

E-COMMERCE

Wprowadzenie do Bitcoin

UWALNIANIE CYFROWYCH KRYPTOWALUT

Andreas M. Antonopoulos

Wprowadzenie do Bitcoin

Chcecie przyłączyć się do rewolucji technologicznej, która podbija świat finansów? *Wprowadzenie do Bitcoin* to Wasz przewodnik po pozornie złożonym świecie bitcoin prezentujący niezbędną wiedzę umożliwiającą udział w świecie walut internetowych. Niezależnie od tego, czy konstruujecie kolejną zabójczą aplikację, czy dokonujecie inwestycji w świeżo uruchomioną firmę albo po prostu chcecie zapoznać się z nową technologią, ten praktycznych podręcznik do Wasz niezbędny.

Bitcoin – pierwsza skutecznie zdecentralizowana waluta cyfrowa znajdująca się we wczesnej fazie rozwoju – już przyczyniła się do rozwoju wielomiliardowej gospodarki o charakterze globalnym. Jest ona dostępna dla wszystkich, którzy dysponują odpowiednim zakresem wiedzy oraz wykazują pasję uczestnika. *Wprowadzenie do Bitcoin* zawiera niezbędne informacje (z wyłączeniem pasji).

Książka zawiera:

- Ogólne wprowadzenie w tematykę kryptowalut — idealne dla osób nie będących technicznymi ich użytkownikami, inwestorów oraz osób zarządzających podmiotami gospodarczymi
- Podstawy techniczne bitcoin oraz kryptowalut dla deweloperów programów, inżynierów oraz architektów oprogramowania oraz systemów informatycznych
- Szczegółowe informacje na temat zdecentralizowanej sieci bitcoin, architektury typu *peer-to-peer*, cyklu transakcyjnego oraz zasad bezpieczeństwa
- Podstawy opracowania bitcoin oraz sieci blokowych uwzględniające także sieci, waluty i aplikacje alternatywne
- Anegdoty użytkowników, eleganckie analogie, przykłady oraz fragmenty kodów ilustrujące kluczowe koncepcje techniczne

Andreas M. Antonopoulos to ceniony specjalista w zakresie technologii oraz założyciel wielu firm, który stał się jedną z najpowszechniej znanych i szanowanych postaci w świecie bitcoin. Ten doskonaty mówca, nauczyciel oraz pisarz posiada niezwykły dar przekazywania złożonych tematów w sposób przystępny i zrozumiały. Andreas jest także konsultantem dla licznych nowopowstałych firm technologicznych i regularnie wygłasza wykłady podczas konferencji i różnego rodzaju wydarzeń społecznościowych na całym świecie.

“ Kiedy omawiam Bitcoin niekiedy pada pytanie: 'Ale jak to właściwie działa?' teraz mam już właściwą odpowiedź na to pytanie, ponieważ każdy, kto zapozna się z treścią książki „Wprowadzenie do Bitcoin” będzie dysponować dogłębną wiedzą na temat sposobu działania tego systemu oraz odpowiednimi narzędziami umożliwiającymi napisanie aplikacji kryptowalutowych następnej generacji.”

—Gavin Andresen
Główny Specjalista ds. Naukowych, Bitcoin Foundation

“ Książka napisana przez Andreasa pomoże Wam włączyć się a nurt rewolucji w oprogramowaniu przeznaczonym dla świata finansów.”

—Naval Ravikant
Współzałożyciel AngelList

Podziękowanie za opracowanie „Wprowadzenia do Bitcoin

“Kiedy omawiam Bitcoin niekiedy pada pytanie: 'Ale jak to właściwie działa?' teraz mam już właściwą odpowiedź na to pytanie, ponieważ każdy, kto zapozna się z treścią książki „Wprowadzenie do Bitcoin” będzie dysponować dogłębną wiedzą na temat sposobu działania tego systemu oraz odpowiednimi narzędziami umożliwiającymi napisanie aplikacji kryptowalutowych następnej generacji.”

— *Gavin Andresen*

Główny Specjalista ds. Naukowych, Bitcoin Foundation

“Bitcoin oraz technologie sieci blokowych stają się fundamentami Internetu nowej generacji. Największe i najsławniejsze umysły w Dolinie Krzemowej już nad nimi pracują. Książka napisana przez Andreas pozwoli Wam wziąć udział w rewolucji w oprogramowaniu dla świata finansów.”

— *Naval Ravikant*

Współzałożyciel AngelList

“Wprowadzenie do Bitcoin” to najlepszy techniczny materiał źródłowy poświęcony bitcoin obecnie dostępny na rynku. Patrząc wstecz możemy przyjąć, że bitcoin to najważniejsza z technologii ostatniej dekady. I właśnie dlatego niniejsza pozycja mieści się kanonie lektur obowiązkowych każdego programisty, zwłaszcza jeżeli specjalizuje się on w budowaniu aplikacji z protokołem bitcoin. Gorąco polecam.”

— *Balaji S. Srinivasan (@balajis)*

Partner generalny

Wprowadzenie do Bitcoin

Andreas M. Antonopoulos

Mastering Bitcoin

By Andreas M. Antonopoulos

Copyright c 2015 Andreas M. Antonopoulos LLC. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Książki publikowane przez O'Reilly mogą być nabywane w celach edukacyjnych, biznesowych lub do wykorzystania w celu promocji sprzedaży. Dostępne są także wersje online większości opublikowanych pozycji (<http://safaribooksonline.com>). W celu uzyskania bliższych informacji należy kontaktować się z naszym Działem Sprzedaży Korporacyjnej/Instytucjonalnej pod nr: 800-998-9938 lub na adres poczty elektronicznej: corporate@oreilly.com.

Wydawca: Mike Loukides and Allyson MacDonald

Kierownik Produkcji: Melanie Yarbrough

Redakcja: Kim Cofer

Korekta: Carla Thornton

Indeks: WordCo Indexing Services

Projekt okładki: Karen Montgomery

Układ treści: David Futato

Ilustracje: Rebecca Demarest

Grudzień 2014r.: Wydanie pierwsze

Historia zmian do wydania pierwszego:

2014-12-01: Wydanie pierwsze

Patrz: <http://oreilly.com/catalog/errata.csp?isbn=9781449374044> w celu zapoznania się z informacjami szczegółowymi.

Znak O'Reilly to zarejestrowany znak handlowy O'Reilly Media, Inc. *Mastering Bitcoin*, projekt okładki oraz związane znaki handlowe i oznaczenie stanowią znaki handlowe O'Reilly Media, Inc.

Oznaczenia stosowane przez producentów i sprzedawców w celu wyróżnienia swoich wyrobów uznaje się za znaki handlowe. Jeżeli oznaczenia te występują w niniejszej książce i O'Reilly Media, Inc. Posiada informacje dotyczące praw do znaku handlowego są one napisane wielkimi literami lub ich oznaczenia zaczynają się od wielkiej litery.

Pomimo starań ze strony autora oraz wydawcy w celu zapewnienia, że wszelkie informacje oraz instrukcje umieszczone w treści niniejszego opracowania są dokładne wydawca oraz autor zrzekają się odpowiedzialności za wszelkie błędy lub pominięcia z wyłączeniem, ale bez ograniczenia do odpowiedzialności za szkody wynikłe w związku z wykorzystaniem lub opieraniem się na treści niniejszej książki. Wykorzystanie informacji oraz instrukcji zawartych w niniejszej publikacji jest na wyjątkową odpowiedzialność i ryzyko czytelnika. Jeżeli podane w niniejszym materiale przykłady kodów lub innych technologii podlegają licencjom otwartym (*open source*) lub obciążone są prawami stron trzecich odpowiedzialność za wykorzystanie odpowiednich treści zgodnie z warunkami określonymi w tych licencjach lub bez naruszania wymienionych praw spoczywa na czytelniku.

