin construirea în colaborare a unei tranzacții de finanțare, fiecare finanțând canalul cu 5 bitcoin. Soldul inițial este de 5 bitcoin pentru Hitesh și 5 bitcoin pentru Irene. Tranzacția de finanțare blochează starea canalului într-o semnătură 2-din-2, la fel ca în exemplul unui canal simplu.

Tranzacția de finanțare poate avea una sau mai multe intrări de la Hitesh (adăugând până la 5 bitcoin sau mai mult) și una sau mai multe intrări de la Irene (adăugând până la 5 bitcoin sau mai mult). Intrările trebuie să depășească ușor capacitatea canalului pentru a acoperi comisioanele de tranzacție. Tranzacția are o ieșire care blochează un total de 10 bitcoin pe o adresă multisemnătură 2-din-2 controlată atât de Hitesh cât și de Irene. Tranzacția de finanțare poate avea, de asemenea, una sau mai multe ieșiri care returnează restul către Hitesh și Irene dacă contribuțiile lor au depășit contribuția prevăzută pentru canal. Aceasta este o singură tranzacție cu intrări oferite și semnate de două părți. Trebuie să fie construită în colaborare și semnată de fiecare parte înainte de a fi transmisă.

Acum, în loc să creeze o singură tranzacție de angajament pe care ambele părți o semnează, Hitesh și Irene creează două tranzacții de angajament diferite, care sunt *asimetrice*.

Hitesh are o tranzacție de angajament cu două ieșiri. Prima ieșire îi plătește *imediat* lui Irene cei 5 bitcoin care îi sunt datorați. Cea de-a doua ieșire îi plătește lui Hitesh 5 bitcoin care îi sunt datorați, dar numai după un timp-de-blocare de 1000 de blocuri. Ieșirile tranzacției arată astfel:

Input: 2-of-2 funding output, signed by Irene

Output 0 <5 bitcoin>:

<Irene's Public Key> CHECKSIG

Output 1:

<1000 blocks>

CHECKSEQUENCEVERIFY

DROP

<Hitesh's Public Key> CHECKSIG

Irene are o tranzacție de angajament diferită cu două ieșiri. Prima ieșire îi plătește *imediat* lui Hitesh cei 5 bitcoin care îi sunt datorați. Cea de-a doua ieșire îi plătește lui Irene cei 5 bitcoin care îi sunt datorați, dar numai după un timp-de-blocare de 1000 de blocuri. Tranzacția de angajament deținută de Irene (semnată de Hitesh) arată astfel:

Input: 2-of-2 funding output, signed by Hitesh

Output 0 <5 bitcoin>:

<Hitesh's Public Key> CHECKSIG

Output 1:

<1000 blocks>

CHECKSEQUENCEVERIFY

DROP

<Irene's Public Key> CHECKSIG

În acest fel, fiecare parte are o tranzacție de angajament, cheltuind cele 2-din-2 ieșiri de finanțare. Această intrare este semnată de către partea *cealaltă*. În orice moment, partea care deține tranzacția poate, de asemenea, să semneze (completând 2-din-2) și să o difuzeze. Cu toate acestea, dacă difuzează tranzacția de angajament, aceasta plătește celeilalte părți imediat, în timp ce trebuie să aștepte expirarea timpului-de-blocare. Impunând o întârziere la răscumpărarea uneia dintre ieșiri, punem fiecare parte într-un ușor dezavantaj atunci când aleg să difuzeze unilateral o tranzacție de angajament. Dar doar o întârziere nu este suficientă pentru a încuraja o conduită corectă.

**Două tranzacții de angajament asimetrice cu plata întârziată pentru partea care deține tranzacția** prezintă două tranzacții de angajament asimetrice, în care ieșirea care plătește titularul angajamentului este întârziată.

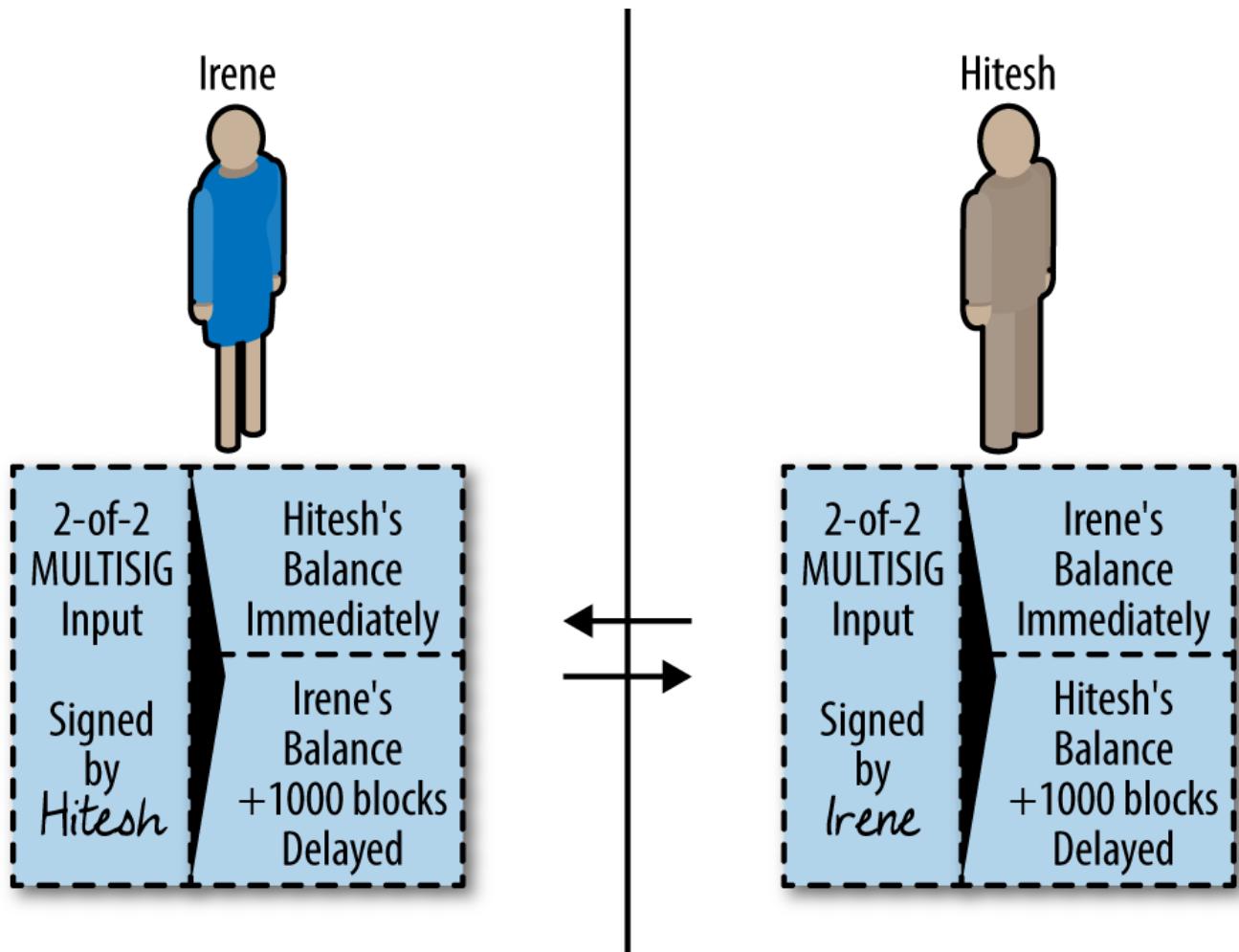


Figure 80. Două tranzacții de angajament asimetrice cu plata întârziată pentru partea care deține tranzacția

Acum introducem elementul final al acestei scheme: o cheie de revocare care împiedică un trișor să transmită un angajament expirat. Cheia de revocare permite părții nedreptățite să pedepsească trișorul prin luarea întregii sume a canalului.

Cheia de revocare (revocation key) este compusă din două secrete, fiecare jumătate generată independent de fiecare participant la canal. Este similar cu o multisemnătură 2-din-2, dar construită folosind aritmetică curbei eliptice, astfel încât ambele părți cunosc cheia publică de revocare, dar fiecare parte cunoaște doar jumătate din cheia secretă de revocare.

În fiecare rundă, ambele părți își dezvăluie jumătatea secretului de revocare celeilalte părți, oferind astfel celeilalte părți (care are acum ambele jumătăți) mijloacele de a solicita ieșirea de penalizare dacă această tranzacție revocată este transmisă vreodată.

Fiecare tranzacție de angajament are o ieșire "întârziată". Scriptul de răscumpărare pentru acea ieșire permite unei părți să o răscumpere după 1000 de blocuri, sau cealaltă parte să o răscumpere dacă are o cheie de revocare, penalizând transmiterea unui angajament revocat.

Așadar, atunci când Hitesh creează o tranzacție de angajament pentru ca Irene să o semneze, el face ca a doua ieșire să fie plătită lui însuși după 1000 de blocuri sau la cheia publică de revocare (despre care știe doar jumătate din secret). Hitesh construiește această tranzacție. El își va dezvăluji jumătatea secretului de revocare lui Irene doar atunci când va fi pregătit să tranzitioneze la o nouă stare a canalului și dorește să revoce acest angajament.

Scriptul celei de-a doua ieșiri arată astfel:

```
Output 0 <5 bitcoin>:  
  <Irene's Public Key> CHECKSIG  
  
Output 1 <5 bitcoin>:  
IF  
  # Revocation penalty output  
  <Revocation Public Key>  
ELSE  
  <1000 blocks>  
  CHECKSEQUENCEVERIFY  
  DROP  
  <Hitesh's Public Key>  
ENDIF  
CHECKSIG
```

Irene poate semna cu încredere această tranzacție, deoarece, dacă este transmisă, îi va plăti imediat ceea ce i se cuvine. Hitesh deține tranzacția, dar știe că, dacă o transmite într-o închidere de canal unilaterală, va trebui să aștepte 1000 de blocuri pentru a fi plătit.

Când canalul avansează la următoarea stare, Hitesh trebuie să *revoce* această tranzacție de angajament înainte ca Irene să fie de acord să semneze următoarea tranzacție de angajament. Pentru a face asta, tot ce trebuie să facă este să-i trimită jumătate lui din *cheia de revocare* lui Irene. După ce Irene are ambele jumătăți ale cheii secrete de revocare pentru acest angajament, ea poate semna cu încredere următorul angajament. Știe că, dacă Hitesh încearcă să înceleze publicând angajamentul anterior, ea poate utiliza cheia de revocare pentru a răscumpăra rezultatul întârziat al lui Hitesh. *Dacă Hitesh trășează, Irene obține cele două ieșiri.* Între timp, Hitesh are doar jumătatea secretului de revocare pentru acea cheie publică de revocare și nu poate răscumpăra ieșirea până ce trec 1000 de blocuri. Irene va putea răscumpăra ieșirea și să îl sanctioneze pe Hitesh înainte de trecerea celor 1000 de blocuri.

Protocolul de revocare este bilateral, ceea ce înseamnă că în fiecare rundă, pe măsură ce starea canalului avansează, cele două părți schimbă noi angajamente, schimbând secrete de revocare pentru angajamentele anterioare și își semnează reciproc noi tranzacții de angajament. Deoarece acceptă o nouă stare, ei fac imposibil de utilizat starea anterioară, oferindu-și reciproc secretele de revocare necesare pentru a sanctiona orice înșelăciune.

Să ne uităm la un exemplu de funcționare. Unul dintre clienții lui Irene vrea să trimită 2 bitcoin unuia dintre clienții lui Hitesh. Pentru a transmite 2 bitcoin pe canal, Hitesh și Irene trebuie să avanseze starea canalului pentru a reflecta noul sold. Ei se vor angaja într-o nouă stare (starea numărul 2), în care cei 10 bitcoin ai canalului sunt împărțiți, 7 bitcoin la Hitesh și 3 bitcoin la Irene. Pentru a avansa starea canalului, fiecare va crea noi tranzacții de angajament care reflectă noul bilanț al canalului.

Ca mai înainte, aceste tranzacții de angajament sunt asimetrice, astfel încât tranzacția de angajament pe care fiecare parte o deține o obligă să aștepte dacă o răscumpără. Foarte important, înainte de a semna noi tranzacții de angajament, trebuie să schimbe mai întâi cheile de revocare pentru a invalida angajamentul anterior. În acest caz particular, interesele lui Hitesh sunt

conforme cu starea reală a canalului și, prin urmare, el nu are motive să transmită o stare anterioară. Cu toate acestea, lui Irene, starea numărul 1 îi lasă un sold mai mare decât starea 2. Atunci când Irene îi oferă lui Hitesh cheia de revocare a tranzacției sale anterioare de angajament (starea numărul 1), ea își revocă efectiv capacitatea de a profita de regresarea canalului către o stare anterioară, deoarece cu cheia de revocare, Hitesh poate răscumpăra fără întârziere ambele ieșiri ale tranzacției de angajament anterior. Adică dacă Irene transmite starea anterioară, Hitesh își poate exercita dreptul de a lua toate ieșirile.

Important, revocarea nu se produce automat. În timp ce Hitesh are capacitatea de a o pedepsi pe Irene pentru înșelăciune, el trebuie să urmărească cu diligență lanțul-de-blocuri pentru a descoperi semne de înșelăciune. Dacă vede o tranzacție de angajament anterioară difuzată, el are 1000 de blocuri pentru a acționa și pentru a utiliza cheia de revocare pentru a contracara înșelăciunea lui Irene și a o pedepsi luând întregul sold, toți cei 10 bitcoin.

Angajamentele revocabile asimetrice cu timpi-de-blocare relativi (CSV) reprezintă o modalitate mult mai bună de a implementa canale de plată și o inovație foarte semnificativă în această tehnologie. Cu această construcție, canalul poate rămâne deschis la nesfârșit și poate avea miliarde de tranzacții cu angajamente intermediare. În implementarea prototipurilor din Lightning Network, starea de angajament este identificată printr-un indice de 48 de biți, care permite mai mult de 281 trilioane ( $2,8 \times 10^{14}$ ) tranzacții de stare în orice canal!

## Contracte cu Rezumat și Timp-de-Blocare (Hash Time Lock Contracts - HTLC)

Canalele de plată pot fi extinse în continuare cu un tip special de contract inteligent care permite participanților să angajeze fonduri (commit funds) într-un secret rambursabil, cu un termen de expirare. Această caracteristică se numește *Contract cu Timp-de-Blocare* (Hash Time Lock Contract), sau *HTLC* și este utilizată atât în canale de plată bidirectionale, cât și în cele rutate.

Să explicăm mai întâi partea de "rezumat" a HTLC. Pentru a crea un HTLC, destinatarul intenționat al plății va crea mai întâi un secret *R*. Apoi calculează rezumatul acestui secret *H*:

$$H = \text{Hash}(R)$$

Această funcție produce un rezumat *H* care poate fi inclus în scriptul de blocare al unei ieșiri. Cine știe secretul *R* poate folosi pentru a răscumpăra rezultatul. Secretul *R* este denumit și *preimagine* pentru funcția de rezumare. Preimaginea reprezintă doar datele care sunt utilizate ca intrare pentru o funcție de rezumare.

A doua parte a unui HTLC este componenta "timp-de-blocare". Dacă secretul nu este dezvăluit, plăitorul HTLC poate primi o "rambursare" după ceva timp. Acest lucru se realizează cu un timp-de-blocare absolut folosind **CHECKLOCKTIMEVERIFY**.

Scriptul care implementează un HTLC ar putea arăta astfel:

```

IF
  # Payment if you have the secret R
  HASH160 <H> EQUALVERIFY
ELSE
  # Refund after timeout.
  <locktime> CHECKLOCKTIMEVERIFY DROP
  <Payer Public Key> CHECKSIG
ENDIF

```

Oricine cunoaște secretul  $R$ , care atunci când este rezumat este egal cu  $H$ , poate răscumpăra această ieșire exercitând prima clauză **IF**.

Dacă secretul nu este dezvăluit și HTLC-ul revendicat, după un anumit număr de blocuri, plătitorul poate solicita o rambursare folosind a doua clauză din **IF**.

Aceasta este o implementare de bază a unui HTLC. Acest tip de HTLC poate fi răscumpărat de către *oricine* are secretul  $R$ . Un HTLC poate lua multe forme diferite, cu ușoare variații asupra scriptului. De exemplu, adăugarea unui operator **CHECKSIG** și o cheie publică în prima clauză restricționează răscumpărarea la un destinatar anume, care trebuie să cunoască și secretul  $R$ .