ISBN: 978-1-449-37404-4

[LSI]

Spis treści

Wstęp.....	11
Opracowanie materiału dotyczącego Bitcoin.....	11
Odbiorcy.....	11
Dlaczego na okładce są owady?.....	11
Konwencje stosowane książce.....	12
Przykładowe kody.....	12
Wykorzystanie kodów przykładowych.....	13
Safari® Books Online.....	13
Jak się z nami skontaktować.....	14
Podziękowania.....	14
Wersja wstępna (materiały z GitHub).....	16
Skrócony słownik.....	16
ROZDZIAŁ 1.....	19
Czym jest Bitcoin?.....	19
Historia Bitcoin.....	21
Wykorzystanie Bitcoin, użytkownicy oraz ich historie.....	22
Początek.....	24
Szybki start.....	25
Pierwsze bitcoiny.....	26
ROZDZIAŁ 2.....	31
Transakcje, bloki, mining i łańcuchy blokowe.....	31
Prezentacja ogólna Bitcoin.....	31
Kupujemy filiżankę kawy.....	32
Transakcje bitcoinowe.....	34
Popularne formy transakcji.....	36
Konstrukcja transakcji.....	37
Otrzymywanie odpowiednich strumieni wejściowych.....	38
Tworzenie bloków wyjściowych.....	39
Dodawanie transakcji do księgi.....	40
Wysyłanie transakcji.....	40
Powielanie.....	40

Perspektywa Boba.....	41
Bitcoin Mining.....	41
Bloki transakcji miningowych.....	42
Wydatkowanie transakcji.....	44
ROZDZIAŁ 3.....	45
Rdzeń Bitcoin: oprogramowanie wzorcowe.....	45
Pierwsze uruchomienie oprogramowania podstawowego Bitcoin.....	45
Kompilacja Bitcoin Core w oparciu o kod źródłowy.....	46
Korzystanie z JSON-RPC API Bitcoin Core w linii poleceń.....	51
Informacje o statusie Bitcoin Core Client.....	53
Ustawienia portfela oraz szyfrowanie.....	53
Aktualizacja portfela, usuwanie oraz odzyskiwanie tekstu Plain-text Dump.....	54
Adres portfela i odbieranie transakcji.....	54
Sprawdzanie i odszyfrowywanie transakcji.....	56
Wyszukiwanie bloków.....	59
Tworzenie, podpisywanie oraz przekazywanie transakcji w oparciu o niezrealizowane kwoty wyjściowe	60
Alternatywni klienci, biblioteki oraz zestawy narzędzi.....	65
Libbitcoin i narzędzia sx.....	66
ROZDZIAŁ 4.....	69
Wstęp.....	69
Szyfrowanie przy użyciu klucza publicznego oraz kryptowaluty.....	69
Klucze prywatne i publiczne.....	70
Klucze prywatne.....	70
Generowanie klucza prywatnego w oparciu o liczbę losową.....	71
Klucze publiczne.....	72
Wyjaśnienie szyfrowania krzywą eliptyczną.....	72
Generowanie klucza publicznego.....	75
Adresy bitcoin.....	76
Szyfrowanie Base58 i Base58Check.....	78
Formaty kluczy.....	81
Wdrażanie kluczy i adresów w Python.....	85
Portfele.....	88

Portfele niedeterministyczne (losowe).....	88
Portfele deterministyczne (źródłowe).....	88
Kody mnemotechniczne.....	89
Zaawansowane klucze i adresy.....	98
Szyfrowane klucze prywatne (BIP0038).....	98
Funkcja hashująca Pay-to-Script (P2SH) i adresy Multi-Sig.....	99
Adresy vanity.....	101
Portfele papierowe.....	105
Rozdział 5.....	108
Wstęp.....	108
Cykl transakcji.....	108
Tworzenie Transakcji.....	108
Rozprzestrzenianie transakcji po sieci Bitcoin.....	109
Rozprzestrzenianie transakcji w sieci bitcoin.....	109
Struktura transakcji.....	110
Bloki wyjściowe transakcji.....	112
Warunki wydatkowania (ograniczenia).....	115
Opłaty transakcyjne.....	119
Dodawanie opłat do transakcji.....	119
Transakcja łańcuchowe i sieroce.....	120
Skrypty transakcji oraz język skryptów.....	121
Budowa Skryptu (blokowanie i odblokowanie).....	122
Język skryptu.....	123
Niekompletność Turinga.....	126
Bezstatusowa weryfikacja.....	126
Standardowe transakcje.....	126
Pay-to-public-key.....	128
Skrypty multi-signature.....	129
Blok danych wyjściowych (OP.RETURN).....	130
Pay-by-Script-Hash (P2SH).....	132
Adresy Pay-to-script-hash.....	134
Korzyści z pay-to-script-hash.....	135
Realizacja (umarzanie) skryptu i walidacja isStandard.....	135

Rozdział 6.....	136
Sieć Bitcoin.....	136
Węzły - rodzaje i role.....	137
Rozszerzenie sieci.....	138
Wykrywanie sieci:.....	141
Pełne Węzły.....	144
Wymiana „Inwentarza”.....	145
Uproszczone weryfikacje płatności (SPV) węzłów.....	145
Filtry Bloom.....	148
Filtr Bloom i aktualizacje inwentarza.....	152
Zbiorniki transakcyjne.....	153
Wiadomości Alertowe.....	153
Rozdział 7.....	155
Łańcuch blokowy.....	155
Wprowadzenie.....	155
Struktura bloku.....	156
Nagłówek bloku.....	157
Identyfikatory bloku: Hash nagłówka bloku i wysokość bloku.....	157
Blok genezy.....	158
Łączenie bloków w łańcuchy.....	159
Drzewko Merkla.....	160
Drzewka skrótów (Merklego) i SPV.....	166
Rozdział 8.....	169
Górnictwo i konsensus.....	169
Wstęp.....	169
Ekonomia Bitcoin i tworzenie waluty.....	170
Consensus decentralizacji.....	172
Niezależna weryfikacja transakcji.....	172
Data transakcji, opłaty i priorytet.....	175
Transakcja generowania.....	176
Wynagrodzenie Coinbase i opłaty.....	178
Struktura transakcji generowania.....	179
Dane Coinbase.....	179

Konstruowanie nagłówka bloku.....	181
Wydobycie bloku.....	182
Algorytm Proof-of-work.....	182
Trudność reprezentacji.....	187
Trudność docelowa i przekierowywanie.....	187
Skuteczne wydobycie bloku.....	189
Weryfikacja (validacja) nowego bloku.....	189
Gromadzenie oraz wybór bloków łańcucha.....	190
Rozwidlenia łańcuchów blokowych.....	191
Mining i wyścig hashowania.....	195
Rozwiązywanie kwestii dodatkowych wartości jednorazowych.....	197
Pula minibowa.....	197
Pule zarządzane.....	199
P2Pool.....	199
Ataki na consensus.....	200
ROZDZIAŁ 9.....	203
Podział taksonomiczny walut i łańcuchów alternatywnych.....	203
Platformy meta coin.....	204
Colored Coins.....	204