## Canale de Plată Rutate (Lightning Network)

Lightning Network este o rețea de rutare de canale de plată bidirectionale conectate de-la-un-capăt-la-altul. O rețea ca aceasta poate permite oricărui participant să direcționeze o plată de la un canal la un alt canal fără să aibă încredere în niciunul dintre intermediari. Lightning Network a fost descrisă prima dată de [Joseph Poon și Thadeus Dryja în februarie 2015](#), bazându-se pe conceptul de canale de plată așa cum a fost propus și elaborat de mulți alții.

"Lightning Network" se referă la o proiectare specifică pentru o rețea de canale de plată rutate, care a fost implementată până acum de cel puțin cinci echipe open source diferite. Implementările independente sunt coordonate de un set de standarde de interoperabilitate descrise în lucrarea [Basics of Lightning Technology \(BOLT\) paper](#).

Implementări prototip pentru Lightning Network au fost lansate de mai multe echipe.

Lightning Network este unul din modurile posibile de implementare a canalelor de plată rutate. Există câteva alte modele care urmăresc atingerea unor scopuri similare, cum ar fi Teechan și Tumblebit.

### Exemplu Lightning Network Simplu

Să vedem cum funcționează.

În acest exemplu, avem cinci participanți: Alice, Bob, Carol, Diana și Eric. Acești cinci participanți și-au deschis canale de plată între ei, în perechi. Alice are un canal de plată cu Bob. Bob este conectat la Carol, Carol la Diana, iar Diana la Eric. Pentru simplitate, să presupunem că fiecare canal este finanțat cu 2 bitcoin de fiecare participant, pentru o capacitate totală de 4 bitcoin în fiecare canal.

O serie de canale de plată bidirectionale legate pentru a forma o rețea Lightning care poate direcționa o plată de la Alice la Eric arată cinci participanți la o rețea Lightning, conectați prin canale de plată bidirectionale care pot fi legate pentru a efectua o plată de la Alice la Eric (Canale de Plată Rutate (Lightning Network)).

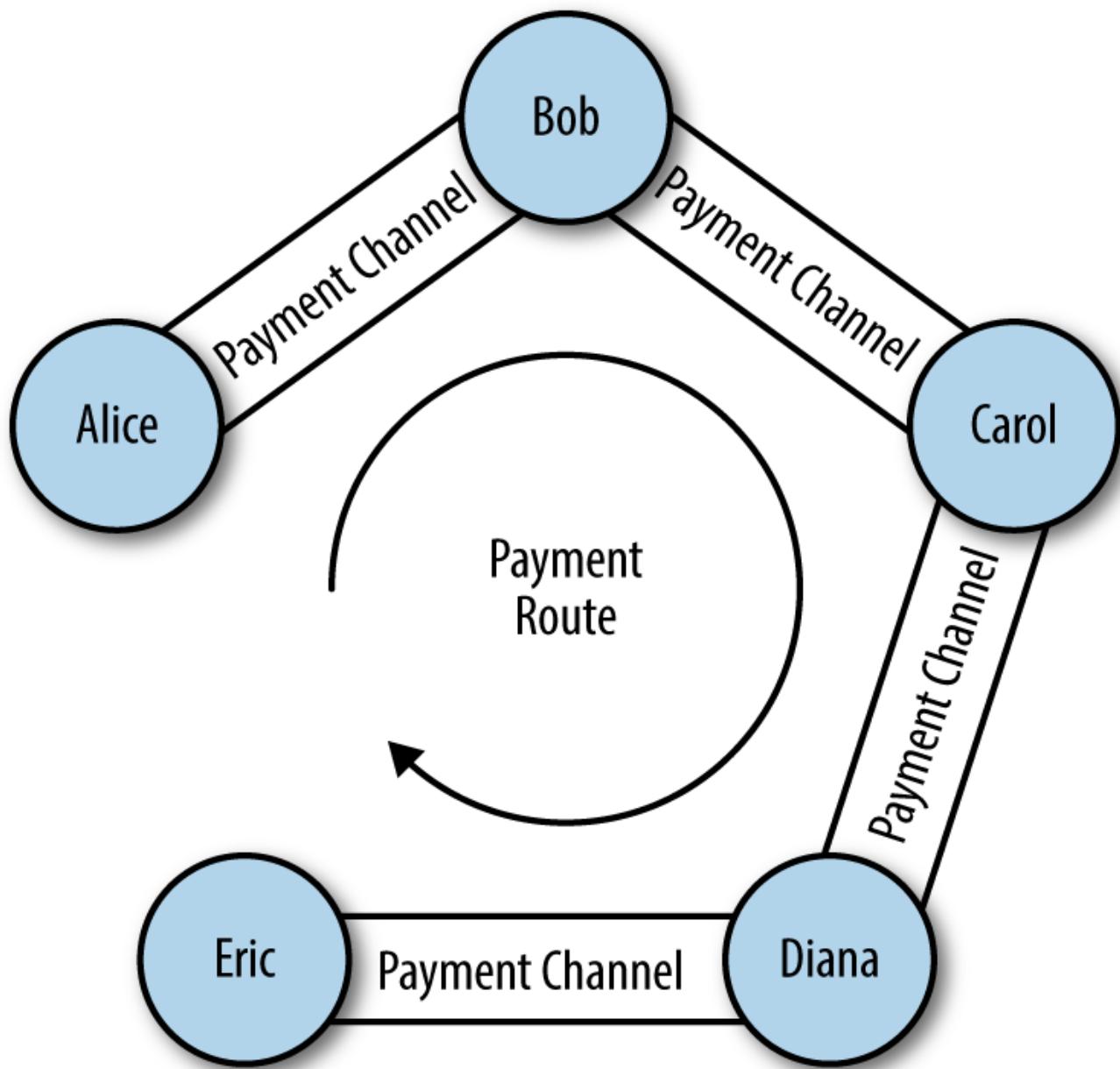


Figure 81. O serie de canale de plată bidirectionale legate pentru a forma o rețea Lightning care poate direcționa o plată de la Alice la Eric

Alice vrea să îi plătească lui Eric 1 bitcoin. Cu toate acestea, Alice nu este conectată la Eric printr-un canal de plată. Crearea unui canal de plată necesită o tranzacție de finanțare, care trebuie angajată (committed) în lanțul-de-blocuri bitcoin. Alice nu vrea să deschidă un nou canal de plată și să aloce mai multe fonduri. Există o modalitate de a-l plăti pe Eric, indirect?

**Rutarea pas-cu-pas printr-o rețea Lightning** prezintă procesul pas cu pas de direcționare a unei plăți de la Alice către Eric, printr-o serie de angajamente HTLC pe canalele de plată care leagă participanții.

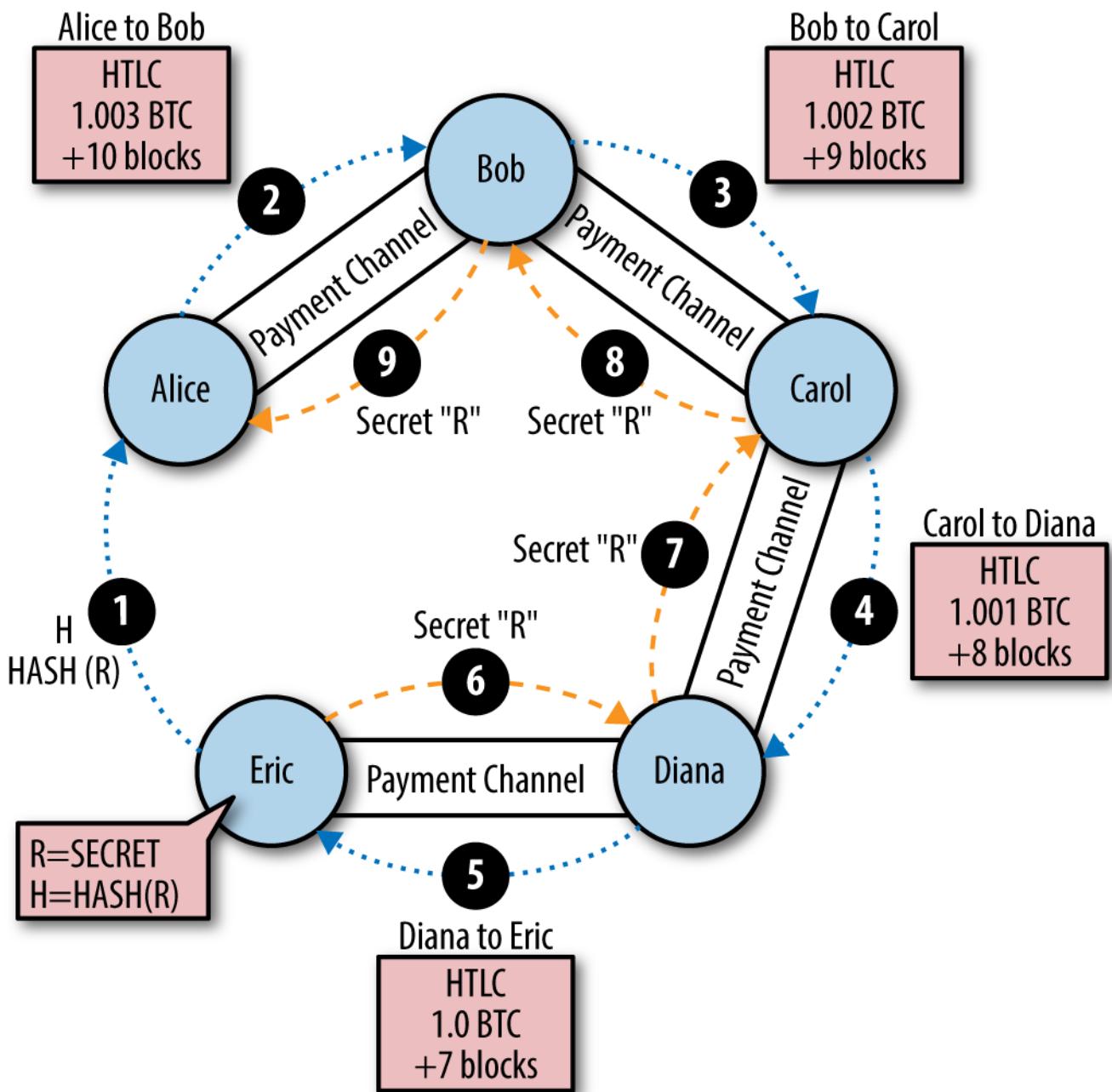


Figure 82. Rutarea pas-cu-pas printr-o rețea Lightning

Alice rulează un nod Lightning Network (LN) care menține canalul de plată cu Bob și are capacitatea de a descoperi rute între canalele de plată. Nodul LN al lui Alice are, de asemenea, capacitatea de a se conecta prin internet la nodul LN al lui Eric. Nodul LN al lui Eric creează un secret  $R$  folosind un generator de numere aleatoare. Nodul lui Eric nu dezvăluie nimănui acest secret. În schimb, nodul lui Eric calculează un rezumat  $H$  al secretului  $R$  și transmite acest rezumat către nodul lui Alice (vezi [Rutarea pas-cu-pas printr-o rețea Lightning](#) pasul 1).

Acum, nodul LN al lui Alice construiește o rută între nodul LN lui Alice și nodul LN al lui Eric. Algoritmul de rutare utilizat va fi examinat mai în detaliu mai târziu, dar deocamdată să presupunem că nodul lui Alice poate găsi o rută eficientă.

Nodul lui Alice construiește apoi un HTLC, plătibil rezumatului  $H$ , cu un interval de timp de restituire de 10 blocuri (bloc curent + 10), pentru o sumă de 1,003 bitcoin (a se vedea [Rutarea pas-cu-pas printr-o rețea Lightning](#) pasul 2). Cei 0,003 suplimentari vor fi utilizati pentru a compensa nodurile intermediare pentru participarea lor la această rută de plată. Alice oferă acest HTLC lui

Bob, deducând 1,003 bitcoin din soldul canalului său cu Bob și alocându-l la HTLC. HTLC-ul are următoarea semnificație: *"Alice alocă 1,003 din soldul canalului său care trebuie plătit lui Bob dacă Bob cunoaște secretul sau restituie soldul lui Alice dacă trec 10 blocuri."* Bilanțul canalului dintre Alice și Bob este acum exprimat prin tranzacții de angajament cu trei ieșiri: 2 bitcoin pentru Bob, 0,997 sold bitcoin pentru Alice, 1,003 bitcoin alocăți în HTLC-ul lui Alice. Soldul lui Alice este redus cu suma alocată în HTLC.

Bob are acum un angajament că, dacă este capabil să obțină secretul  $R$  în următoarele 10 blocuri, poate solicita cei 1,003 blocați de Alice. Cu acest angajament în mână, nodul lui Bob construiește un HTLC pe canalul său de plată cu Carol. HTLC-ul lui Bob alocă 1,002 bitcoin pentru rezumatul  $H$  pentru 9 blocuri, pe care Carol le poate răscumpăra dacă obține secretul  $R$  (vezi [Rutarea pas-cu-pas printr-o rețea Lightning](#) pasul 3). Bob știe că, dacă Carol își poate revendica HTLC-ul, ea trebuie să obțină  $R$ . Dacă Bob are  $R$  în nouă blocuri, îl poate folosi pentru a solicita HTLC-ul lui Alice. De asemenea, el câștigă 0,001 bitcoin pentru alocarea soldului canalului său pentru nouă blocuri. Dacă Carol nu poate revendica HTLC-ul lui, iar Bob nu poate revendica HTLC-ul lui Alice, totul revine la soldurile anterioare ale canalului și nimici nu este în pierdere. Bilanțul canalului dintre Bob și Carol este acum: 2 la Carol, 0,998 la Bob, 1,002 alocat de Bob în HTLC.

Carol are acum un angajament că, dacă primește  $R$  în următoarele nouă blocuri, poate solicita 1,002 bitcoin blocați de Bob. Acum ea poate face un angajament HTLC pe canalul ei cu Diana. Ea alocă un HTLC de 1,001 bitcoin pentru rezumatul  $H$ , pentru opt blocuri, pe care Diana îi poate răscumpăra dacă are secretul  $R$  (vezi [Rutarea pas-cu-pas printr-o rețea Lightning](#) pasul 4). Din perspectiva lui Carol, dacă aceasta funcționează, ea câștigă 0,001 bitcoin iar dacă nu, nu pierde nimic. HTLC-ul ei către Diana este viabil numai dacă  $R$  este dezvăluit, moment în care ea poate solicita HTLC de la Bob. Bilanțul canalului dintre Carol și Diana este acum: 2 la Diana, 0,999 la Carol, 1,001 alocăți de Carol în HTLC.

În cele din urmă, Diana poate oferi un HTLC către Eric, angajând 1 bitcoin către rezumatul  $H$  pentru șapte blocuri (vezi [Rutarea pas-cu-pas printr-o rețea Lightning](#) pasul 5). Bilanțul canalului dintre Diana și Eric este acum: 2 la Eric, 1 la Diana, 1 angajat de Diana în HTLC.

Cu toate acestea, la acest hop în rută, Eric are secretul  $R$ . Prin urmare, el poate revendica HTLC-ul oferit de Diana. El trimite  $R$  către Diana și solicită 1 bitcoin, adăugându-l la soldul canalului său (vezi [Rutarea pas-cu-pas printr-o rețea Lightning](#) pasul 6). Bilanțul canalului este acum: 1 la Diana, 3 la Eric.

Acum, Diana are secretul  $R$ . Prin urmare, acum poate solicita HTLC-ul de la Carol. Diana transmite  $R$  lui Carol și adaugă 1,001 bitcoin la soldul canalului său (vezi [Rutarea pas-cu-pas printr-o rețea Lightning](#) pasul 7). Acum, soldul canalului dintre Carol și Diana este: 0,999 la Carol, 3,001 la Diana. Diana a "câștigat" 0,001 pentru participarea la această rută de plată.

În sens invers pe rută, secretul  $R$  permite fiecărui participant să revendice HTLC-urile restante. Carol cere 1,002 de la Bob, modificând soldul pe canalul lor la: 0,998 la Bob, 3,002 la Carol (vezi [Rutarea pas-cu-pas printr-o rețea Lightning](#) pasul 8). În cele din urmă, Bob revendică HTLC de la Alice (vezi [Rutarea pas-cu-pas printr-o rețea Lightning](#) pasul 9). Soldul canalului lor este actualizat la: 0,997 la Alice, 3,003 la Bob.

Alice a plătit lui Eric 1 bitcoin fără a deschide un canal către Eric. Niciuna dintre părțile intermediare din ruta de plată nu a trebuit să aibă încredere una în celaltă. Pentru angajamentul

pe termen scurt al fondurilor lor în canal, aceștia reușesc să câștige un mic comision, singurul risc fiind o mică întârziere în rambursare în cazul în care canalul a fost închis sau plata rutată a eșuat.

## Transport și Rutare în Lightning Network

Toate comunicațiile dintre nodurile LN sunt criptate punct-la-punct. În plus, nodurile au o cheie publică pe termen lung pe care o folosesc ca identificator și pentru a se autentifica reciproc.

Ori de câte ori un nod dorește să trimită o plată către un alt nod, trebuie mai întâi să construiască o *cale* prin rețea prin conectarea canalelor de plată cu o capacitate suficientă. Nodurile fac cunoscute informațiile lor de rutare, inclusiv canalele deschise, capacitatea pe care o are fiecare canal și ce comisione percep pentru plățile de rutare. Informațiile de rutare pot fi împărtășite într-o varietate de moduri și este posibil să apară diferite protocoale de rutare pe măsură ce tehnologia Lightning Network avansează. Unele implementări ale rețelei Lightning folosesc protocolul IRC ca un mecanism convenabil pentru noduri pentru anunțarea informațiilor de rutare. O altă implementare de descoperire a rutelor folosește un model P2P (de-la-egal-la-egal) în care nodurile propagă anunțuri despre canal către semenii lor, într-un model de "inundare", similar cu modul în care bitcoin propagă tranzacțiile. Planurile de viitor includ o propunere numită [Flare](#), care este un model de rutare hibrid cu noduri locale "cartiere" și noduri "far" cu rază mai lungă.

În exemplul nostru anterior, nodul lui Alice folosește unul dintre aceste mecanisme de descoperire a rutei pentru a găsi una sau mai multe căi care leagă nodul ei la nodul lui Eric. După ce nodul lui Alice a construit o cale, ea va inițializa acea cale prin rețea, prin propagarea unei serii de instrucțiuni criptate și imbricate pentru a conecta fiecare dintre canalele de plată adiacente.

Important este că această cale este cunoscută doar de nodul lui Alice. Toți ceilalți participanți la ruta de plată văd numai nodurile adiacente. Din perspectiva lui Carol, această plată arată ca o plată de la Bob la Diana. Carol nu știe că Bob transmite de fapt o plată de la Alice. De asemenea, nu știe că Diana va transmite o plată către Eric.

Aceasta este o caracteristică critică a Lightning Network, deoarece asigură confidențialitatea plăților și face foarte dificilă aplicarea supravegherii, cenzurii sau listelor negre. Dar cum stabilește Alice această cale de plată, fără să dezvăluie nimic nodurilor intermediare?

Lightning Network implementează un protocol de rutare (onion-routed) bazat pe o schemă numită [Sphinx](#). Acest protocol de rutare asigură că un expeditor de plată poate construi și comunica o cale prin Lightning Network, astfel încât:

- Nodurile intermediare pot verifica și decripta porțiunea lor de informații despre rută și pot găsi următorul hop.
- Cu excepția hopurilor anterior și următor, nu pot afla despre alte noduri care fac parte din traseu.
- Nu pot identifica lungimea căii de plată sau poziția proprie din calea respectivă.
- Fiecare parte a căii este criptată astfel încât un atacator la nivel de rețea nu poate asocia între ele pachetele din diferite părți ale căii.
- Spre deosebire de Tor (un protocol de rutare cu anonimizare pe internet), nu există "noduri de ieșire" care să poată fi puse sub supraveghere. Plățile nu trebuie transmise către lanțul-de-blocuri bitcoin; nodurile actualizează doar soldurile canalelor.

Utilizând acest protocol de rutare (onion-routed), Alice ambalează (wraps) fiecare element al căii într-un strat de criptare, începând de la sfârșit și mergând înapoi. Ea criptează un mesaj către Eric cu cheia publică a lui Eric. Acest mesaj este ambalat într-un mesaj criptat către Diana, identificându-l pe Eric drept următorul destinatar. Mesajul către Diana este ambalat într-un mesaj criptat cu cheia publică a lui Carol și identificând-o pe Diana drept următorul destinatar. Mesajul către Carol este criptat pentru cheia lui Bob. Astfel, Alice a construit această "ceapă" (onion) de mesaje multistrat criptată. Ea trimită "ceapa de mesaje" lui Bob, care poate decripta și desface doar stratul exterior. În interior, Bob găsește un mesaj adresat lui Carol pe care îl poate transmite lui Carol, dar pe care nu îl poate decifra. Urmând calea, mesajele sunt redirecționate, decriptate, redirecționate etc., până la Eric. Fiecare participant cunoaște doar nodul anterior și următor din fiecare hop.

Fiecare element al căii conține informații despre HTLC-ul care trebuie extins până la următorul hop, suma care este trimisă, comisionul care este inclus și expirarea CLTV locktime (în blocuri) a HTLC. Pe măsură ce informațiile de rută se propagă, nodurile își asumă angajamente HTLC către următorul hop.

În acest moment, s-ar putea să vă întrebați cum este posibil ca nodurile să nu cunoască lungimea căii și poziția lor în acea cale. La urma urmei, primesc un mesaj și îl transmit către următorul hop. Nu se face mai scurt, permăndu-le să deducă dimensiunea căii și poziția lor? Pentru a preveni acest lucru, calea este întotdeauna fixată la 20 de hopuri și umplută cu date aleatorii. Fiecare nod vede următorul salt și un mesaj criptat de lungime fixă de redirecționat. Doar destinatarul final vede că nu există următorul hop. Pentru toți ceilalți, pare fi mereu încă 20 de hopuri.

## Beneficiile Lightning Network

O rețea Lightning este o tehnologie de rutare din stratul al doilea (second layer). Poate fi aplicată oricărui lanț-de-blocuri care acceptă unele funcții de bază, cum ar fi tranzacții multisemnătură, timpi-de-blocare și contracte inteligente simple.

Dacă o rețea Lightning este stivuită deasupra rețelei bitcoin, rețeaua bitcoin poate obține o creștere semnificativă a capacitatei, a confidențialității, a granularității și a vitezei, nesacrificând principiile de funcționare fără încredere în intermediari:

### Confidențialitate

Plățile Lightning Network sunt mult mai private decât plățile pe lanțul-de-blocuri bitcoin, întrucât nu sunt publice. În timp ce participanții la o rută pot vedea propagarea plășilor pe canalele lor, ei nu cunosc expeditorul sau destinatarul.

### Fungibilitate

O rețea Lightning face mult mai dificilă aplicarea supravegherii și listelor negre pe bitcoin, crescând fungibilitatea monedei (monedele pot fi ușor interschimbabile).

### Viteza

Tranzacțiile bitcoin care folosesc Lightning Network sunt decontate în milisecunde, în loc de minute, deoarece HTLC-urile sunt aprobată fără a adăuga tranzacții la un bloc.

### Granularitate

O rețea Lightning poate permite plășii cel puțin la fel de mici ca limita "prafului" bitcoin, poate

chiar mai mici. Unele propuneri permit unități de subsatoshi.

## Capacitate

O rețea Lightning crește capacitatea sistemului bitcoin cu mai multe ordine de mărime. Nu există o valoare practică superioară a numărului de plăți pe secundă care să poată fi rutată printr-o rețea Lightning, deoarece depinde doar de capacitatea și de viteza fiecărui nod.

## Funcționare Fără Încredere (Trustless)

O rețea Lightning folosește tranzacții bitcoin între noduri care funcționează ca semeni, fără a avea încredere reciprocă. Astfel, o rețea Lightning păstrează principiile sistemului bitcoin, extinzând semnificativ parametrii de funcționare.

Desigur, după cum am menționat anterior, protocolul Lightning Network nu este singura modalitate de a implementa canale de plată rutate. Alte sisteme propuse includ Tumblebit și Teechan. În acest moment, însă, Lightning Network a fost deja implementată pe testnet. Mai multe echipe diferite au dezvoltat implementări concurente ale LN și lucrează la un standard comun de interoperabilitate (numit BOLT). Este probabil ca Lightning Network să fie prima rețea de canale de plată rutate care să fie implementată în producție.

## Concluzie

Am examinat doar câteva dintre aplicațiile emergente care pot fi construite folosind lanțul-de-blocuri bitcoin ca platformă de încredere. Aceste aplicații extind domeniul de aplicare bitcoin dincolo de plăți și dincolo de instrumentele financiare, pentru a cuprinde multe alte aplicații în care încrederea este critică. Prin descentralizarea bazei de încredere, lanțul-de-blocuri bitcoin este o platformă care va genera multe aplicații revoluționare într-o mare varietate de industrii.

# Appendix A: Referatul Bitcoin de Satoshi Nakamoto

### NOTE

Acesta este referatul original, reprobus în întregime exact cum a fost publicat de Satoshi Nakamoto în octombrie 2008.

## Bitcoin - Un Sistem Electronic de Numerar De-la-Egal-la-Egal

Satoshi Nakamoto

*satoshi@gmx.com*

[www.bitcoin.org](http://www.bitcoin.org)

**Abstract.** O versiune exclusiv de-la-egal-la-egal de numerar electronic ar permite trimitera plăților direct de la o entitate la alta fără a trece printr-o instituție financiară. Semnăturile digitale oferă o parte din soluție, dar principalele beneficii sunt pierdute dacă o a treia entitate considerată

de încredere este încă necesară pentru a preveni cheltuirea dublă. Propunem o soluție la problema dublei-cheltuiiri folosind o rețea de-la-egal-la-egal. Rețeaua pune tranzacțiilor o marcă de timp (timestamp) adăugând rezumatul (hash-ul) lor într-un lanț continuu de rezumate (hash-uri) bazate pe dovada-de-lucru, formând un registru ce nu poate fi schimbat fără a refațe dovada-de-lucru. Cel mai lung lanț servește nu doar ca dovadă a secvenței de evenimente observate, dar și ca dovadă că a fost creat folosind cea mai mare cantitate de putere de calcul (CPU power). Cât timp majoritatea puterii de calcul este controlată de către noduri care nu cooperează pentru a ataca rețeaua, aceste noduri vor genera cel mai lung lanț și vor surclasă atacatorii. Rețeaua necesită o structură minimă. Fiecare nod difuzează (broadcasts) mesajele cât poate de bine, iar nodurile pot părași sau se pot (re)alătura rețelei după cum doresc, acceptând cel mai lung lanț bazat pe dovadă-de-lucru ca mărturie la ce s-a întâmplat cât timp au fost absente.

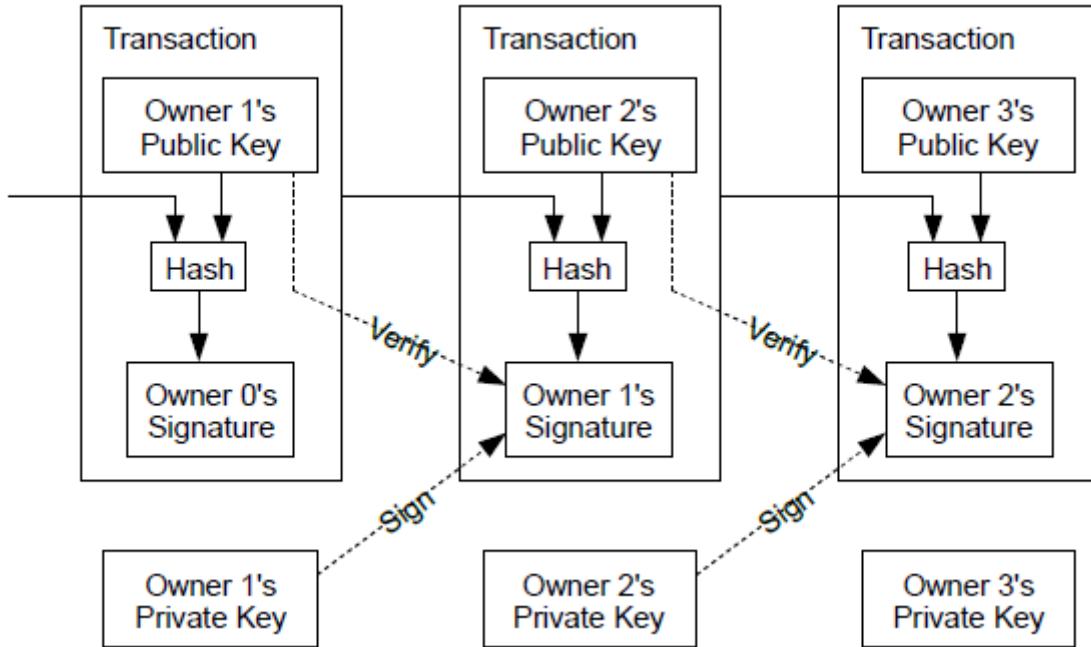
## Introducere

Comerțul pe Internet a ajuns să se bazeze aproape exclusiv pe instituții financiare care joacă rolul de entități terțe de încredere pentru a procesa plățile electronice. Chiar dacă sistemul funcționează destul de bine pentru majoritatea tranzacțiilor, acesta încă suferă de slăbiciunile intrinseci ale modelului bazat pe încredere. Tranzacții complet ireversibile nu sunt cu adevărat posibile, deoarece instituțiile financiare nu pot evita medierea disputelor. Costul de mediere crește costurile tranzacției, limitând dimensiunea minimă a tranzacției și reducând posibilitatea pentru tranzacții ocazionale mici, existând un cost mai extins în pierderea capacitatei de a efectua plăți ireversibile pentru servicii ireversibile. Având posibilitatea de anulare, nevoia de încredere se extinde. Comerțanții trebuie să fie precauți cu clienții lor, cerându-le mai multe informații decât ar avea nevoie în mod normal. Un anumit procent de fraudă este acceptat ca inevitabil. Aceste costuri și încercătudini legate de plată pot fi evitate efectuând plăți în persoană cu bani fizici, dar nu există un mecanism pentru a face plăți folosind un canal de comunicare fără implicarea unei entități de încredere.

Ceea ce este necesar, este un sistem electronic de plată care să fie bazat pe dovadă criptografică în loc să fie bazat pe încredere (într-un terț), care să permită oricărui două entități să tranzacționeze direct între ele fără a fi nevoie de o a treia entitate (considerată) de încredere. Tranzacții care sunt nepractic computațional să fie inversate (anulate) ar proteja vânzătorii în fața fraudelor, iar mecanismele obișnuite de conturi de garanție ar putea fi ușor implementate pentru a proteja cumpărătorii. În această lucrare, propunem o soluție la problema dublei-cheltuiiri folosind un server distribuit de-la-egal-la-egal pentru a genera dovada computațională a ordinii cronologice a tranzacțiilor. Sistemul este securizat atât timp cât nodurile oneste controlează împreună mai multă putere de calcul (CPU power) decât orice grup cooperant de noduri atacatoare.

## Tranzacții

Definim o monedă electronică ca fiind un lanț de semnături digitale. Fiecare proprietar transferă moneda următorului proprietar aplicând o semnătură digitală la rezumatul (hash-ul) tranzacției precedente și cheii publice a următorului proprietar și adăugându-le pe acestea la sfârșitul monedei. Un beneficiar poate verifica semnăturile pentru a verifica lanțul de proprietate.

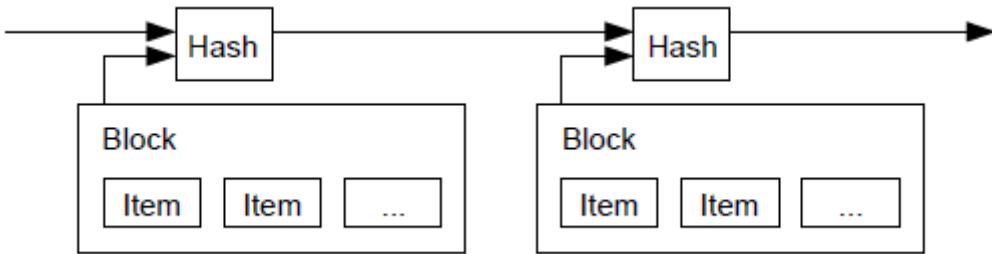


Problema este desigur că beneficiarul nu poate verifica dacă unul dintre proprietari a cheltuit de două ori moneda. O soluție obișnuită este folosirea unei autorități centrale, sau emitent, care inspectează fiecare tranzacție pentru cheltuiuri duble. După fiecare tranzacție, moneda trebuie returnată la emitent pentru a emite o nouă monedă, și doar monedele emise direct de către emitent sunt considerate sigure și că nu au fost cheltuite de două ori. Problema cu această soluție este că soarta întregului sistem monetar depinde de compania care rulează emitentul, fiecare tranzacție trebuind să treacă prin el, exact ca o bancă.