Wstęp

Opracowanie materiału dotyczącego Bitcoin

Z bitcoin po raz pierwszy zetknąłem się w połowie 2011 roku. Moja pierwsza reakcja była mniej więcej taka: "Fuuu! Cyfrowa kasa!" po czym temat zniknął z mojego życia na kolejne sześć miesięcy, a ja nie doceniłem jego roli. Podobną reakcję widziałem także u osób, które uznaję za wybitnie inteligentne, co stanowi dla mnie pewne pocieszenie. Ale kiedy natknąłem się na bitcoin po raz drugi, podczas omawiania listy mailingowej, postanowiłem zbadać treść białej księgi opracowanej przez Satoshi Nakamoto, aby zapoznać się z opinią osoby uznawanej powszechnie za autorytet i zrozumieć o co w tym chodzi. Wciąż pamiętam chwilę, kiedy zakończyłem lekturę tych dziewięciu stron i uświadomiłem sobie, że bitcoin to nie tylko waluta cyfrowa, ale sieć zaufania, która tak naprawdę może się stać podstawą czegoś o wiele większym niż tylko pieniądz. Uświadomienie sobie, że "to nie pieniądz, ale zdecentralizowana sieć powiernicza" spowodowało, że udało się w czterodniową podróż po wiedzę, w trakcie której dosłownie pochłaniałem każdy skrawek informacji na temat bitcoin, jaki udało mi się znaleźć. Podekscytowany, niemal popadając w obsesję spędzałem 12 lub więcej przed monitorem czytając, pisząc, kodując oraz zdobywając tyle wiedzy, ile tylko zdolałem przyswoić. Kończąc tę ciężką pracę byłem o jakieś 10 kilogramów lżejszy wskutek nieregularnego odżywiania, ale zdeterminowany, aby poświęcić się pracy nad bitcoin.

Dwa lata później założywszy kilka małych firm (startupy), których głównym zadaniem było badanie różnego rodzaju usług i produktów związanych z technologią bitcoin doszedłem do wniosku, że czas napisać moją pierwszą książkę. Bitcoin było to coś, dla czego rzuciłem się w wir pracy twórczej i co całkowicie objęło panowanie nad moim umysłem; była to najbardziej ekscytująca technologia, z jaką się spotkałem od czasu Internetu. A teraz nadszedł także czas, aby podzielić się swoją pasją dla tej niezwykłej technologii z szerszą publicznością.

Odbiorcy

Niniejsza książka jest przeznaczona przede wszystkim dla osób zajmujących się tworzeniem kodów źródłowych. Jeżeli znacie język programowania, to ta książka pokaż Wam w jaki sposób działają waluty kryptograficzne, jak z nich korzystać oraz w jaki sposób tworzyć oprogramowanie, które będzie z nimi współdziałać. Pierwszych kilka rozdziałów jest poświęconych to także dogłębne wprowadzenie do technologii bitcoin dla tych, którzy nie mają pojęcia o kodowaniu, a chcą zrozumieć zawiłości tego systemu oraz zasady działania kryptowalut.

Dlaczego na okładce są owady?

Mrówka parasolowa to gatunek, który charakteryzuje się złożonym zachowaniem występującym w grupie, za to występując pojedynczo działa w oparciu od zestaw bardzo prostych reguł współpracy społecznego w oparciu o chemiczne substancje zapachowe (feromony). Według Wikipedii: "Obok ludzi, mrówki parasolowe stanowią największą i najbardziej złożoną społeczność zwierząt na ziemi". Mrówki te w zasadzie nie żywią się liśćmi, ale wykorzystują je do hodowania grzybów, które stanowią główne źródło pożywienia kolonii, w których żyją. Pojmujecie? Te mrówki prowadzą własną hodowlę!

Chociaż mrówki tworzą społeczeństwo kastowe z królową, której głównym zadaniem jest rozmnażanie, nie posiadają żadnej władzy nadzędnej lub lidera kolonii. Bardzo inteligentne i zaawansowane zachowania tych kolonii liczących wiele milionów członków to skutek wzajemnych interakcji pomiędzy członkami tej sieci społecznej.

Natura uczy nas, że systemy zdecentralizowane mogą wykazywać się wyższą odpornością oraz rosnącą złożonością i nieprawdopodobnym poziomem zaawansowania bez posiadania władzy centralnej, hierarchii lub złożonych elementów składowych.

Bitcoin to bardzo zaawansowania, zdecentralizowane sieć wspierająca niezliczone liczby procesów i transakcji finansowych. Każdy węzeł sieci bitcoin działa w oparciu o kilka prostych reguł matematycznych. Wzajemne oddziaływanie pomiędzy nimi prowadzi do powstania złożonych zachowań, a nie dowolnej, z natury skomplikowanej całości lub trustu w pojedynczym węźle. Podobnie jak w przypadku kolonii mrówek, sieć bitcoin jest to wytrzymała sieć prostych węzłów źródłowych stosujących się do prostych zasad, które współpracując ze sobą mogą dokonywać zadziwiających rzeczy bez potrzeby centralnej koordynacji.

Konwencje stosowane książce

W książce stosowane są następujące konwencje zapisu:

Italika

Oznacza nowe terminy, URL, adresy email, nazwy plików oraz rozszerzenia nazw plików.

Stała szerokość

Wykorzystywana do tworzenia list programów oraz wewnętrz akapitów, w odniesieniu do elementów programów takich, jak zmienne lub nazwy funkcji, bazy danych, typy danych, zmienne środowiskowe, oświadczenia oraz słowa kluczowe.

Stała szerokość pogrubiona

Oznacza polecenia lub inny rodzaj tekstu, który powinien być przepisany dosłownie (bez zmian) przez użytkownika.

Stała szerokość italicika

Oznacza tekst, który należy zastąpić wartościami podstawionymi przez użytkownika lub wynikającymi z kontekstu.



Ikonka ta oznacza podpowiedź, sugestię lub uwagę o charakterze ogólnym.



Ikonka ta oznacza ostrzeżenie lub ważną informację.

Przykładowe kody

Przykłady są podane w Python, C++ oraz linii poleceń system operacyjnego typu Unix, np.

Linux lub Mac OS X. Wszystkie fragment kodów są dostępne w repozytorium [GitHub](#), w podkatalogu kodów głównego repozytorium. Opracujcie skrypty, wypróbujcie przykładowe kody lub przekażcie poprawki za pośrednictwem GitHub.

Wszystkie fragmenty kodów można powiełać na większości systemów operacyjnych przy znikomej ilości instalacji niezbędnych kompilatorów oraz tłumaczy na odpowiednie języki. W razie potrzeby, udzielamy

podstawowych instrukcji instalacji oraz prezentujemy przykłady krok po kroku wyników, jakie powinno się uzyskać dzięki przestrzeganiu instrukcji.

Niektóre z przykładowych kodów oraz kody wyjściowe zostały preformatowane w celu wydruku. We wszystkich tych przypadkach, linie zostały podzielone ukośnikiem (\), po którym następuje znak nowej linii. Dokonując transkrypcji przykładów należy usunąć te dwa znaki i połączyć ponownie linie, aby uzyskać wyniki identyczne z przykładami.