Avem nevoie de o modalitate prin care beneficiarul să știe că proprietarii precedenții nu au semnat vreo tranzacție anterioară. Pentru scopurile noastre, cea mai recentă tranzacție este cea care contează, deci nu ne pasă de încercările ulterioare de a cheltui a doua oară. Singurul mod de a confirma absența unei tranzacții este de a fi la curent cu toate tranzacțiile. În modelul bazat pe emitent, emitentul a fost la curent cu toate tranzacțiile și a decis care a ajuns prima. Pentru a realiza acest lucru fără o entitate de încredere, tranzacțiile trebuie să fie anunțate public [1], și avem nevoie de un sistem pentru ca participanții să cadă de acord asupra unui singur istoric al ordinii în care tranzacțiile au ajuns. Beneficiarul are nevoie de dovada că la momentul fiecărei tranzacții, majoritatea nodurilor au fost de acord că a fost prima primită.

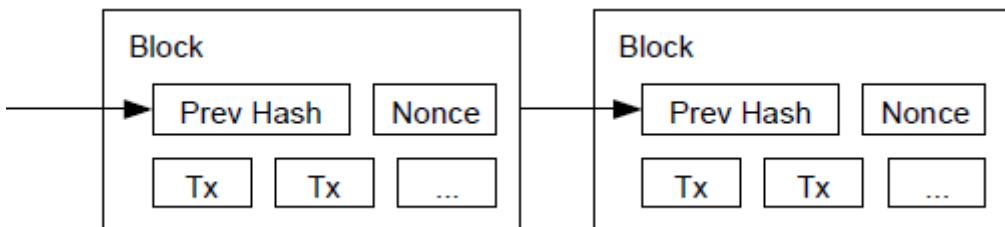
## Server de Marcare a Timpului (timestamp)

Soluția pe care o propunem începe cu un server de marcare a timpului (timestamp). Un server de marcare a timpului funcționează luând rezumatul (hash-ul) unui bloc de elemente ce urmează să fie marcate și publicând pe scară largă rezumatul, la fel cum procedează un ziar sau un post Usenet [2-5]. Marca-de-timp dovedește că datele trebuie să fi existat la momentul respectiv, evident, pentru a putea fi incluse în rezumat (hash). Fiecare marcă-de-timp include marca precedentă în rezumatul ei, formând un lanț, fiecare marcă-de-timp adițională consolidându-le pe cele dinaintea ei.



## Dovadă-de-Lucru

Pentru a implementa un server de marcare a timpului într-o rețea de-la-egal-la-egal, va trebui să folosim mai degrabă un sistem de dovadă-de-lucru similar cu soluția Hashcash [5] propusă de Adam Back, decât publicarea într-un ziar sau posturi Usenet. Soluția dovadă-de-lucru implică căutarea unei valori care atunci când este rezumată (hashed), cum ar fi cu o funcție SHA-256, rezumatul începe cu un anumit număr de biți de zero. Cantitatea de lucru medie necesară crește exponențial cu numărul de biți de zero necesari și poate fi verificată calculând un singur rezumat. Pentru rețeaua noastră de marcare a timpului, implementăm dovadă-de-lucru incrementând un câmp (nonce) în bloc până când o valoare este găsită (pentru nonce) care face ca rezumatul blocului să înceapă cu numărul necesar de biți de zero. Odată ce efortul CPU a fost investit pentru a face blocul să satisfacă dovadă-de-lucru, acest bloc nu poate fi schimbat fără a reface calculele. Pentru că blocurile ulterioare sunt înlăntuite după el, efortul necesar pentru a schimba blocul implică refacerea tututor blocurilor de după el.



Dovadă-de-lucru rezolvă de asemenea problema reprezentării majorității în luarea deciziilor. Dacă majoritatea ar fi fost bazată pe principiul o-adresă-IP-un-vot, atunci ar fi fost răsturnată de oricine ar fi fost în stare să aloce mai multe adrese IP. Dovadă-de-lucru este în esență un-CPU-un-vot. Decizia majorității este reprezentată de lanțul cel mai lung, care are cea mai mare dovadă-de-lucru acumulată. Dacă majoritatea puterii de calcul este controlată de către noduri oneste, lanțul onest va crește cel mai rapid și va depăși orice lanț concurrent. Pentru a modifica un bloc precedent, un atacator ar trebui să refacă dovadă-de-lucru a blocului respectiv, dar și a tuturor blocurilor de după el, iar apoi să ajungă din urmă și să depășească nodurile oneste. Vom arăta mai târziu că probabilitatea ca un atacator mai lent să ajungă din urmă (nodurile oneste) scade exponențial pe măsură ce noi blocuri sunt adăugate (la lanțul onest).

Pentru a compensa pentru creșterea performanței hardware-ului și a diverselor motive pentru a rula noduri de-a lungul timpului, dificultatea dovezii-de-lucru este determinată de o medie dinamică raportată la un număr mediu de blocuri pe oră. Dacă sunt generate prea repede, dificultatea crește.

## Rețea

Pașii pentru a rula o rețea sunt următorii:

1. Tranzacțiile noi sunt difuzate (broadcast) către toate nodurile.
2. Fiecare nod colectează tranzacții noi într-un bloc
3. Fiecare nod lucrează pentru a găsi o dovardă-de-lucru suficient de dificilă pentru blocul lui
4. Când un nod găsește o dovardă-de-lucru, difuzează (broadcasts) blocul către toate nodurile
5. Nodurile acceptă blocul doar dacă toate tranzacțiile conținute în el sunt valide și nu sunt deja cheltuite.
6. Nodurile își exprimă acordul asupra blocului începând lucrul la creeare următorului bloc din lanț, folosind rezumatul (hash-ul) blocului acceptat ca rezumat anterior.

Nodurile întotdeauna consideră lanțul cel mai lung ca fiind cel corect și vor continua să lucreze la extinderea acestuia. Dacă două noduri difuzează (broadcast) simultan două versiuni (valide) diferite ale următorului bloc, unele noduri vor primi una dintre versiuni întâi. În acest caz, vor lucra cu prima pe care au primit-o, dar vor salva și cealaltă ramură în caz că devine mai lungă. Egalitatea va fi întreruptă când următoarea dovardă-de-lucru este găsită și una din ramuri devine mai lungă; nodurile care lucrau la cealaltă ramură vor schimba către cea mai lungă.

Nu este nevoie ca tranzacțiile nou difuzate să ajungă la toate nodurile. Cât timp reușesc să ajungă la multe noduri, tranzacțiile vor reuși să fie incluse într-un bloc în scurtă vreme. Difuzarea de blocuri este de asemenea tolerantă cu mesajele pierdute. Dacă un nod nu primește un bloc, îl va cere atunci când primește următorul bloc și își dă seama că îi lipsește unul.

## Stimulente

Prin convenție, prima tranzacție dintr-un bloc este o tranzacție specială care creează monedă nouă deținută de către creatorul blocului. Această tranzacție adaugă un stimulent nodurilor pentru a susține rețeaua și oferă o modalitate de distribuire inițială a monedelor în circulație, din moment ce nu există o autoritate centrală care să le emită. Adăugarea stabilă a unei cantități constantă de noi monede este similară minerilor care cheltuiesc resurse pentru a pune aur în circulație. În cazul nostru este vorba de puterea de calcul și electricitatea care sunt cheltuite.

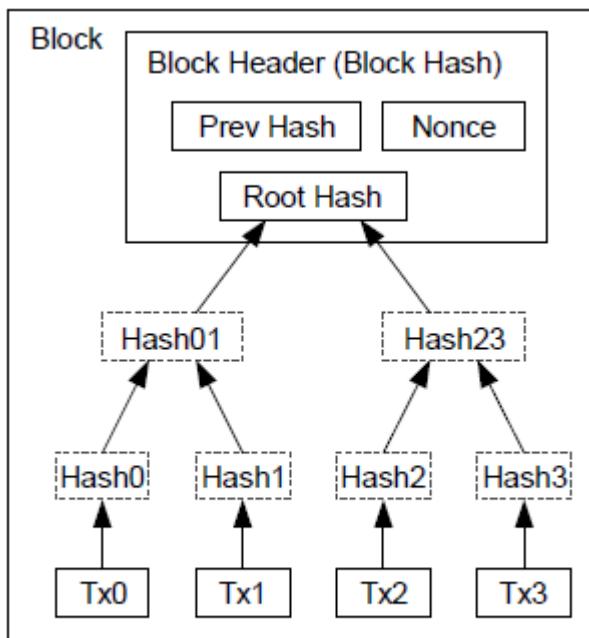
Stimulentul poate fi de asemenea finanțat și din comisioanele de tranzacție. Dacă valoarea de ieșire a unei tranzacții este mai mică decât valoarea de intrare, atunci diferența este un comision de tranzacție care este adăugat la valoarea stimulentului oferit pentru blocul care conține tranzacția. Odată ce un număr predeterminat de monede a intrat în circulație, stimulentul poate trece în întregime la comisioane din tranzacții și poate fi complet lipsit de inflație.

Stimulentul poate încuraja nodurile să rămână oneste. Dacă un atacator lacom reușește să reunească mai multă putere de calcul decât toate nodurile oneste, el ar trebui să decidă între a delapida oamenii furând înapoi plățile sale, și a folosi puterea de calcul pentru a genera noi monede. S-ar putea să fie mai profitabil să joace după reguli, aceste reguli care îi aduc mai multe monede noi decât restului de participanți la un loc, decât să submineze sistemul și validitatea propriei averi.

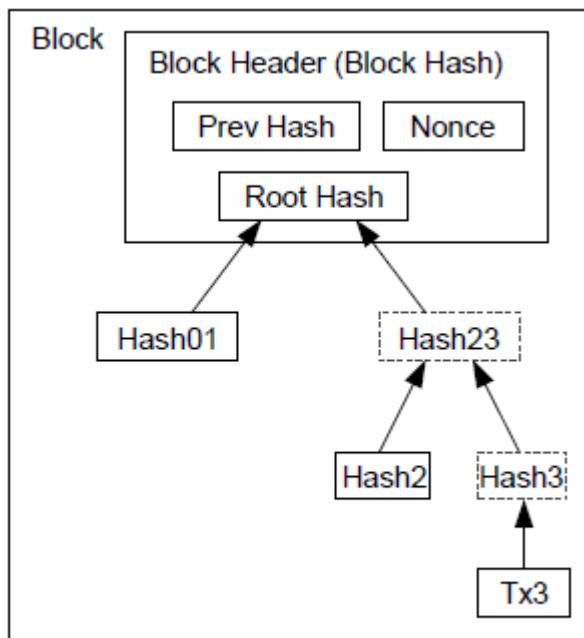
## Recuperarea Spațiului pe Disc

Odată ce ultima tranzacție dintr-o monedă este acoperită de destule blocuri, tranzacțiile cheltuite înaintea ei pot fi înălțurate pentru a economisi spațiu pe disc. Pentru a facilita acest lucru fără a

afecta rezumatul blocului, tranzacțiile sunt rezumate (hashed) într-un arbore Merkle [7] [2] [5], doar rădăcina arborelui fiind inclusă în rezumatul blocului. Blocurile vechi pot fi apoi compactate prin scurtarea ramurilor arborelui. Rezumatele interne nu trebuie să fie stocate.



Transactions Hashed in a Merkle Tree



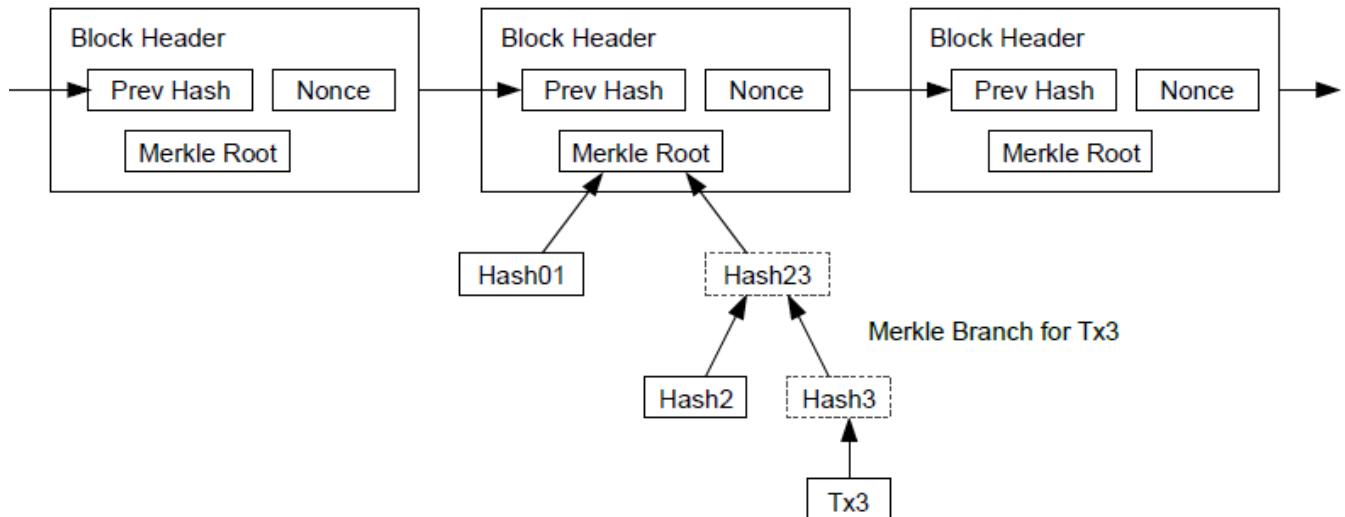
After Pruning Tx0-2 from the Block

Antetul unui bloc fără nici o tranzacție are aproximativ 80 de octeți. Dacă presupunem că blocurile sunt generate la aproximativ 10 minute,  $80 \text{ de octeți} * 6 * 24 * 365 = 4.2\text{MB}$  pe an. Având calculatoare care se vând de obicei cu 2GB de RAM în 2008, iar Legea lui Moore care prevede o creștere de 1.2BG pe an, stocarea nu ar trebui să fie o problemă chiar dacă antetele blocurilor trebuie să fie în memorie.

## Verificarea Simplificată a Plății

Este posibilă verificarea plăților fără a rula un nod de rețea complet. Un utilizator are nevoie să păstreze doar o copie a antetelor celui mai lung lanț bazat pe dovada-de-lucru, pe care o poate obține (copia antetelor) întreagând nodurile rețelei până când este convins că are cel mai lung lanț, și apoi să obțină ramura Merkle legând tranzacția de blocul în care a fost inclusă. Utilizatorul nu poate verifica tranzacția de unul singur, dar legând-o într-un anumit loc în lanț, poate vedea că un nod din rețea a acceptat-o, iar blocurile adăugate după ea, confirmă și mai mult că rețea a acceptat-o.

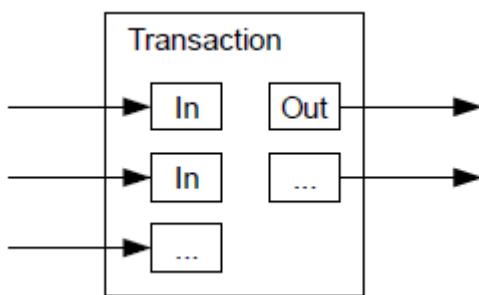
## Longest Proof-of-Work Chain



Ca atare, verificarea este de încredere cât timp nodurile oneste controlează rețeaua, dar este mai vulnerabilă dacă rețeaua este copleșită de un atacator. În timp ce nodurile pot verifica tranzacția direct, metoda simplificată poate fi păcălită de tranzacții inventate ale unui atacator atâtă timp cât atacatorul poate continua să depășească rețeaua. O strategie de protecție împotriva acestui lucru ar fi să accepte alerte de la nodurile rețelei când acestea detectează un bloc invalid, determinând software-ul utilizatorului să downloadeze blocul complet împreună cu tranzacții suspecte pentru a confirma inconsistența. Companiile care primesc plăți frecvent vor dori probabil să ruleze propriile lor noduri pentru a avea o securitate mai independentă și o verificare mai rapidă.