We wszystkich fragmentach kodów, jeżeli było to możliwe, podano rzeczywiste wartości i obliczenia, aby umożliwić korzystającym tworzenie własnych rozwiązań na bazie przykładów oraz uzyskiwania takich samych wyników w dowolnie skonstruowanych kodach w celu obliczenia tych samych wartości. Na przykład, klucze prywatne oraz odpowiadające im klucze publiczne oraz adresy są rzeczywiste. Przykładowe transakcje, bloki oraz informacje źródłowe dotyczące łańcuchów blokowych zostały wprowadzone do rzeczywistego łańcucha bitcoin i stanowią element księgi publicznej, z którą można zapoznać się w dowolnym systemie bitcoin.

Wykorzystanie kodów przykładowych

Niniejsza książka ma służyć czytelnikom pomocą w realizacji stawianych im zadań. Generalnie, jeżeli w niniejszej książce udostępniany jest kod przykładowy, to może on być wykorzystywany w opracowywanej przez Was dokumentacji lub programach. Nie ma potrzeby kontaktowania się z nami w celu uzyskania zgody, chyba że planujecie powielić znaczną część kodu.

Na przykład opracowanie programu w oparciu o duże fragmenty kodu przedstawionego w niniejszej książce nie wymaga uzyskania zgody. Sprzedaż lub dystrybucja CD-ROM-u z przykładami zaczerpniętymi z publikacji O'Reilly wymaga uzyskania zgody. Udzielenie odpowiedzi powołując się na niniejszą książkę oraz podawania przykładu kodu źródłowego nie wymaga zgody. Wykorzystanie dużej części przykładowego kodu zawartego w treści niniejszej książki w dokumentacji opracowanego przez Was produktu wymaga uzyskania zgody.

Doceniamy, ale nie wymagamy powołania się na źródło. Z reguły jednak podaje się tytuł, autora, wydawcę oraz numer ISBN. Na przykład: "Wprowadzenie do systemu *Bitcoin*", Andreas M. Antonopoulos (O'Reilly). Copyright 2015 Andreas M. Antonopoulos, 978-1-449-37404-4."

Wybrane wersje niniejszej książki są dostępne w ramach licencji otwartej, np. CC-BYNC (creativecommons.org) i w takiej sytuacji stosuje się postanowienia tej licencji.

Jeżeli Waszym zadaniem zakres wykorzystania przykładowych kodów wykracza poza zakres zgody udzielonej powyżej, możecie skontaktować się z nami tutaj: permissions@oreilly.com.

Safari® Books Online

Safari Books Online to biblioteka cyfrowa udostępniająca pozycje zawierające materiały profesjonalne w formie wideo oraz książkowej opracowane przez czołowych światowych autorów w specjalizujących się w rozwiązaniach technologicznych i biznesowych.

Specjaliści w dziedzinie technologii, programiści, projektanci sieci oraz osoby działające w biznesie i osoby twórczo zajmujące się różnymi dziedzinami wykorzystują Safari Books Online jako swoje główne źródło informacji o charakterze badawczym, materiałów do nauki, rozwiązywania problemów oraz szkoleń certyfikujących.

Safari Books Online oferuje szeroką gamę **mieszanek produktów** oraz programów wyceny opracowanych z myślą o **organizacjach**, agencjach rządowych oraz **osobach fizycznych**. Subskrybenci uzyskują dostęp do tysięcy książek, materiałów szkoleniowych wideo oraz nieopublikowanych jeszcze rękopisów, które znajdują się w jednej bazie, w której umieszczone są przez wydawców takich, jak O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, 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, Course Technology i dziesiątki **innych**. W celu uzyskania bliższych informacji na temat Safari Books Online prosimy o kontakt [online](#).

Jak się z nami skontaktować

Wszelkie uwagi i pytania dotyczące tej książki prosimy kierować do wydawcy:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (w Stanach Zjednoczonych i Kanadzie)

707-829-0515 (linia międzynarodowa lub lokalna)

707-829-0104 (faks)

Książka ta posiada swoją stronę internetową, na której publikowane są erraty, przykłady oraz inne informacje dodatkowe. Strona ta dostępna jest pod tym adresem: http://bit.ly/mastering_bitcoin.

Komentarze lub pytania techniczne dotyczące niniejszej publikacji można przesyłać pocztą elektroniczną na adres: bookquestions@oreilly.com.

W celu uzyskania bliższych informacji na temat naszych książek, kursów, konferencji oraz najnowszych wiadomości prosimy zatrzymać się na naszą stronie: <http://www.oreilly.com>.

Jesteśmy także na Facebooku: <http://facebook.com/oreilly>

Możecie nas także śledzić na Twitterze: <http://twitter.com/oreillymedia>

Lub oglądać na YouTube: <http://www.youtube.com/oreillymedia>

Podziękowania

Niniejsza książka to efekt pracy i wkładu wielu osób. Jestem niezmiernie wdzięczny za pomoc, która otrzymałem od przyjaciół, kolegów, a nawet osób zupełnie obcych, które wzięły udział w procesie opracowywania ostatecznej wersji tego technicznego materiału poświęconej kryptowalutom oraz systemowi bitcoin.

Nie jest możliwe rozróżnienie pomiędzy technologią bitcoin, a społecznością bitcoin; niniejszy materiał jest równym stopniu produktem tej społeczności i jednocześnie materiałem dotyczącym tej technologii. Do pracy nad tą publikacją zachęcała mnie, podtrzymywała na duchu oraz nagrodziła całą społeczność bitcoin – od samego początku aż po jej efekt końcowy. Ponadto, niniejszy materiał umożliwił mi stanie się częścią tej wspaniałej społeczności w okresie ostatnich dwóch lat i nie wiem jak wszystkim tym ludziom dziękować za to, że stało się to moim udziałem. Zbyt wiele osób musiałbym wymienić tutaj z nazwiska – osób, które spotkałem podczas konferencji, wydarzeń, seminariów, spotkań, imprez przy pizzy, a także zgromadzeń z ograniczoną liczbą uczestników oraz tych wszystkich, którzy kontaktowali się ze mną za pośrednictwem Twittera, na portalach reddit, bitcointalk.org, czy GitHub przez co wywarli wpływ na ostateczny kształt tej książki. Każdy

pomysł, analogia, pytanie, odpowiedź oraz wyjaśnienia zawarte w treści tej publikacji w pewnym momencie wynikały z inspiracji, były testowane lub korygowane w drodze moich interakcji z ta społecznością. Dziękuję za udzielone mi wsparcie; bez Was książki ta nigdy nie powstały. Jestem Wam dozgannie wdzięczny.

Moja droga do autorstwa zaczęła się na długo przed napisaniem pierwszej książki naturalnie. Moim pierwszym językiem (oraz językiem mojej edukacji) był grecki w związku z tym na pierwszym roku studiów musiałem zapisać się na dodatkowy kurs wspomagający z pisania. Mojej nauczycielce od pisania, pani Diana Kordas zauważam moje umiejętności i pewność jaką zdziałałem wtedy sobie wyrobić. Później, pracując zawodowo, doskonaliłem umiejętności w zakresie tekstów technicznych dotyczących centrów danych pisząc artykuły dla magazynu *Network World*. Jestem także winien podziękowanie Johnowi Dix oraz Johnowi Gallant, którzy jako pierwsi zatrudnili mnie jako publicystę w *Network World*, mojemu redaktorowi Michaelowi Cooney'owi oraz mojej koleżance Johna Till Johnson, która edytowała moje artykuły i dopasowywała do wydania. Pisanie 500 słów tygodniowo przez cztery lata pozwoliło mi zdobyć wystarczające doświadczenie, aby w końcu przystąpić do pracy pisarskiej. Dziękuję także Jean de Vera za ciepłe słowa zachęty do podjęcia prób pisarskich i za to, że we mnie wierzyła i przekonała mnie, że nienapisana książka już stanowiła część mnie samego.