## Combinarea și Divizarea Valorii

Deși ar fi posibil să gestioneze monede în mod individual, ar fi greoi să se execute o tranzacție separată pentru fiecare cent dintr-un transfer. Pentru a permite ca valoarea să fie divizată și combinată, tranzacțiiile conțin intrări și ieșiri multiple. În mod normal va exista fie o singură intrare de la o tranzacție precedentă mai mare fie mai multe intrări care combină sume mai mici, și cel mult două ieșiri: una pentru plată, și una pentru returnarea restului, dacă este cazul, înapoi la expeditor.



Ar trebui menționat că "mufarea", când o tranzacție depinde de câteva alte tranzacții, iar acele tranzacții depind de multe altele, nu este o problemă aici. Nu este niciodată necesară extragerea unei copii complete independente din istoricul unei tranzacții.

## Confidențialitatea

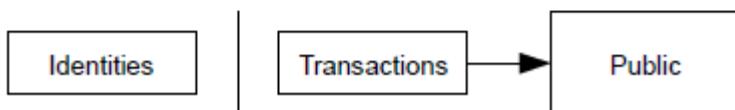
Modelul bancar tradițional atinge un anumit nivel de confidențialitate prin limitarea accesului la

informații celor implicați și a entității terțe considerate de încredere. Necesitatea de a anunța toate tranzacțiile public înălătură acest model, dar confidențialitatea încă poate fi păstrată întrerupând fluxul de informații într-un alt loc: prin păstrarea cheilor publice anonte. Publicul poate vedea că cineva trimite o sumă la altcineva, dar fără informații care să lege tranzacția de cineva anume. Acest lucru este similar cu nivelul de informații deblocat de bursele de valori, unde timpul și mărimea tranzacțiilor individuale, "banda", sunt făcute publice, dar fără a spune cine sunt participanții.

#### Traditional Privacy Model



#### New Privacy Model



Ca protecție suplimentară, o nouă pereche de chei ar trebui folosită pentru fiecare tranzacție pentru a le împiedica să fie corelate la un proprietar comun. Un anumit nivel de corelare nu poate fi evitat pentru tranzacții cu intrări multiple, care în mod necesar dezvăluie că intrările lor au fost deținute de același proprietar. Riscul este că, în cazul în care deținătorul unei chei este dezvăluit, corelarea ar putea trăda alte tranzacții care aparțin aceluiași proprietar.

## Calcule

Considerăm scenariul în care un atacator încearcă să genereze un lanț alternativ mai rapid decât lanțul onest. Chiar dacă acest lucru este realizat, nu înseamnă că sistemul este acum deschis unor schimbări arbitrară, cum ar fi creare de valoare din nimic sau furtul banilor care nu au aparținut niciodata atacatorului. Nodurile nu vor accepta ca plată o tranzacție invalidă, iar nodurile oneste nu vor accepta vreodată un bloc care le conține. Un atacator poate doar să încerce să schimbe una din tranzacțiile lui pentru a recupera banii pe care i-a cheltuit recent.

Cursa între lanțul onest și lanțul atacatorului poate fi caracterizată ca o Plimbare Aleatorie în Tandem. Un eveniment de succes este atunci când lanțul onest este extins cu un bloc, mărindu-și avantajul cu +1, iar un eveniment de eșec este atunci când lanțul atacatorului este extins cu un bloc, reducând diferența cu -1.

Probabilitatea ca un atacator să ajungă din urmă de la o anumită diferență este similară cu problema referitoare la Ruina Pariorului. Presupunem că un parior cu fonduri nelimitate începe la un anume deficit și joacă un număr potențial infinit de încercări pentru a ajunge pe zero. Putem calcula probabilitatea ca el să ajungă vreodată pe zero, sau că un atacator ajunge din urmă lanțul onest, după cum urmează [8]:

$p$  = probabilitatea ca un nod onest să găsească blocul următor

$q$  = probabilitatea ca atacatorul să găsească blocul următor

$q_z$  = probabilitatea ca atacatorul să ajungă din urmă de la  $z$  blocuri în spate

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{cases}$$

Având în vedere presupunerea noastră că  $p > q$ , probabilitatea scade exponențial pe măsură ce crește numărul de blocuri pe care atacatorul trebuie să le ajungă din urmă. Având sortii de izbândă impotriva lui, dacă nu reușește un salt norocos de la început, şansele lui devin extrem de mici pe măsură ce ramâne din ce în ce mai în urmă.

Acum vom lua în considerare cât timp trebuie să aștepte beneficiarul unei noi tranzacții până când să fie suficient de sigur că expeditorul nu poate schimba tranzacția. Presupunem că expeditorul este un atacator care vrea să îl facă pe beneficiar să credă pentru o vreme că a fost plătit, apoi să se plătească înapoi pe el după ce un anumit interval de timp a trecut. Beneficiarul va fi alertat când acest lucru se întâmplă, dar expeditorul speră că va fi prea târziu.

Destinatarul generează o nouă pereche de chei și trimite cheia publică expeditorului cu puțin timp înainte ca acesta să semneze tranzacția. Acest lucru împiedică expeditorul să pregătească un lanț -de-blocuri în avans lucrând la el în mod continuu până când e destul de norocos pentru a ajunge destul de în față, apoi să execute tranzacția la acel moment. Odată ce tranzacția este trimisă, expeditorul necinstit începe lucrul în secret la un lanț paralel ce conține o versiune alternativă a tranzacției originale.

Destinatarul așteaptă până când tranzacția a fost adăugată la un bloc, iar după acel bloc încă  $z$  blocuri au fost adăugate. El nu știe exact progresul pe care atacatorul l-a făcut, dar presupunând că blocurile oneste au fost create în intervalul mediu preconizat de timp per bloc, progresul potențial al atacatorului va fi o distribuție Poisson cu valoarea:

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

Pentru a obține probabilitatea ca atacatorul să poată încă să ajungă din urmă, vom înmulții densitatea Poisson pentru fiecare măsură de progres pe care ar fi putut-o face cu probabilitatea ca să poată ajunge din urmă din acel punct:

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

Rearanjare pentru a evita însumarea cozii infinite a distribuției...

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

implementarea în cod C...

```

#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}

```

După câteva rulări, putem vedea că probabilitatea scade exponențial cu  $z$ .

```

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

```

```

q=0.3
z=0 P=1.0000000
z=5 P=0.1773523
z=10 P=0.0416605
z=15 P=0.0101008
z=20 P=0.0024804
z=25 P=0.0006132
z=30 P=0.0001522
z=35 P=0.0000379
z=40 P=0.0000095
z=45 P=0.0000024
z=50 P=0.0000006

```

Reazolvarea pentru  $P$  mai mic decât 0,1%...

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

## Concluzie

Am propus un sistem pentru tranzacții electronice fără a ne baza pe încredere. Am început cu cadrul obișnuit al monedelor realizate din semnături digitale, care asigură control întărit de deținere, dar este incomplet fără o modalitate de a preveni cheltuirea-dublă. Pentru a rezolva această problemă, am propus o rețea de-la-egal-la-egal folosind dovada-de-lucru pentru a înregistra un istoric public al tranzacțiilor și care devine rapid nepractic computațional pentru un atacator să-l schimbe cât timp nodurile oneste controleaza majoritatea puterii de calcul. Rețeaua este robustă în simplitatea ei nestructurată. Nodurile lucreaza toate simultan cu coordonare minimă. Ele nu trebuie să fie identificate, deoarece mesajele nu sunt direcționate către un anumit loc ci trebuie doar să fie transmise cât se poate de bine mai departe. Nodurile pot părăsi și se pot realătura rețelei după cum doresc, acceptând lanțul de dovadă-de-lucru ca dovadă la ce s-a întâmplat cât timp au fost plecate. Nodurile votează cu puterea lor de calcul, exprimându-și acceptul asupra blocurilor valide și lucrând pentru a le extinde și respingând blocurile invalide prin refuzul de a lucra cu ele. Orice reguli și stimulente pot fi impuse cu acest mecanism de consens.

## Referințe

- [1] W. Dai, "b-money", <http://www.weidai.com/bmoney.txt>, 1998.
- [2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In 20th Symposium on Information Theory in the Benelux, May 1999.
- [3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In Journal of Cryptology, vol 3, no 2, paginile 99-111, 1991.
- [4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In Sequences II: Methods in Communication, Security and Computer Science, paginile 329-334, 1993.
- [5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In Proceedings of the 4th ACM Conference on Computer and Communications Security, paginile 28-35, April 1997.
- [6] A. Back, "Hashcash - a denial of service counter-measure", <http://www.hashcash.org/papers/hashcash.pdf>, 2002.
- [7] R.C. Merkle, "Protocols for public key cryptosystems," In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, paginile 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.

## Licență

Acest referat a fost publicat în octombrie 2009 de către Satoshi Nakamoto. A fost mai târziu (2009) adăugat ca documentație la software-ul bitcoin și are aceeași licență MIT. A fost reprodus în această carte doar cu modificări legate de formatare, sub termenii licenței MIT:

Licența MIT (MIT) - Engleză Copyright (c) 2008 Satoshi Nakamoto

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## Appendix B: Operatori, Constante și Simboluri ale Limbajului de Scriptare pentru Tranzacții

**NOTE** Tabele și descrieri provenite de la <https://en.bitcoin.it/wiki/Script>.

[Împinge valoare pe stivă](#) arată operatorii pentru împingerea valorilor pe stivă.

Table 28. *Împinge valoare pe stivă*

Simbol	Valoare (hexa)	Descriere
OP_0 sau OP_FALSE	0x00	Un sir gol este împins pe stiva
1 până la 75	0x01 până la 0x4b	Împinge următorii N octeți pe stivă, unde N este de la 1 la 75 de octeți
OP_PUSHDATA1	0x4C	Următorul octet script conține N, împinge următorii N octeți pe stivă

Simbol	Valoare (hexa)	Descriere
OP_PUSHDATA2	0x4d	Următorii doi octeți script conțin N, împinge următorii N bytes în stivă
OP_PUSHDATA4	0x4e	Următorii patru octeți script conțin N, împinge următorii N bytes pe stivă
OP_1NEGATE	0x4f	Împinge valoarea "-1" pe stivă
OP_RESERVED	0x50	Stop - tranzacție invalidă dacă nu se găsește într-o clauză OP_IF neexecutată
OP_1 sau OP_TRUE	0x51	Împinge valoarea "1" pe stivă
OP_2 până la OP_16	0x52 până la 0x60	Pentru OP_N, împinge valoarea "N" pe stivă, de exemplu, OP_2 împinge "2"

[Controlul condițional al execuției](#) prezintă operatorii de control condițional al execuției.

Table 29. *Controlul condițional al execuției*

Simbol	Valoare (hexa)	Descriere
OP_NOP	0x61	Nu fă nimic
OP_VER	0x62	Stop - tranzacție invalidă dacă nu se găsește într-o clauză OP_IF neexecutată
OP_IF	0x63	Execută instrucțiunile care urmează dacă vârful stivei nu este 0
OP_NOTIF	0x64	Execută instrucțiunile care urmează dacă vârful stivei este 0
OP_VERIF	0x65	Stop - tranzacție invalidă
OP_VERNOTIF	0x66	Stop - tranzacție invalidă
OP_ELSE	0x67	Execută doar dacă declarațiile anterioare nu au fost executate
OP_ENDIF	0x68	Termina blocul OP_IF, OP_NOTIF, OP_ELSE
OP_VERIFY	0x69	Verifică vârful stivei, oprește și invalidează tranzacția dacă nu e TRUE
OP_RETURN	0x6a	Oprește și invalidează tranzacția

[Operații cu timpii-de-blocare](#) prezintă operatorii utilizati pentru timpii-de-blocare.

Table 30. *Operații cu timpii-de-blocare*

Simbol	Valoare (hexa)	Descriere
OP_CHECKLOCKTIMEVERIFY (anterior OP_NOP2)	0xb1	Marchează tranzacția ca invalidă dacă vârful stivei este mai mare decât câmpul nLockTime al tranzacției, în caz contrar, evaluarea scriptului continuă ca și cum a fost executat un OP_NOP. Tranzacția este de asemenea invalidă dacă 1. stiva este goală; sau 2. elementul din vârful stivei este negativ; sau 3. elementul din vârful stivei este mai mare sau egal cu 500000000, în timp ce câmpul nLockTime al tranzacției este mai mic decât 500000000, sau invers; sau 4. câmpul nSequence al intrării este egal cu 0xffffffff. Semantica precisă este descrisă în BIP-65
OP_CHECKSEQUENCEVERIFY (anterior OP_NOP3)	0xb2	Marchează tranzacția ca invalidă dacă timpul de blocare relativ al intrării (impus de BIP 0068 cu nSequence) nu este egal sau mai lung decât valoarea elementului din vârful stivei. Semantica precisă este descrisă în BIP-112

[Operații pe stivă](#) arată operatorii utilizati pentru manipularea stivei.

Table 31. *Operații pe stivă*

Simbol	Valoare (hexa)	Descriere
OP_TOALTSTACK	0x6b	Scoate elementul din vârful stivei și îl împinge pe stiva alternativă
OP_FROMALTSTACK	0x6c	Scoate elementul din vârful stivei alternative și îl împinge pe stivă
OP_2DROP	0x6d	Scoate două elemente din vârful stivei

Simbol	Valoare (hexa)	Descriere
OP_2DUP	0x6e	Duplică primele două elemente din vârful stivei
OP_3DUP	0x6f	Duplică primele trei elemente din vârful stivei
OP_2OVER	0x70	Copiază cel de-al treilea și al patrulea element din stivă în vârful stivei
OP_2ROT	0x71	Mută al cincilea și al șaselea element din stivă în vârful stivei
OP_2SWAP	0x72	Interschimbă cele două perechi de elemente din vârful stivei
OP_IFDUP	0x73	Duplică elementul din vârful stivei dacă nu este 0
OP_DEPTH	0x74	Numără elementele de pe stivă și împinge numărul rezultat
OP_DROP	0x75	Scoate elementul din vârful stivei
OP_DUP	0x76	Duplică elementul din vârful stivei
OP_NIP	0x77	Scoate al doilea element din stivă
OP_OVER	0x78	Copie al doilea element din stivă și îl împinge în vârful stivei
OP_PICK	0x79	Scoate valoarea de la poziția N din stivă, apoi copiază cel de-al N-ulea element în vârful stivei
OP_ROLL	0x7a	Scoate valoarea de la poziția N din stivă, apoi mută cel de-al N-ulea element în vârful stivei
OP_ROT	0x7b	Rotește primele trei elemente din stivă
OP_SWAP	0x7c	Interschimbă primele două elemente din stivă
OP_TUCK	0x7d	Copiază elementul din vârf și îl introduce între primul și cel de-al doilea element

[Operații de manipulare a sirurilor de caractere](#) prezinta operatori pentru text.

Table 32. Operații de manipulare a șirurilor de caractere

Simbol	Valoare (hexa)	Descriere
OP_CAT	0x7e	Dezactivat (concatenează primele două elemente)
OP_SUBSTR	0x7f	Dezactivat (returnează un sub șir)
OP_LEFT	0x80	Dezactivat (returnează subșirul stâng)
OP_RIGHT	0x81	Dezactivat (returnează subșirul drept)
OP_SIZE	0x82	Calculează lungimea șirului de caractere pentru elementului din vârful stivei și împinge rezultatul

Aritmetică binară și condiționare prezintă operatori binari aritmetici și de logică booleană.

Table 33. Aritmetică binară și condiționare

Simbol	Valoare (hexa)	Descriere
OP_INVERT	0x83	Dezactivat (inversează biții vârfului stivei)
OP_AND	0x84	Dezactivat (AND boolean pe primele două elemente)
OP_OR	0x85	Dezactivat (OR boolean pe primele două elemente)
OP_XOR	0x86	Dezactivat (XOR boolean pe primele două elemente)
OP_EQUAL	0x87	Împinge TRUE (1) dacă primele două elemente din vârf sunt exact egale, altfel împinge FALSE (0)
OP_EQUALVERIFY	0x88	La fel ca OP_EQUAL, dar execută OP_VERIFY și oprește, dacă nu TRUE
OP_RESERVED1	0x89	Stop - tranzacție invalidă dacă nu se găsește într-o clauză OP_IF neexecutată
OP_RESERVED2	0x8a	Halt - tranzacție invalidă dacă nu se găsește într-o clauză OP_IF neexecutată

Operatori numerici prezintă operatori numerici (aritmetici).

Table 34. Operatori numerici

Simbol	Valoare (hexa)	Descriere
OP_1ADD	0x8b	Adăugă 1 la elementul din vârf
OP_1SUB	0x8c	Scade 1 din elementul din vârf
OP_2MUL	0x8d	Dezactivat (multiplică elementul din vârf cu 2)
OP_2DIV	0x8e	Dezactivat (împarte elementul din vârf la 2)
OP_NEGATE	0x8f	Inversează semnul elementului din vârf
OP_ABS	0x90	Schimbă semnul elementului din vârf în pozitiv
OP_NOT	0x91	Dacă elementul din vârf este 0 sau 1 inversează boolean, altfel returnează 0
OP_0NOTEQUAL	0x92	Dacă din vârv este 0 returnează 0, altfel returnează 1
OP_ADD	0x93	Scoate două elemente, le adună și împinge rezultatul
OP_SUB	0x94	Scoate două elemente, scade primul din al doilea, împinge rezultatul
OP_MUL	0x95	Dezactivat (multiplică primele două elemente)
OP_DIV	0x96	Dezactivează (împarte al doilea element la primul element)
OP_MOD	0x97	Dezactivat (restul împărțirii celui de-al doilea element la primul)
OP_LSHIFT	0x98	Dezactivat (deplasează la stânga biții pentru cel de-al doilea element cu un număr specificat de primul element)
OP_RSHIFT	0x99	Dezactivat (deplasează la stânga biții pentru cel de-al doilea element cu un număr specificat de primul element)
OP_BOOLAND	0x9a	AND boolean din primele două elemente

Simbol	Valoare (hexa)	Descriere
OP_BOOLOR	0x9b	OR boolean din primele două elemente
OP_NUMEQUAL	0x9c	Returnează TRUE dacă cele două elemente din vârf sunt numere egale
OP_NUMEQUALVERIFY	0x9d	La fel ca NUMEQUAL, apoi OP_VERIFY pentru a opri, dacă nu TRUE
OP_NUMNOTEQUAL	0x9e	Returnează TRUE dacă cele două elemente din vârf nu sunt numere egale
OP_LESSTHAN	0x9f	Returnează TRUE dacă al doilea element este mai mic decât elementul din vârf
OP_GREATERTHAN	0xa0	Returnează TRUE dacă al doilea element este mai mare decât elementul din vârf
OP_LESSTHANOREQUAL	0xa1	Returnează TRUE dacă al doilea element este mai mic sau egal cu elementul din vârf
OP_GREATERTHANOREQUAL	0xa2	Returnează TRUE dacă al doilea element este mai mare sau egal cu elementul din vârf
OP_MIN	0xa3	Returnează cel mai mic dintre cele două elemente din vârf
OP_MAX	0xa4	Returnează cel mai mare dintre cele două elemente din vârf
OP_WITHIN	0xa5	Returnează TRUE dacă cel de-al treilea element se află între al doilea element (sau egal) și primul element

[Operații criptografice și de rezumare](#) prezintă operatorii funcții criptografice.

Table 35. *Operații criptografice și de rezumare*

Simbol	Valoare (hexa)	Descriere
OP_RIPEMD160	0xa6	Returnează rezumatul RIPEMD160 al elementului din vârf
OP_SHA1	0xa7	Returnă rezumatul SHA1 al elementului din vârf

Simbol	Valoare (hexa)	Descriere
OP_SHA256	0xa8	Returnează rezumatul SHA256 al elementului din vârf
OP_HASH160	0xa9	Returnează rezumatul RIPEMD160 (SHA256 (x)) al elementului din vârf
OP_HASH256	0xaa	Returnează rezumatul SHA256 (SHA256 (x)) al elementului din vârf
OP_CODESEPARATOR	0xab	Marchează începutul datelor verificate prin semnatură
OP_CHECKSIG	0xac	Extrage o cheie publică și semnatura și validează semnatura pentru datele rezumate ale tranzacției, întoarce TRUE dacă se potrivește
OP_CHECKSIGVERIFY	0xad	La fel ca CHECKSIG, apoi OP_VERIFY pentru a opri, dacă nu TRUE
OP_CHECKMULTISIG	0xae	Rulează CHECKSIG pentru fiecare pereche de semnatură și chei publică furnizate. Toate trebuie să se potrivească. Un defect în implementare extrage o valoare suplimentară, prefixare cu OP_0 ca soluție
OP_CHECKMULTISIGVERIFY	0xaf	La fel ca CHECKMULTISIG, apoi OP_VERIFY pentru a opri dacă nu TRUE

**Nonoperatori** afișează simboluri nonoperator.

Table 36. Nonoperatori

Simbol	Valoare (hexa)	Descriere
OP_NOP1 până la OP_NOP10	0xb0 la 0xb9	Nu face nimic, ignorat

**Coduri OP rezervate pentru utilizare internă de către parser** prezintă codurile rezervate pentru a fi folosit de parserul de script intern.

Table 37. Coduri OP rezervate pentru utilizare internă de către parser

Simbol	Valoare (hexa)	Descriere
OP_SMALLDATA	0xf9	Reprezintă un câmp de date mic

Simbol	Valoare (hexa)	Descriere
OP_SMALLINTEGER	0xfa	Reprezintă un câmp de date întregi mici
OP_PUBKEYS	0xfb	Reprezintă câmpuri cheie publică
OP_PUBKEYHASH	0xfd	Reprezintă un câmp rezumat de cheie publică
OP_PUBKEY	0xfe	Reprezintă un câmp cheie publică
OP_INVALIDOPCODE	0xff	Reprezintă orice cod OP care nu este atribuit în prezent

## Appendix C: Propuneri de Îmbunătățire Bitcoin

Propunerile de îmbunătățire bitcoin sunt documente de design care furnizează informații comunității bitcoin sau descriu o nouă funcționalitate pentru bitcoin sau pentru procesele sau mediul său.

În conformitate cu BIP-01 *BIP Motivație și Ghid*, există trei tipuri de BIP:

### **Standard BIP**

Descrie orice schimbare care afectează majoritatea ori toate implementările Bitcoin, cum ar fi o modificare a protocolului de rețea, o schimbare în validarea blocurilor sau tranzacțiilor, sau orice modificare sau completare care afectează interoperabilitatea aplicațiilor care folosesc bitcoin.

### **Informational BIP**

Descrie o problemă de proiectare a bitcoin, sau furnizează orientări generale sau informații pentru comunitatea bitcoin, dar nu propune o nouă funcționalitate. BIP-urile informaționale nu reprezintă în mod necesar un consens în comunitatea bitcoin sau o recomandare, astfel încât utilizatorii și implementatorii pot ignora sau pot urma aceste BIP-uri.

### **Process BIP**

Descrie un proces bitcoin sau propune o modificare a unui proces. BIP-urile pentru proces sunt ca BIP-urile standard, dar se aplică în alte zone decât protocolul bitcoin în sine. Aceste BIP-uri ar putea propune o implementare, dar nu asupra cosului sursă bitcoin; adesea necesită consens comunitar; și spre deosebire de BIP-urile informaționale, acestea sunt mai mult decât recomandări, iar utilizatorii nu sunt de obicei liberi să le ignore. Exemple includ proceduri, ghiduri, modificări ale procesului de luare a deciziilor și modificări la instrumentele sau mediul utilizat în dezvoltarea bitcoin. Orice meta-BIP este, de asemenea, considerat un proces BIP.

BIP-urile sunt înregistrate într-un repo versionat pe GitHub: <https://github.com/bitcoin/bips>. Starea [BIP-urilor](#) prezintă o imagine a BIP-urilor din aprilie 2017. Consultați repo-ul autoritar pentru

informații actualizate despre BIP-urile existente și conținutul acestora ("proponeri de îmbunătățire bitcoin", "stare"), id = „BIPsnap15” )

Table 38. Starea BIP-urilor

Număr BIP	Titlu	Proprietar	Tip	Staș
BIP-1	BIP Purpose and Guidelines	Amir Taaki	Process	Replaced
BIP-2	BIP process, revised	Luke Dashjr	Process	Active
BIP-8	Version bits with guaranteed lock-in	Shaolin Fry	Informational	Draft
BIP-9	Version bits with timeout and delay	Pieter Wuille, Peter Todd, Greg Maxwell, Rusty Russell	Informational	Final
BIP-10	Multi-Sig Transaction Distribution	Alan Reiner	Informational	Withdrawn
BIP-11	M-of-N Standard Transactions	Gavin Andresen	Standard	Final
BIP-12	OP_EVAL	Gavin Andresen	Standard	Withdrawn
BIP-13	Address Format for pay-to-script-hash	Gavin Andresen	Standard	Final
BIP-14	Protocol Version and User Agent	Amir Taaki, Patrick Strateman	Standard	Final
BIP-15	Aliases	Amir Taaki	Standard	Deferred
BIP-16	Pay to Script Hash	Gavin Andresen	Standard	Final
BIP-17	OP_CHECKHASHVERIFY (CHV)	Luke Dashjr	Standard	Withdrawn
BIP-18	hashScriptCheck	Luke Dashjr	Standard	Proposed
BIP-19	M-of-N Standard Transactions (Low SigOp)	Luke Dashjr	Standard	Draft
BIP-20	URI Scheme	Luke Dashjr	Standard	Replaced
BIP-21	URI Scheme	Nils Schneider, Matt Corallo	Standard	Final
BIP-22	getblocktemplate - Fundamentals	Luke Dashjr	Standard	Final

Număr BIP	Titlu	Proprietar	Tip	Stuș
BIP-23	getblocktemplate - Pooled Mining	Luke Dashjr	Standard	Final
BIP-30	Duplicate transactions	Pieter Wuille	Standard	Final
BIP-31	Pong message	Mike Hearn	Standard	Final
BIP-32	Hierarchical Deterministic Wallets	Pieter Wuille	Informational	Final
BIP-33	Stratized Nodes	Amir Taaki	Standard	Draft
BIP-34	Block v2, Height in Coinbase	Gavin Andresen	Standard	Final
BIP-35	mempool message	Jeff Garzik	Standard	Final
BIP-36	Custom Services	Stefan Thomas	Standard	Draft
BIP-37	Connection Bloom filtering	Mike Hearn, Matt Corallo	Standard	Final
BIP-39	Mnemonic code for generating deterministic keys	Marek Palatinus, Pavol Rusnak, Aaron Voisine, Sean Bowe	Standard	Proposed
BIP-40	Stratum wire protocol	Marek Palatinus	Standard	BIP number allocated
BIP-41	Stratum mining protocol	Marek Palatinus	Standard	BIP number allocated
BIP-42	A finite monetary supply for Bitcoin	Pieter Wuille	Standard	Draft
BIP-43	Purpose Field for Deterministic Wallets	Marek Palatinus, Pavol Rusnak	Informational	Draft
BIP-44	Multi-Account Hierarchy for Deterministic Wallets	Marek Palatinus, Pavol Rusnak	Standard	Proposed
BIP-45	Structure for Deterministic P2SH Multisignature Wallets	Manuel Araoz, Ryan X. Charles, Matias Alejo Garcia	Standard	Proposed

Număr BIP	Titlu	Proprietar	Tip	Stuș
<a href="#">BIP-47</a>	Reusable Payment Codes for Hierarchical Deterministic Wallets	Justus Rannvier	Informational	Draft
<a href="#">BIP-49</a>	Derivation scheme for P2WPKH-nested-in-P2SH based accounts	Daniel Weigl	Informational	Draft
<a href="#">BIP-50</a>	March 2013 Chain Fork Post-Mortem	Gavin Andresen	Informational	Final
<a href="#">BIP-60</a>	Fixed Length "version" Message (Relay-Transactions Field)	Amir Taaki	Standard	Draft
<a href="#">BIP-61</a>	Reject P2P message	Gavin Andresen	Standard	Final
<a href="#">BIP-62</a>	Dealing with malleability	Pieter Wuille	Standard	Withdrawn
<a href="#">BIP-63</a>	Stealth Addresses	Peter Todd	Standard	BIP number allocated
<a href="#">BIP-64</a>	getutxo message	Mike Hearn	Standard	Draft
<a href="#">BIP-65</a>	OP_CHECKLOCKTIMEVERIFY	Peter Todd	Standard	Final
<a href="#">BIP-66</a>	Strict DER signatures	Pieter Wuille	Standard	Final
<a href="#">BIP-67</a>	Deterministic Pay-to-script-hash multi-signature addresses through public key sorting	Thomas Kerin, Jean-Pierre Rupp, Ruben de Vries	Standard	Proposed
<a href="#">BIP-68</a>	Relative lock-time using consensus-enforced sequence numbers	Mark Friedenbach, BtcDrak, Nicolas Dorier, kinoshitajona	Standard	Final
<a href="#">BIP-69</a>	Lexicographical Indexing of Transaction Inputs and Outputs	Kristov Atlas	Informational	Proposed

Număr BIP	Titlu	Proprietar	Tip	Stuș
<a href="#">BIP-70</a>	Payment Protocol	Gavin Andresen, Mike Hearn	Standard	Final
<a href="#">BIP-71</a>	Payment Protocol MIME types	Gavin Andresen	Standard	Final
<a href="#">BIP-72</a>	bitcoin: uri extensions for Payment Protocol	Gavin Andresen	Standard	Final
<a href="#">BIP-73</a>	Use "Accept" header for response type negotiation with Payment Request URLs	Stephen Pair	Standard	Final
<a href="#">BIP-74</a>	Allow zero value OP_RETURN in Payment Protocol	Toby Padilla	Standard	Draft
<a href="#">BIP-75</a>	Out of Band Address Exchange using Payment Protocol Encryption	Justin Newton, Matt David, Aaron Voisine, James MacWhyte	Standard	Draft
<a href="#">BIP-80</a>	Hierarchy for Non-Colored Voting Pool Deterministic Multisig Wallets	Justus Ravnier, Jimmy Song	Informational	Deferred
<a href="#">BIP-81</a>	Hierarchy for Colored Voting Pool Deterministic Multisig Wallets	Justus Ravnier, Jimmy Song	Informational	Deferred
<a href="#">BIP-83</a>	Dynamic Hierarchical Deterministic Key Trees	Eric Lombrozo	Standard	Draft
<a href="#">BIP-90</a>	Buried Deployments	Suhas Daftuar	Informational	Draft
<a href="#">BIP-99</a>	Motivation and deployment of consensus rule changes ([soft/hard]forks)	Jorge Timón	Informational	Draft

Număr BIP	Titlu	Proprietar	Tip	Stuș
<a href="#">BIP-101</a>	Increase maximum block size	Gavin Andresen	Standard	Withdrawn
<a href="#">BIP-102</a>	Block size increase to 2MB	Jeff Garzik	Standard	Draft
<a href="#">BIP-103</a>	Block size following technological growth	Pieter Wuille	Standard	Draft
<a href="#">BIP-104</a>	'Block75' - Max block size like difficulty	t.khan	Standard	Draft
<a href="#">BIP-105</a>	Consensus based block size retargeting algorithm	BtcDrak	Standard	Draft
<a href="#">BIP-106</a>	Dynamically Controlled Bitcoin Block Size Max Cap	Upal Chakraborty	Standard	Draft
<a href="#">BIP-107</a>	Dynamic limit on the block size	Washington Y. Sanchez	Standard	Draft
<a href="#">BIP-109</a>	Two million byte size limit with sigop and sighash limits	Gavin Andresen	Standard	Rejected
<a href="#">BIP-111</a>	NODE_BLOOM service bit	Matt Corallo, Peter Todd	Standard	Proposed
<a href="#">BIP-112</a>	CHECKSEQUENCE VERIFY	BtcDrak, Mark Friedenbach, Eric Lombrozo	Standard	Final
<a href="#">BIP-113</a>	Median time-past as endpoint for lock-time calculations	Thomas Kerin, Mark Friedenbach	Standard	Final
<a href="#">BIP-114</a>	Merkelized Abstract Syntax Tree	Johnson Lau	Standard	Draft
<a href="#">BIP-120</a>	Proof of Payment	Kalle Rosenbaum	Standard	Draft
<a href="#">BIP-121</a>	Proof of Payment URI scheme	Kalle Rosenbaum	Standard	Draft