Pragnę także wyrazić wdzięczność tym, którzy mnie wspierali, kiedy przedstawałem swoją książkę w O'Reilly, udzielając mi referencji i zapoznając się krytycznie z opracowanym przeze mnie materiałem. W szczególności pragnę podziękować Johnowi Gallant,

Gregory'emu Ness, Richardowi Stiennon, Joelowi Snyder, Adamowi B. Levine, Sandrze Gittlen, Johnowi Dix, Johnie Till Johnson, Rogerowi Ver oraz Jonowi Matonis. Szczególne podziękowania należą się Richardowi Kagan i Tymonowi Mattoszko, którzy weryfikowali pierwsze wersje książki oraz Matthew Owain Taylorowi, który dokonywał ostatecznej redakcji.

Dziękuję także Cricket Liu, autorowi wydanych przez O'Reilly pozycji *DNS and BIND*, który wprowadził mnie do wydawnictwa O'Reilly. Pragnę wyrazić swoją wdzięczność Michaelowi Loukides oraz Allyson MacDonald z O'Reilly, którzy poświęcili wiele miesięcy pomagając mi stworzyć tę książkę. Szczególnie Allyson wykazywała się ogromną cierpliwością, kiedy nie dotrzymywałem terminów i spóźniałem się z przekazywaniem kolejnych partii materiału w związku z różnymi okolicznościami nieuwzględnionymi w naszym harmonogramie, a napisanymi przez życie.

Kilka pierwszych wersji pierwszych rozdziałów stanowiło największe wyzwanie, ponieważ bitcoin sam sobie jest bardzo trudnym tematem do przedstawienia. Zawsze, kiedy wybierałem jeden z wątków dotyczących technologii bitcoin okazywało się, że muszę przedstawić temat w całości. Niestannie blokowałem się i popadałem w przygnębienie usiłując przedstawić temat w przystępny sposób i podejmując próbę opisania tak skomplikowanej kwestii technicznej. W końcu postanowiłem przybliżyć genezę bitcoin powołując się na historię ludzi wykorzystujących tę technologię dzięki, co znacznie uprościło proces pisarski. Wiele zauważam mojemu przyjacielowi i mentorowi - Richardowi Kagan, który pomógł mi w opracowaniu materiału, chwilach zwątpienia i blokady twórczej oraz Pameli Morgan, która czytywała pierwsze wersje każdego kolejnego rozdziału i zadawała trudne pytania, aby dokonać odpowiednich korekt. Chcę także podziękować programistom z grupy San Francisco Bitcoin Developers Meetup Group oraz Taariqowi Lewi – założycielowi grupy za pomoc w testowaniu pierwszych materiałów.

W trakcie przygotowywania niniejszej książki, pierwsze szkice udostępniłem na GitHub z prośbą o zgłoszenie uwag. W odpowiedzi otrzymałem ponad sto komentarzy, sugestii, sprostowań oraz dodatkowych materiałów - wszystkie są wymienione i opatrzone podziękowaniami w części „[Wersja wstępna \(materiały z GitHub\)](#)”. Szczególne podziękowania należą się Minh T. Nguyenowi, który dobrowolnie podjął się zarządzania materiałami przekazywanymi za pośrednictwem GitHub i jednocześnie przekazał liczne opracowane przez siebie materiały. Chcę także wyrazić wdzięczność Andrew Nauglerowi za projekty infograficzne.

Po napisaniu książki została ona poddana kilku rundom weryfikacji technicznej i w związku z tym chcę raz jeszcze podziękować Cricketowi Liu oraz Lorne Lantz za ich staranne weryfikacje, zgłoszone uwagi oraz wsparcie.

Niektórzy z programistów bitcoin przekazali przykładowe kody, weryfikacje, komentarze, uwagi oraz słowa zachęty. Wielkie podziękowania kieruję do Amir Taaki za przekazanie fragmentów przykładowych kodów oraz cenne uwagi; Vitalikowi Buterin oraz Richardowi Kiss za pomoc w zadaniach związanych z krzywa eliptyczną oraz przekazanie kodów; Gavinowi Andresen za poprawki, uwagi oraz słowa zachęty i Michalisowi Kargakis za uwagi, materiały oraz recenzję w btcd.

Mojej Mamie - Teresie zawsze dzięczę miłość do słowa oraz książek, które zajmowały wszystkie ściany w naszym domu. To właśnie Ona kupiła mi pierwszy komputer w 1982 pomimo, że sama siebie określała jako dedykowanego technofoba. Mój Tata - Menelaos, inżynier budowlany, który swoja pierwsza książkę wydał w wieku lat 80 był tym, który nauczył mnie logicznego i analitycznego myślenia oraz uwielbienia dla nauki i inżynierii.

Dziękuję wszystkim, którzy wsparli mnie na tej drodze.

Wersja wstępna (materiały z GitHub)

Wiele osób przekazało swoje uwagi, korekty oraz uzupełnienia do pierwszej wersji materiału za pośrednictwem GitHub. Dziękuję wszystkim, którzy przyczynili się do powstania tej książki. Poniżej znajduje się lista tych osób wraz z identyfikatorami GitHub w nawiasach:

- Minh T. Nguyen, GitHub contribution editor (enderminh)
- Ed Eykholt (edeykholt)
- Michalis Kargakis (kargakis)
- Erik Wahlstrom (erikwam)
- Richard Kiss (richardkiss)
- Eric Winchell (winchell)
- Sergej Kotliar (ziggamon)
- Nagaraj Hubli (nagarajhubli)
- ethers
- Alex Waters (alexwaters)
- Mihail Russu (MihailRussu)
- Ish Ot Jr. (ishotjr)
- James Addison (jayaddison)
- Nekomata (nekomata-3)
- Simon de la Rouviere (simondlr)
- Chapman Shoop (belovachap)
- Holger Schinzel (schinzelh)
- effectsToCause (vericoin)
- Stephan Oeste (Emzy)
- Joe Bauers (joebauers)
- Jason Bisterfeldt (jbisterfeldt)
- Ed Leafe (EdLeafe)

Skrócony słownik

Przedstawiony poniżej skrócony słownik zawiera wiele terminów dotyczących bitcoin. Zwroty te są wykorzystywane w książce, więc warto go sobie zaznaczyć i wykorzystywać w przyszłości.

adres

Adres bitcoin wygląda tak: 1DSrfJdB2AnWaFNgSbv3MZC2m74996JafV. Składa się ze strumienia liter i liczb zaczynających się od "1" (liczby). Podobnie jak ma to miejsce przypadku adresu email, można podawać ten adres jako adres do przekazywania.

bip

Bitcoin Improvement Proposals (Propozycje Usprawnień Bitcoin). Zestaw propozycji zgłoszonych przez członków społeczności bitcoin w celu poprawy systemu. Na przykład, BIP0021 to propozycja dotycząca ulepszenia jednolitego identyfikatora zasobów bitcoin (URI).

bitcoin

Nazwa jednostki walutowej (monety), sieci oraz oprogramowania.

blok

Grupa transakcji oznaczonych znacznikiem czasu oraz „odciskiem palca” poprzedniego bloku. Nagłówek bloku jest hash-owany, aby stanowić dowód wykonanego zadania jednocześnie walidując przeprowadzone transakcje. Ważne bloki są dodawane do głównych łańcuchów blokowych w drodze konsensusu sieciowego.

blok źródłowy

Pierwszy blok w łańcuchu wykorzystywany do inicjalizacji kryptowaluty.

hash

Cyfrowy odcisk palca binarnych danych wejściowych.