Număr BIP	Titlu	Proprietar	Tip	Stuș
<a href="#">BIP-122</a>	URI scheme for Blockchain references / exploration	Marco Pontello	Standard	Draft
<a href="#">BIP-123</a>	BIP Classification	Eric Lombrozo	Process	Active
<a href="#">BIP-124</a>	Hierarchical Deterministic Script Templates	Eric Lombrozo, William Swanson	Informational	Draft
<a href="#">BIP-125</a>	Opt-in Full Replace-by-Fee Signaling	David A. Harding, Peter Todd	Standard	Proposed
<a href="#">BIP-126</a>	Best Practices for Heterogeneous Input Script Transactions	Kristov Atlas	Informational	Draft
<a href="#">BIP-130</a>	sendheaders message	Suhas Daftuar	Standard	Proposed
<a href="#">BIP-131</a>	"Coalescing Transaction" Specification (wildcard inputs)	Chris Priest	Standard	Draft
<a href="#">BIP-132</a>	Committee-based BIP Acceptance Process	Andy Chase	Process	Withdrawn
<a href="#">BIP-133</a>	feefilter message	Alex Morcos	Standard	Draft
<a href="#">BIP-134</a>	Flexible Transactions	Tom Zander	Standard	Draft
<a href="#">BIP-140</a>	Normalized TXID	Christian Decker	Standard	Draft
<a href="#">BIP-141</a>	Segregated Witness (Consensus layer)	Eric Lombrozo, Johnson Lau, Pieter Wuille	Standard	Draft
<a href="#">BIP-142</a>	Address Format for Segregated Witness	Johnson Lau	Standard	Deferred
<a href="#">BIP-143</a>	Transaction Signature Verification for Version 0 Witness Program	Johnson Lau, Pieter Wuille	Standard	Draft

Număr BIP	Titlu	Proprietar	Tip	Stuș
<a href="#">BIP-144</a>	Segregated Witness (Peer Services)	Eric Lombrozo, Pieter Wuille	Standard	Draft
<a href="#">BIP-145</a>	getblocktemplate Updates for Segregated Witness	Luke Dashjr	Standard	Draft
<a href="#">BIP-146</a>	Dealing with signature encoding malleability	Johnson Lau, Pieter Wuille	Standard	Draft
<a href="#">BIP-147</a>	Dealing with dummy stack element malleability	Johnson Lau	Standard	Draft
<a href="#">BIP-148</a>	Mandatory activation of segwit deployment	Shaolin Fry	Standard	Draft
<a href="#">BIP-150</a>	Peer Authentication	Jonas Schnelli	Standard	Draft
<a href="#">BIP-151</a>	Peer-to-Peer Communication Encryption	Jonas Schnelli	Standard	Draft
<a href="#">BIP-152</a>	Compact Block Relay	Matt Corallo	Standard	Draft
<a href="#">BIP-171</a>	Currency/exchange rate information API	Luke Dashjr	Standard	Draft
<a href="#">BIP-180</a>	Block size/weight fraud proof	Luke Dashjr	Standard	Draft
<a href="#">BIP-199</a>	Hashed Time-Locked Contract transactions	Sean Bowe, Daira Hopwood	Standard	Draft

## Appendix D: Bitcore

Bitcore este o suită de instrumente oferite de BitPay. Scopul ei este să ofere instrumente ușor de folosit pentru programatorii Bitcoin. Aproape tot codul Bitcore este scris în JavaScript. Există unele module scrise special pentru NodeJS. Modulul "nod" al Bitcore include codul C++ al Bitcoin Core. Vă rugăm vizitați <https://bitcore.io> pentru mai multe informații.

# **Lista de Funcționalități Bitcore**

- Nod-complet bitcoin (bitcore-node)
- Explorator de blocuri (insight)
- Utilitar pentru blocuri, tranzacții, și portofel (bitcore-lib)
- Comunicarea directă cu rețeaua de-la-egal-la-egal bitcoin (bitcore-p2p)
- Generare entropie pentru sămânță mnemonică (bitcore-mnemonic)
- Protocol de plată (bitcore-payment-protocol)
- Verificarea și semnarea mesajelor (bitcore-message)
- Schemă de Criptare Integrată pentru Curba Eliptică (bitcore-ecies)
- Serviciu portofel (bitcoin-wallet-service)
- Client portofel (bitcore-wallet-client)
- Integrarea serviciilor direct cu Bitcoin Core (bitcore-node)

## **Exemple din Biblioteca Bitcore**

### **Cerințe Preliminare**

- NodeJS >= 4.x

Dacă utilizați NodeJS și nodul REPL:

```
$ npm install -g bitcore-lib bitcore-p2p
```

### **Exemple de Portofel folosind bitcore-lib**

Crearea unei noi adrese bitcoin cu cheia privată asociată:

```
> bitcore = require('bitcore-lib')
> privateKey = new bitcore.PrivateKey()
> address = privateKey.toAddress().toString()
```

Crearea unei chei private deterministe ierarhic și a adresei:

```
> hdPrivateKey = bitcore.HDPrivateKey()
> hdPublicKey = bitcore.HDPublicKey(hdPrivateKey)
> hdAddress = new bitcore.Address(hdPublicKey.publicKey).toString()
```

Crearea și semnarea unei tranzacții de la un UTXO:

```

> utxo = {
  txId: transaction id containing an unspent output,
  outputIndex: output index e.g. 0,
  address: addressOfUtxo,
  script: bitcore.Script.buildPublicKeyHashOut(addressOfUtxo).toString(),
  satoshis: amount sent to the address
}
> fee = 3000 //set appropriately for conditions on the network
> tx = new bitcore.Transaction()
  .from(utxo)
  .to(address, 35000)
  .fee(fee)
  .enableRBF()
  .sign(privateKeyOfUtxo)

```

Înlocuiți ultima tranzacție din mempool pe baza comisionului (replace-by-fee):

```

> rbfTx = new Transaction()
  .from(utxo)
  .to(address, 35000)
  .fee(fee*2)
  .enableRBF()
  .sign(privateKeyOfUtxo);
> tx.serialize();
> rbfTx.serialize();

```

Difuzarea unei tranzacții către rețeaua bitcoin (notă: difuzați numai tranzacții valide; consultați <https://bitnodes.21.co/nodes> pentru gazdele de-la-egal-la-egal):

1. Copiați codul de mai jos într-un fișier numit *broadcast.js*.
2. Variabilele tx și rbfTx sunt ieșirea din tx.serialize() și rbfTx.serialize().
3. Pentru a înlocui comisionul (replace-by-fee), seamănul (peer) din rețea trebuie să suporte opțiunea mempoolreplace a bitcoind și să o aibă setată pe 1.
4. Rulați fișierul node *broadcast.js*:

```

var p2p = require('bitcore-p2p');
var bitcore = require('bitcore-lib');
var tx = new bitcore.Transaction('output from serialize function');
var rbfTx = new bitcore.Transaction('output from serialize function');
var host = 'ip address'; //use valid peer listening on tcp 8333
var peer = new p2p.Peer({host: host});
var messages = new p2p.Messages();
peer.on('ready', function() {
  var txs = [messages.Transaction(tx), messages.Transaction(rbfTx)];
  var index = 0;
  var interval = setInterval(function() {
    peer.sendMessage(txs[index++]);
    console.log('tx: ' + index + ' sent');
    if (index === txs.length) {
      clearInterval(interval);
      console.log('disconnecting from peer: ' + host);
      peer.disconnect();
    }
  }, 2000);
});
peer.connect();

```

## Appendix E: pycoin, ku, și tx

Biblioteca Python [pycoin](#), scrisă inițial și întreținută de Richard Kiss, este o bibliotecă bazată pe Python care acceptă manipularea cheilor bitcoin și a tranzacțiilor, chiar și suportarea limbajului de script suficient pentru a face față corect tranzacțiilor non standard.

Biblioteca pycoin acceptă atât Python 2 (2.7.x) cât și Python 3 (3.3 și ulterior) și vine cu câteva utilitare pentru linia de comandă, [ku](#) și [tx](#).

### Utilitar pentru Chei (Key Utility - KU)

Utilitarul din linia de comandă [ku](#) ("key utility") este ca un cuțit elvețian pentru manipularea cheilor. Acceptă cheile BIP-32, WIF și adrese (monede bitcoin și altcoin). Urmează câteva exemple.

Crează o cheie BIP-32 folosind sursele de entropie implicate ale GPG și [/dev/random](#):

```

$ ku create

input          : create
network        : Bitcoin
wallet key     : xprv9s21ZrQH143K3LU5ctPZTBnb9kTjA5Su9DcWHvXJemiJBsY7VqXUG7hipgdWaU
                 m2nhnzdvxJf5KJ09vjP2nABX65c5sFsWsV8oXcbpehtJi
public version : xpub661MyMwAqRbcFpYYiuvZpKjKhnJDZYAkWSY76JvvD7FH4fsG3Nqiov2CfxzxY8
                 D6cpfT56AMFeo8M8KPkFMfLUtvwjwb6WPv8rY65L2q8Hz
tree depth     : 0
fingerprint   : 9d9c6092
parent f'print : 00000000
child index    : 0
chain code     : 80574fb260edaa4905bc86c9a47d30c697c50047ed466c0d4a5167f6821e8f3c
private key    : yes
secret exponent :
112471538590155650688604752840386134637231974546906847202389294096567806844862
  hex          : f8a8a28b28a916e1043cc0aca52033a18a13cab1638d544006469bc171fddfbe
  wif          : L5Z54xi6qJusQT42JHA44mfPVZGjyb4XBRWfxAzUWwRiGx1kV4sP
  uncompressed : 5KhoEavGNNH4GHKoy2Ptu4KfdNp4r56L5B5un8FP6RZnbsz5Nmb
public pair x  :
76460638240546478364843397478278468101877117767873462127021560368290114016034
public pair y  :
59807879657469774102040120298272207730921291736633247737077406753676825777701
  x as hex    : a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
  y as hex    : 843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcfd625
  y parity    : odd
key pair as sec:
  compressed  : 03a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
  uncompressed : 04a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
                 843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcfd625
hash160        : 9d9c609247174ae323acf96c852753fe3c8819d
  uncompressed : 8870d869800c9b91ce1eb460f4c60540f87c15d7
Bitcoin address:
  compressed  : 1FNNRQ5fSv1wBi5gyfVBs2rkNheMGt86sp
  uncompressed : 1DSS5isnH4FsVaLVjeVXewVSpfqktdiQAM

```

Crează o cheie BIP-32 dintr-o frază de acces:

**WARNING** Fraza de acces din acest exemplu este mult prea ușor de ghicit.

```
$ ku P:foo
```

```
input          : P:foo
network        : Bitcoin
wallet key    : xprv9s21ZrQH143K31AgNK5pyVvW23gHnkBq2wh5aEk6g1s496M8ZMjxncCKZKgb5j
                  ZoY5eSJMJ2Vbyvi2hbmQnCuHBujZ2WXGTux1X2k9Krdtq
versiune publică : xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtS
                  VYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy
tree depth    : 0
fingerprint   : 5d353a2e
parent f'print : 00000000
child index   : 0
chain code    : 5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc
private key   : yes
secret exponent :
65825730547097305716057160437970790220123864299761908948746835886007793998275
  hex          : 91880b0e3017ba586b735fe7d04f1790f3c46b818a2151fb2def5f14dd2fd9c3
  wif          : L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
  uncompressed : 5JvNzA5vXDoKYJdw8SwwLHxUxaWvn9mDea6k1vRPCX7KLUVWa7W
public pair x :
81821982719381104061777349269130419024493616650993589394553404347774393168191
public pair y :
58994218069605424278320703250689780154785099509277691723126325051200459038290
  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 : 1MwkRkogzBRMehBntgcq2aJhXCXStJTXHT
```

Obține informația în format JSON:

```
$ ku P:foo -P -j
```

```
{
  "yparity": "even",
  "publicpairyhex": "826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
  "privatekey": "no",
  "parentfingerprint": "00000000",
  "treedepth": "0",
  "network": "Bitcoin",
  "btcaddressuncompressed": "1MwkRkogzBRMehBntgcq2aJhXCXStJTXHT",
  "keypairassecuncompressed": "04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
  "publicpairxhex": "b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f",
  "walletkey": "xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtSVYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy",
  "chaincode": "5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc",
  "childindex": "0",
  "hash160uncompressed": "e5bd3a7e6cb62b4c820e51200fb1c148d79e67da",
  "btcaddress": "19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii",
  "fingerprint": "5d353a2e",
  "hash160": "5d353a2ecdb262477172852d57a3f11de0c19286",
  "input": "P:foo",
  "publicpairx": "81821982719381104061777349269130419024493616650993589394553404347774393168191",
  "publicpairy": "58994218069605424278320703250689780154785099509277691723126325051200459038290",
  "keypairassec": "02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f"
}
```

Cheie Publică BIP32:

```
$ ku -w -P P:foo
xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtSVYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy
```

Genereză o sub-cheie:

```
$ ku -w -s3/2 P:foo
xprv9wTERTSkjVyJa1v4cUTMFkWMe5eu8ErbQcs9xajnsUzCBT7ykHAwdrxvG3g3f6BFk7ms5hHBvmbdutNmyg6iogWKxx6mefEw4M8EroLgKj
```

Subcheie întărită:

```
$ ku -w -s3/2H P:foo
xprv9wTERTSu5AWGkDeUPmqBcbZX1xq85ZNX9iQRQW9DXwygFp7iRGJo79dsVctcsCHsnZ3XU3DhsuaGZbDh8
iDkBN45k67UKsJUXM1JfRCdn1
```

WIF:

```
$ ku -W P:foo
L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
```

Adresă:

```
$ ku -a P:foo
19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii
```

Genereaza o grămadă de subchei:

```
$ ku P:foo -s 0/0-5 -w
xprv9xWkBDfyBXmZjBG9EiXBpy67KK72fphUp9utJokEBFtjsjiuKUUDF5V3TU8U8cDzytqYnSekc8bYuJS8G3
bhXxKWB89Ggn2dzLcoJsuEdRK
xprv9xWkBDfyBXmZnzKf3bAGifK593gT7WJZPnYAmvc77gUQVej5QHckc5Adtwxa28ACmANi9XhCrRvtFqQcUx
t8rUgFz3souMiDdWxJDZnQxzx
xprv9xWkBDfyBXmZqdXA8y4SWqfBdy71gSW9sjx9JpCiJEiBwSMQyRxan6srXUPBtj3PTxQFkZJAiwoUpmvtrx
KZu4zfsnr3pqyy2vthpkwoVq
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8Esk
pzKL1Y8Gk9aX6QbryA5raK73p
xprv9xWkBDfyBXmZv2q3N66hhZ8DAcEnQDnXML1J62krJAcf7Xb1HJwuW2VMJQrCofY2jtFXdiEY8UsRNJfqK6
DAdyZXoMvtaLHyWQx3FS4A9zw
xprv9xWkBDfyBXmZw4jEYXUHYc9fT25k9irP87n2RqfJ5bqbjKdT84Mm7Wtc2xmzFuKg7iYf7XFHkkSsaYKWKJ
bR54bnyAD9GzjUYbAYTtN4ruo
```

Genereaza adresele corespunzătoare:

```
$ ku P:foo -s 0/0-5 -a
1MrjE78H1R1rqdFrmkjdhnPUDLCJALbv3x
1AnYyVEcuqeoVzH96zj1eYKwoWfwte2pxu
1GXr1kZfxE1FcK6ZRD5sqqqs5YfvuzA1Lb
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
1Cz2rTLjRM6pMnxPNrRKp9ZSvRtj5dDUML
1WstdwPnU6HEUPme1DQayN9nm6j7nDVE
```

Generarea de WIFs corespunzătoare:

```
$ ku P:foo -s 0/0-5 -W
L5a4iE5k9gcJKGqX3FWmxzBYQc29PvZ6pgBaePLVqT5YByEnBomx
Kyjgne6GZwPGB6G6kJEhoPbmyjMP7D5d3zRbHVjwcq4iQXD9QqKQ
L4B3ygQxK6zH2NQGxLDee2H9v4Lvvg14cLJW7QwWPzCtKhdWMaQz
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmTDMrqemY8UF
L2oD6vA4TUyqPF8QG4vhUFsgwCyuvFZ3v8SKHYFDwkbM765Nrfd
KzChTbc3kZFxUSJ3Kt54cxsogeFAD9CCM4zGB22si8nfKcThQn8C
```

Verificați dacă funcționează prin alegerea unui sir BIP32 (cel corespunzător subiectului 0/3):

```
$ ku -W
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8Esk
pzKL1Y8Gk9aX6QbryA5raK73p
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmTDMrqemY8UF
$ ku -a
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8Esk
pzKL1Y8Gk9aX6QbryA5raK73p
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
```

Da, pare familiar.