łańcuch blokowy (blockchain)

Lista zwalidowanych bloków, z których każdy jest połączony z poprzednim, aż do bloku źródłowego.

miner (górnik)

Węzeł sieciowy, który znajduje ważny dowód sieciowy wykorzystywany przez nowe bloki poprzez powtarzający się proces hashowania (mieszania).

nagroda

Kwota dodana do każdego nowego bloku jako nagroda ze strony sieci dla górnika, który znalazł rozwiązanie dla dowodu wykonania zadania Proof-Of-Work. Obecnie jest to 25BTC dla każdego bloku.

opłaty

Nadawca transakcji często dodaje opłatę na rzecz sieci za przetworzenie zgłoszonej transakcji. W większości przypadków wymagane jest opłata minimalna w wysokości 0,5 mBTC.

portfel

Oprogramowanie, w którym przechowywane są adresy bitcoin oraz tajne klucze. Może być wykorzystywane do wysyłania, odbierania oraz przechowywania bitcoinów.

potwierdzenia

Kiedy transakcja zostaje umieszczona w bloku, otrzymuje jednorazowe potwierdzenie. Jeżeli do danego łańcucha dodany jest kolejny blok, transakcja otrzymuje dwa potwierdzenia, itd.

Sześć lub więcej potwierdzeń uznaje się za wystarczający dowód na nieodwracalność transakcji.

proof of work

Zestaw danych, którego znalezienie wymaga dużego nakładu obliczeniowego. W przypadku bitcoin, górnicy muszą znaleźć rozwiązanie liczbowe zgodne z algorytmem SHA256, który spełnia ogólnosieciowy cel utrudnienia.

siec

Sieć wzajemnej wymiany informacji przekazująca informacje o transakcjach i blokach do każdego węzła bitcoin w sieci.

tajny klucz (także prywatny)

Tajna liczba otwierająca bitcoin'y wysłane pod odpowiednie adresy. Tajny klucz wygląda następująco: 5J76sF8L5jTzE96r66Sf8cka9y44wdpJjMwCxR3tzLh3ibVPxh.

transakcja

W najprostszym ujęciu jest to transfer bitcoinów z jednego adresu do drugiego. Mówiąc nieco dokładniej jest to transakcja z przypisaną strukturą danych wyrażająca transfer wartości. Transakcje są przekazywane za pośrednictwem sieci bitcoin, gromadzone przez górników, dołączane do bloków i utrwalane w łańcuchu bloków.

utrudnienia

Ogólnosieciowe ustawienia kontrolujące ilość obliczeń niezbędnych do wygenerowania potwierdzenia wykonania zadania.

utrudnienia docelowe

Utrudnienia, po osiągnięciu których wszystkie obliczenia w sieci natrafią na bloki w przybliżeniu co ok.10 minut.

zmiana utrudnień docelowych

Ogólnosieciowa zmiana obliczeń utrudnień, które występują co 2 106 bloków uwzględniająca moc mieszania poprzednich 2 106 bloków.

ROZDZIAŁ 1

Wstęp

Czym jest Bitcoin?

Bitcoin to zestaw koncepcji i technologii, który stanowi podstawę ekosystemu waluty cyfrowej. Jednostkami tej waluty są bitcoiny, które są wykorzystywane do przechowywania i przenoszenia wartości pomiędzy uczestnikami sieci bitcoin. Uczestnicy ci komunikują się ze sobą wykorzystując protokół bitcoin głównie w Internecie, chociaż dopuszczalne jest także wykorzystanie innych sieci. Stos protokołów bitcoin dostępny w postaci oprogramowania typu open source, może działać na różnego typu urządzeniach obliczeniowych takich jak laptopy czy smartfony, co czyni technologie powszechnie dostępną.

Użytkownicy mogą dokonywać transferu bitcoinów w sieci i realizować wszystkie funkcje, które można realizować wykorzystując waluty konwencjonalne w tym transakcje kupna i sprzedaży towarów, przelewy pieniężne dla osób fizycznych oraz organizacji, a nawet udzielać kredytów. Bitcoiny można nabywać, sprzedawać oraz dokonywać ich wymiany na inne waluty w specjalnych kantorach. Pod pewnymi względami Bitcoin to doskonały środek płatniczy dla potrzeb Internetu, ponieważ jest szybki, bezpieczny i transgraniczny. W odróżnieniu od walut konwencjonalnych, bitcoiny są produktem całkowicie wirtualnym. Nie występują pod żadną postacią fizyczną, ani nawet, jako monety cyfrowe jako takie. Monety są dorozumianym elementem transakcji, której towarzyszy transfer wartości pomiędzy nadawca, a odbiorcą. Użytkownicy bitcoinów posiadają własne klucze, które przypisują im własność transakcji oraz umożliwiają potwierdzenie stanu posiadania w sieci bitcoin, odblokowując wartości w celu wydatkowania lub przeniesienia na rzecz nowego odbiorcy. Klucze te są często przechowywane w portfelu cyfrowym na komputerach poszczególnych użytkowników. Posiadanie klucza odblokowującego transakcję stanowi jedyną przesłankę do wydatkowania bitcoinów dając użytkownikowi pełną kontrolę.

Bitcoin jest dystrybuowany w systemach typu peer-to-peer. Dzięki temu nie ma potrzeby stosowania „centralnego” serwera lub punktu kontroli. Bitcoiny są tworzone w procesie nazywanym “mining”, który jest swoistym wyścigiem w celu znalezienia rozwiązania dla problemu matematycznego w trakcie przetwarzania transakcji bitcoin. Każdy uczestnik sieci bitcoin (tzn. każda osoba wykorzystująca urządzenie, a n którym działa pełny stos protokołów bitcoin) może wykonywać zadania w charakterze minera (górnika) wykorzystując moc obliczeniową swojego komputera w celu zweryfikowania i zarejestrowania transakcji. Średnio co 10 minut, jedna z osób jest w stanie dokonać walidacji transakcji, które miały miejsce w ciągu ostatnich 10 minut, za co otrzymuje całkowicie nowe bitcoiny. Zasadniczo, proces bitcoin mining polega na decentralizacji emisji waluty oraz funkcji rozliczeniowych pełnionych przez bank centralny, które zastępuje swoistym globalnym wyścigiem - konkurencją.

Protokół bitcoin składa się z wbudowanych algorytmów, które regulują funkcje mining w całej sieci. Trudność przetwarzania, które każdy miner musi wykonać — aby skutecznie zarejestrować blok transakcji w sieci bitcoin — jest regulowana w sposób dynamiczny w taki sposób, że, średnio, co 10 minut ktoś odnosi sukces niezależnie od tego ilu minerów (oraz jednostek CPU) pracuje nad wybranym zadaniem jednocześnie. Protokół dokonuje zwalnia o połowę prędkość tworzenia nowych bitcoins co cztery lata oraz ogranicza łączną liczbę Bitcoinów, które można będzie utworzyć do stałej liczby 21 milionów monet. W efekcie liczba dostępnych w obiegu bitcoinów ma przebieg zgodny z łatwo przewidywalną krzywą, zgodnie z którą liczba 21 milionów zostanie osiągnięta w roku 2140. Z uwagi na malejącą emisję bitcoinów w perspektywie długookresowej bitcoiny mają charakter deflacyjny. Co więcej, nie można doprowadzić do inflacji bitcoinów poprzez „dodrukowywanie” nowych pieniędzy ponad przewidywaną wartość emisji.

Tak właściwie bitcoin to także nazwa protokołu, sieci oraz rozproszonych innowacji obliczeniowych. Waluta bitcoin to tak naprawdę jedynie pierwsze zastosowanie tych unowocześnienie. Jako programista postrzegam bitcoin jako produkt pokrewny internetowi pieniężnemu (finansowemu) sieci służącej propagowaniu wartości oraz zabezpieczającej własność wartości cyfrowych za pośrednictwem dystrybuowanych mocy obliczeniowych. Bitcoin to więcej niż możemy zauważać na pierwszy rzut oka.

Niniejszy rozdział zaczyna się od wyjaśnienia wybranych kluczowych koncepcji oraz terminologii, niezbędnego oprogramowania oraz wykorzystania bitcoinów w nieskomplikowanych transakcjach. W kolejnych rozdziałach omawiamy kolejne warstwy technologiczne, które umożliwiają stosowanie bitcoinów oraz omawiamy mechanizmy wewnętrzne sieci bitcoin oraz protokoły.

Waluty cyfrowe zanim pojawiły się bitcoiny

Pojawienie się cyfrowych walut jest ściśle powiązane z rozwojem kryptografii. Nie dziwi to specjalnie, jeśli weźmie się pod uwagę zasadnicze wyzwania związane z wykorzystywaniem bitów jako reprezentacji wartości, które można wymieniać na towary i usługi. Dwa podstawowe pytania, które zadają sobie użytkownicy walut cyfrowych są następujące:

1. Czy mogę mieć pewność, że pieniądze te są autentyczne i nie stanowią podróbek?
2. Czy mogę mieć pewność, że nikt inny nie będzie rościć sobie praw własności do nich? (problem dublowania płatności).

Emitenci pieniędzy papierowych stale borykają się z problemem fałszowania w związku z coraz to bardziej zaawansowaną technologią produkcji papieru oraz technik drukarskich. Pieniądz fizyczny w prosty sposób rozwiązuje problem podwójnego wydatkowania, ponieważ nie jest możliwe, aby pojawił się w dwóch miejscach jednocześnie. Oczywiście tradycyjne pieniądze są także często przechowywane i przekazywane metodami cyfrowymi. W takich przypadkach, kwestie fałszowania i podwójnego wydatkowania są rozwiązywane w drodze rozliczenia wszystkich elektronicznych transakcji za pośrednictwem jednostek centralnych, które mają globalnygląd waluty będącej w obiegu. W przypadku pieniądza cyfrowego, który nie może korzystać z dobrodziejstwa ezoterycznych atramentów lub pasków holograficznych, kryptografia stanowi podstawę zaufania do prawomocnego roszczenia prawa użytkownika do danej wartości. W szczególności, szyfrowane podpisy cyfrowe umożliwiają użytkownikom złożenie podpisu na towarze lub transakcji stanowiąc dowód prawa własności do danego produktu. Dysponując odpowiednią architekturą, można wykorzystać podpisy cyfrowe w celu rozwiązania problemu podwójnego wydatkowania.

Kiedy w latach 1980-tych szyfrowanie stało się bardziej powszechnie i zrozumiałe, wielu naukowców zaczęło podejmować próby wykorzystania tej techniki w celu tworzenia walut cyfrowych. Pierwsze projekty walut cyfrowych polegały na emisji pieniędzy cyfrowych zwykle zabezpieczonych w walucie krajowej lub mających pokrycie w metalu szlachetnym, np. złocie.

Chociaż te wczesne waluty sprawdzały się w praktyce, działały w ramach silnie zcentralizowanych struktur i w efekcie były łatwym obiektem ataków ze strony rządu oraz hakerów. Wykorzystywały one centralny system rozliczania wszystkich transakcji w regularnych odstępach czasu, podobnie jak ma to miejsce w tradycyjnym systemie bankowym. Niestety, w większości przypadków te nowo powstałe waluty cyfrowe były przedmiotem troski rządów i w ostatecznym rozliczeniu zostały wycofane z obiegu. Niektóre wyeliminowano podczas spektakularnych krachów w związku z pośpieszną likwidacją spółek macierzystych. Aby utrzymać stabilność w obliczu niekorzystnych czynników takich jak uprawnione działania podejmowane przez rządy czy działalność przestępca konieczne było wprowadzenie zdecentralizowanej waluty cyfrowej, aby uniknąć jednego punktu ataku. Bitcoin jest właśnie takim systemem – z założenie całkowicie zdecentralizowanym i wolnym od nadzoru jakichkolwiek jednostek nadzorczych, które mogą być narażone na ataki lub działania korupcyjne.

Bitcoin to punkt kulminacyjny dziesiątek lat badań nad systemami szyfrowania oraz systemami rozproszonymi, który składa się z czterech zasadniczych innowacji stanowiących razem wyjątkową i niezwykle silną kombinację. A zatem Bitcoin składa się ze:

- Zdecentralizowanej sieci typu peer-to-peer (protokół bitcoin)
- Publicznej księgi transakcji (łańcuch blokowy)
- Zdecentralizowanej emisji waluty matematycznej i deterministyczne (rozproszone procesy miningowe)
- Zdecentralizowanego systemu weryfikacji transakcji (skrypty transakcyjne)

[Historia Bitcoin](#)

Pomysł stworzenia bitcoinów zrodził się w roku 2008 wraz z publikacją dokumentu zatytułowanego "Bitcoin: elektroniczny system transakcji gotówkowych typu peer-to-peer," napisanego przez autora występującego pod nazwiskiem Satoshi Nakamoto. Nakamoto dokonał scalenia kilku wcześniejszych rozwiązań takich, jak b-money czy HashCash, aby stworzyć całkowicie zdecentralizowanych elektrycznych system transakcji gotówkowych, który nie jest zależny od żadnego organu centralnego w kwestiach związanych z emisją, rozliczaniem lub walidacją transakcji. Zasadnicza zmiana polegała na wykorzystaniu rozproszonego systemu obliczeniowego (pod nazwą algorytm „proof-of-work”) w celu przeprowadzania globalnych „wyborów” co 10 minut umożliwiających zdecentralizowanej sieci osiąganie *consensusu* w kwestii statusu transakcji. To umożliwia eleganckie rozwiązanie kwestii dublowania wydatków, gdzie jedna jednostka walutowa może być dwukrotnie wydatkowana. Wcześniej. Problem dublowania stanowił poważną słabość walut cyfrowych i był rozwiązywany dzięki rozliczaniu transakcji za pośrednictwem centralnej izby rozrachunkowej.

Sieć bitcoin została uruchomiona w roku 2009 w oparciu o wdrożenie wzorcowe opublikowane przez Nakamoto i od tego czasu uległa licznym modyfikacjom ze strony innych programistów. Rozproszone moce obliczeniowe gwarantujące bezpieczeństwo oraz odporności (stabilność) bitcoinów wzrosło wykładniczo i obecnie przekracza kombinowaną moc przetworzeniową największych światowych super-komputerów. Wartość całego rynku bitcoinów szacuje się między 5 a 10 miliardami dolarów amerykańskich w zależności od kursu wymiany bitcoin/-dolar. Jak dotąd największą transakcją przetworzoną w sieci była transakcja o wartości 150 milionów dolarów amerykańskich – przekaz był natychmiastowy i nieobarczony dodatkowymi opłatami.