Din exponentul secret:

```
$ ku 1

input          : 1
network        : Bitcoin
secret exponent : 1
hex            : 1
wif            : KwDiBf89QgGbxEhKnhXJuH7LrciVrZi3qYjgd9M7rFU73sVHnoWn
uncompressed   : 5HpHagT65TZzG1PH3CSu63k8DbpvD8s5ip4nEB3kEsreAnchuDf
public pair x  :
5506626302227734366957871889516853432625060345377594175500187360389116729240
public pair y  :
32670510020758816978083085130507043184471273380659243275938904335757337482424
  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: 1Bg6Z9tcN4rm9KBzDn7KprQz87SZ26SAMH
  uncompressed  : 1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZm
```

Versiunea Litecoin:

```
$ ku -nL 1

input          : 1
network        : Litecoin
secret exponent : 1
hex            : 1
wif            : T33ydQRKp4FCW5LCLLUB7deioUMoveiwekdwUwyfRDeGZm76aUjV
uncompressed   : 6u823ozcyt2rjPH8Z2ErsSXJB5PPQwK7VVTwwN4mxLBFrao69XQ
public pair x  :
55066263022277343669578718895168534326250603453777594175500187360389116729240
public pair y  :
32670510020758816978083085130507043184471273380659243275938904335757337482424
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
QNcdLVw8fHkixm6NNyN6nVwxKek4u7qrioRbQmjxac5TVoTtZuot
```

Din perechea publică (pe Testnet):

```

$ ku -nT
55066263022277343669578718895168534326250603453777594175500187360389116729240,even

input          :
550662630222773436695787188951685343262506034537775941755001873603
                           89116729240,even
network        : Bitcoin testnet
public pair x  :
55066263022277343669578718895168534326250603453777594175500187360389116729240
public pair y  :
32670510020758816978083085130507043184471273380659243275938904335757337482424
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   : mtoKs9V381UAhUi3d7Vb9GNak8Qvmcsme

```

Din hash160:

```

$ ku 751e76e8199196d454941c45d1b3a323f1433bd6

input          : 751e76e8199196d454941c45d1b3a323f1433bd6
network        : Bitcoin
hash160        : 751e76e8199196d454941c45d1b3a323f1433bd6
Bitcoin address : 1Bg6Z9tcN4rm9KBzDn7KprQz87SZ26SAMH

```

Ca adresă Dogecoin:

```

$ ku -nD 751e76e8199196d454941c45d1b3a323f1433bd6

input          : 751e76e8199196d454941c45d1b3a323f1433bd6
network        : Dogecoin
hash160        : 751e76e8199196d454941c45d1b3a323f1433bd6
Dogecoin address : DFpN6QqFfUm3gKNaxN6tNcab1FArL9cZLE

```

## Utilitar pentru Tranzacții (TX)

Utilitarul de la linia de comandă `tx` va afișa tranzacții sub o formă care poate fi citită de oameni, va prelua tranzacțiile de bază din cache-ul Pycoin sau de la servicii web (blockchain.info, blockcypher.com, blockr.io și chain.so sunt acceptate în prezent), va uni tranzacțiile, va adăuga sau șterge intrări sau ieșiri și va semna tranzacțiile.

În continuare sunt prezentate câteva exemple

Vizualizați celebra tranzacție pentru "pizza":

```
$ tx 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
warning: consider setting environment variable PYCOIN_CACHE_DIR=~/pycoin_cache to
cache transactions fetched via web services
warning: no service providers found for get_tx; consider setting environment variable
PYCOIN_BTC_PROVIDERS
usage: 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
```

Ups! Nu avem servicii web setate. Să facem asta acum:

```
$ PYCOIN_CACHE_DIR=~/pycoin_cache
$ PYCOIN_BTC_PROVIDERS="block.io blockchain.info blockexplorer.com"
$ export PYCOIN_CACHE_DIR PYCOIN_BTC_PROVIDERS
```

Nu se face automat, astfel încât un instrument de linie de comandă nu va scurge informații potențial private despre tranzacțiile care vă interesează pe un site terț. Dacă nu vă pasă, puteți introduce aceste rânduri în `.profile`.

Să încercăm din nou:

```

$ 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
includem unspents în hex deoarece tranzacția nu este total semnată
01000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a49
3046022100a7f26eda874931999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e91
99238eb73f07c8f209504c84b80f03e30ed8169edd44f80ed17ddf451901fffffff010010a5d4e8000000
1976a9147ec1003336542cae8bded8909cdd6b5e48ba0ab688ac0000000

** can't validate transaction as source transactions missing

```

Linia finală apare pentru a valida semnăturile tranzacțiilor, aveți tehnic nevoie de tranzacțiile sursă. Deci, să adăugăm **-a** pentru a augmenta tranzacțiile cu informații sursă:

```

$ 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

01000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a49
3046022100a7f26eda874931999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e91
99238eb73f07c8f209504c84b80f03e30ed8169edd44f80ed17ddf451901fffffff010010a5d4e8000000
1976a9147ec1003336542cae8bded8909cdd6b5e48ba0ab688ac0000000

all incoming transaction values validated

```

Acum, să ne uităm la ieșirile necheltuite pentru o anumită adresă (UTXO). În blocul #1, vedem o tranzacție coinbase la `12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX`. Să folosim `fetch_unspent` pentru a găsi toate monedele de la această adresă:

```
$ fetch_unspent 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX
a3a6f902a51a2cbebede144e48a88c05e608c2cce28024041a5b9874013a1e2a/0/76a914119b098e2e980
a229e139a9ed01a469e518e6f2688ac/333000
cea36d008badf5c7866894b191d3239de9582d89b6b452b596f1f1b76347f8cb/31/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/10000
065ef6b1463f552f675622a5d1fd2c08d6324b4402049f68e767a719e2049e8d/86/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/10000
a66ddddd42f9f2491d3c336ce5527d45cc5c2163aaed3158f81dc054447f447a2/0/76a914119b098e2e980
a229e139a9ed01a469e518e6f2688ac/10000
ffd901679de65d4398de90cefef68d2c3ef073c41f7e8dbec2fb5cd75fe71dfe7/0/76a914119b098e2e980
a229e139a9ed01a469e518e6f2688ac/100
d658ab87cc053b8dbcfd4aa2717fd23cc3edfe90ec75351fadd6a0f7993b461d/5/76a914119b098e2e980
a229e139a9ed01a469e518e6f2688ac/911
36ebe0ca3237002acb12e1474a3859bde0ac84b419ec4ae373e63363ebef731c/1/76a914119b098e2e980
a229e139a9ed01a469e518e6f2688ac/10000
fd87f9adecbb17f4ebb1673da76ff48ad29e64b7afa02fda0f2c14e43d220fe24/0/76a914119b098e2e980
a229e139a9ed01a469e518e6f2688ac/1
dfdf0b375a987f17056e5e919ee6eadd87dad36c09c4016d4a03cea15e5c05e3/1/76a914119b098e2e980
a229e139a9ed01a469e518e6f2688ac/1337
cb2679bfd0a557b2dc0d8a6116822f3fcbe281ca3f3e18d3855aa7ea378fa373/0/76a914119b098e2e980
a229e139a9ed01a469e518e6f2688ac/1337
d6be34ccf6edddc3cf69842dce99fe503bf632ba2c2adb0f95c63f6706ae0c52/1/76a914119b098e2e980
a229e139a9ed01a469e518e6f2688ac/2000000
0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098/0/410496b538e853519c7
26a2c91e61ec11600ae1390813a627c66fb8be7947be63c52da7589379515d4e0a604f8141781e62294721
166bf621e73a82cbf2342c858eeac/5000000000
```

## Appendix F: Comenzi Bitcoin Explorer (bx)

Bitcoin Explorer (bx) este un instrument pentru linia de comandă care oferă o varietate de comenzi pentru gestionarea cheilor și construcția tranzacțiilor. Face parte din biblioteca libbitcoin.

Utilizare: bx COMMAND [--help]

Info: Comenzile bx sunt:

- 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
```

Pentru mai multe informații, consultați documentația de utilizare [Pagina Bitcoin Explorer](#) și [Documentație Utilizatori Bitcoin Explorer](#).

## Exemple de Utilizare a Comenzii bx

Să ne uităm la câteva exemple de utilizare a comenziilor Bitcoin Explorer pentru a experimenta cu chei și adrese.

Generarea unei valori aleatorii "sămânță" folosind comanda `seed`, care utilizează generatorul de numere aleatoare ale sistemului de operare. Păsați sămânța la comanda `ec-new` pentru a genera o nouă cheie privată. Salvăm ieșirea standard în fișierul `private_key`:

```
$ bx seed | bx ec-new > private_key
$ cat private_key
73096ed11ab9f1db6135857958ece7d73ea7c30862145bcc4bbc7649075de474
```

Acum, generați cheia publică din acea cheie privată folosind comanda `ec-to-public`. Pasăm fișierul `private_key` la intrarea standard și salvăm ieșirea standard a comenziîntr-un nou fișier `public_key`:

```
$ bx ec-to-public < private_key > public_key
$ cat public_key
02fc46a6006a62dfdd2dbb2149359d0d97a04f430f12a7626dd409256c12be500
```

Putem reforma `public_key` ca adresă folosind comanda `ec-to-address`. Pasăm `public_key` la intrare standard:

```
$ bx ec-to-address < public_key
17re1S4Q8ZHypCP8Kw7xQad1Lr6XUzWUnkG
```

Cheile generate în acest mod produc un portofel nedeterministic de tip-0. Aceasta înseamnă că

fiecare cheie este generată dintr-o sămânță independentă. Comenzile Bitcoin Explorer pot genera, de asemenea, chei deterministic, în conformitate cu BIP-32. În acest caz, o cheie "master" este creată dintr-o sămânță și apoi extinsă deterministic pentru a produce un arbore de sub-chei, rezultând un portofel deterministic de tip-2.

În primul rând, folosim comenzile **seed** și **hd-new** pentru a genera o cheie master care va fi folosită ca bază pentru a obține o ierarhie de chei:

```
$ bx seed > seed
$ cat seed
eb68ee9f3df6bd4441a9feadec179ff1

$ bx hd-new < seed > master
$ cat master
xprv9s21ZrQH143K2BEhMYpNQoUvAgjEjArAVaZaCTgsaGe6LsAnwubeiTcDzd23mAoyizm9cApe51gNfLMkBq
kYoWWMCRwzfuJk8RwF1SVEpAQ
```

Acum folosim comanda **hd-private** pentru a genera o cheie "cont" întărită și o secvență de două chei private în cont:

```
$ bx hd-private --hard < master > account
$ cat account
xprv9vkDLt81dTKjwHB8fsVB5QK8cGnzveChzSrtCfvu3aMWvQaThp59ueufuyQ8Qi3qpjk4aKsbmbfxwcgS8P
YbgoR2NWHeLyvg4DhoEE68A1n

$ bx hd-private --index 0 < account
xprv9xHfb6w1vX9xgZyPNXVgAhPxSsEkeRcPHEUV5iJcVEsuUEACvR3NRY3fpGhcnbIDbvG4LgndirDsia1e9F
3DWPKX7Tp1V1u97HKG1FJwUpU

$ bx hd-private --index 1 < account
xprv9xHfb6w1vX9xjc8XbN4GN86jzNAZ6xHEqYxzbLB4fzHFd6VqCLPGRZFsdjsuMVERadbgDbziCRJru9n6tz
EWrASVpEdrZrFidt1RDfn4yA3
```

În continuare, folosim comanda **hd-public** pentru a genera secvența corespunzătoare de două chei publice:

```
$ bx hd-public --index 0 < account
xpub6BH1zcTukt1Fu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYPFJ3ve
zHz5wzaSW4FiGrseNDR4LKqTy

$ bx hd-public --index 1 < account
xpub6BH1zcTukt1Fx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWyMaMV1cn7nUPUkgQHPVXJVqsrA8xWb
GQDhohEcDFTEmvYzwRD7Juf8
```

Cheile publice pot fi de asemenea derivate din cheile private corespunzătoare folosind comanda **hd-to-public**:

```
$ bx hd-private --index 0 < account | bx hd-to-public
xpub6BH1zcTuktFu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYPFJ3ve
zHz5wzaSW4FiGrseNDR4LKqTy

$ bx hd-private --index 1 < account | bx hd-to-public
xpub6BH1zcTuktFx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWyMaMV1cn7nUPUkgQHPVXJVsra8xWb
GQDhohEcDFTEYMyYzwRD7Juf8
```

Putem genera un număr de chei practic nelimitat într-un lanț determinist, toate deriveate dintr-o singură sămânță. Această tehnică este folosită în multe aplicații de portofel pentru a genera chei care pot fi salvate și restabilite cu o singură valoare de sămânță. Acest lucru este mai ușor decât să faceți o copie de siguranță a portofelului de fiecare dată când o nouă cheie este creată cu toate cheile sale generate aleatoriu.

Sămânța poate fi codată folosind comanda **mnemonic-codare**:

```
$ bx hd-mnemonic < seed > words
adore repeat vision worst especially veil inch woman cast recall dwell appreciate
```

Sămânța poate fi apoi decodată folosind comanda **mnemonic-decode**:

```
$ bx mnemonic-decode < words
eb68ee9f3df6bd4441a9feadec179ff1
```

Codificarea mnemonică poate face sămânța mai ușor de înregistrat și chiar de amintit

<section data-type="index"/>

<section id="colophon" data-type="colophon"><h1>Casetă</h1>

<p>Animalul de pe coperta <em>Mastering Bitcoin</em> este o furnică tăietoare de frunze (<em>Atta colombica</em>). Furnica tăietoare de frunze (un nume non-generic) este o furnică tropicală, care crește ciuperci, endemică pentru America de Sud și Centrală, Mexic și sudul Statelor Unite. În afară de oameni, furnicile tăietoare de frunze formează cele mai mari și mai complexe societăți de animale de pe planetă. Sunt numite după felul în care mestecă frunzele, care servesc ca nutriție pentru grădina fungică.</p>

<p>Furnicile cu aripi, atât masculi, cât și femele, participă la o ieșire în masă din cuibul lor cunoscut sub numele de <em>revoada</em>, sau zbor nupțial. Femelele se imperechează cu mai mulți bărbați pentru a colecta cei 300 de milioane de spermă necesară pentru înființarea unei colonii. Femelele stochează, de asemenea, bucăți de miceliu din grădina ciupercii parentale în buzunarul infrabucal situat în cavitatea lor orală; ele le vor folosi pentru a începe propriile grădini fungice. Odată legată de pământ, femela își pierde aripile și înființează un bârlog subteran pentru colonia ei. Rata de succes a reginelor noi este scăzută: 2,5% stabilesc o colonie cu viață lungă.</p>

<p>Odată ce o colonie a ajuns la maturitate, furnicile sunt împărțite în caste în funcție de mărime, fiecare casă îndeplinind diferite funcții. De obicei, există patru caste: minime, cei mai mici

muncitori care au grijă tineri și de grădinile de fungus; minorii, puțin mai mari decât minimele, reprezintă prima linie de apărare pentru colonie și patrulează terenul din jur și atacă inamicii; mediae, taie frunzele și aduc fragmente de frunze în cuib; și majori, cele mai mari furnici muncitoare care acționează ca soldați, care apără cuibul împotriva intrușilor. Cercetări recente au arătat că majorii curăță și principalele trasee de hrănire și transportă obiecte voluminoase înapoi în cuib.</p>

<p>Multe dintre animalele de pe copertele O'Reilly sunt pe cale de dispariție; toate sunt importante pentru lume. Pentru a afla mai multe despre cum puteți ajuta, accesați [animals.oreilly.com](http://animals.oreilly.com/) .</p>

<p>Imaginea de copertă este din <em>Insecte din străinăta</em>. Fonturile de pe copertă sunt URW Typewriter și Guardian Sans. Fontul text este Adobe Minion Pro; fontul de la titlu este Adobe Myriad Condensed; iar fontul pentru cod este Ubuntu Mono de Dalton Maag.</p> </section>

[1] "Bitcoin: A Peer-to-Peer Electronic Cash System," Satoshi Nakamoto (<https://bitcoin.org/bitcoin.pdf>).