Satoshi Nakamoto wycofał się z życia publicznego w kwietniu 2011r. pozostawiając obowiązek opracowania kodu oraz budowy sieci błyskotliwej grupie wolontariuszy. Tożsamość osoby lub osób stojących za opracowaniem bitcoinów jest w dalszym ciągu spowita tajemnicą. Nie mniej jednak ani Satoshi Nakamoto, ani żadna inna osoba nie mają kontroli nad systemem bitcoin, który działa w oparciu o całkowicie przejrzyste zasady matematyczne. Sam pomysł ma przełomowe znaczenie i już przyczynił się do powstania nowych dziedzin nauki zajmujących się rozproszonymi systemami obliczeniowymi, ekonomiką oraz ekonometrią.

Rozwiązanie kwestii rozproszonych systemów obliczeniowych

Wynalazek Satoshi Nakamoto to także praktyczne rozwiązanie dla wcześniej nierozwiązywalnego problemu rozproszonych moc obliczeniowych znanych jako „problem bizantyjskich generałów”. W skrócie rzecz polega na podejmowaniu próby uzgodnienia strategii działania w drodze wymiany informacji za pośrednictwem niewiarygodnej i potencjalnie zagrożonej sieci. Rozwiązanie zaproponowane przez Satoshi Nakamoto wykorzystujące koncepcję proof-of-work (dowodu realizacji zadania) w celu osiągnięcia consensusu bez udziału zaufanej jednostki centralnej stanowi przełom w naukach o rozproszonych systemach obliczeniowych i znalazło szerokie zastosowanie także w innych niż waluta dziedzinach. Może no być wykorzystywane w celu uzgadnienia

zdecentralizowanych sieci oraz udowodnienia prawidłowości przeprowadzonych wyborów, loterii, rejestracji aktywów, cyfrowych poświadczzeń/certyfikacji i wielu innych zastosowaniach.

Wykorzystanie Bitcoin, użytkownicy oraz ich historie

Bitcoin to technologia, ale dotyczy ona pieniądza, który jest fundamentalnym sposobem wyrażania i wymiany wartości pomiędzy ludźmi. Przyjrzymy się osobom, które wykorzystują bitcoin i kilku najczęstszym sposobom wykorzystania waluty oraz protokołu z nią związanego i wysłuchajmy ich historii. Będziemy je wielokrotnie wykorzystywać w dalszych częściach tej książki w celu zilustrowania rzeczywistych sposobów wykorzystania waluty cyfrowej oraz procesów technologicznych stanowiących element składowy bitcoin umożliwiających prowadzenie transakcji.

Niskoobrotowy sklep w Ameryce Północnej

Alice mieszka w północnej części Bay Area w Kalifornii. O bitcoin usłyszała od swoich przyjaciół, fanatyków techniki i postanowiła zacząć je wykorzystywać. Będziemy śledzić jej historię i postępy w zapoznawaniu się ta technologią, pierwszymi bitcoinami oraz kupieniem filiżanki kawy za nie w Bob's Cafe w Palo Alto. Historia ta wprowadzi nas w zagadnienia związane z oprogramowaniem, wymianą oraz zasadniczymi transakcjami z punktu widzenia klienta detalicznego.

Dochodowy punkt sprzedaży w Ameryce Północnej

Carol jest właścicielką galerii sztuki w San Francisco. Prowadzi sprzedaż drogich obrazów i realizuje transakcje w bitcoinach. Ta historia zobrazuje nam ryzyka związane z atakiem w przypadku consensusu "51%" na transakcje detaliczne dotyczące zakupu przedmiotów o dużej wartości.

Kontrakty i usługi zagraniczne

Bob, właściciel kafejki w Palo Alto, tworzy nową stronę internetową. Zował umowę z projektantem stron internetowych z Indii - Gopeshem, który mieszka w Bangalore w Indiach. Gopesh zgodził się na zapłatę wynagrodzenia w bitcoinach. Jego historia pozwoli nam zapoznać się z możliwościami wykorzystania bitcoinów w przypadku outsourcingu, kontraktów na świadczenie usług oraz międzynarodowych przelewów telegraficznych.

Przelewy na rzecz organizacji charytatywnych

Eugenia jest dyrektorem organizacji charytatywnej na rzecz dzieci działającej na Filipinach. W ostatnim czasie odkryła bitcoin i zamierza zacząć wykorzystywać ten system, aby dotrzeć do nowej grupy krajowych jak i zagranicznych darczyńców w celu gromadzenia środków dla swojej organizacji. Eugenia bada także możliwości wykorzystania bitcoinów w celu szybkiego przekazywania funduszy do miejsc, w których są one najbardziej potrzebne. Jej historia połuży nam jako przykład wykorzystania bitcoinów celu zbierania środków na arenie globalnej niezależnie od walut i granic oraz możliwości wykorzystania otwartej księgi w celu zapewnienia przejrzystości wymaganej od organizacji charytatywnych.

Import/eksport

Mohammed zajmuje się importem sprzętu elektronicznego w Dubaju. Próbuje zastosować bitcoin w celu zakupu sprzętu w Stanach Zjednoczonych i Chinach i sprowadzenia go do ZEA w celu przyśpieszenia płatności za zakup towarów zagranicą. Jego historia pokażą nam możliwości wykorzystywania bitcoinów do realizacji dużych płatności międzynarodowych w transakcjach pomiędzy firmami w zakresie towarów fizycznych.

Działalność miningowa na rzecz bitcoin

Jing studiuje na wydziale inżynierii komputerowej w Szanghaju. Zbudował platformę "miningową" pozwalającą wyszukiwać bitcoiny wykorzystując swoje umiejętności inżynierskie w celu uzyskania dodatkowego źródła dochodu. Ta historia pozwoli nam zapoznać się z "przemysłowymi" aspektami bitcoinów: wyspecjalizowany sprzęt wykorzystywany do zabezpieczenia sieci bitcoinów i emisji nowej waluty.

Za każdą z tych opowieści stoją prawdziwi ludzie i prawdziwe działania w oparciu o bitcoiny przyczyniające się do powstawania nowych rynków, nowych sektorów przemysłu oraz innowacyjnych rozwiązań dla globalnych problemów ekonomicznych.

Początek

Aby przyłączyć się do sieci bitcoin i zacząć zawierać transakcje przy wykorzystaniu tej waluty wystarczy pobrać aplikację lub skorzystać z aplikacji internetowej. Ponieważ bitcoin jest standardem, istnieje wiele możliwości wdrożenia oprogramowania klienckiego bitcoin. Jest także dostępna implementacja wzorcowa znana także pod nazwą klient Satoshi, która jest zarządzana jako projekt typu open source przez zespół programistów i stanowi pochodną oryginalnego programu napisane przez Satoshi Nakamoto.

Wyróżnia się trzy główne formy klientów bitcoin:

Pełny klient

Pełny klient lub “pełny węzeł” to klient, który przechowuje całą historię transakcji w bitcoin (każda transakcja dokonana przez każdego użytkownika w dowolnym czasie), zarządzania portfelami użytkowników oraz inicjuje transakcje bezpośrednio w sieci bitcoin. Jest to opcja zbliżona do serwera poczty elektronicznej, ponieważ obsługuje wszystkie aspekty protokołu bez pomocy ze strony innych serwerów lub konieczności wykorzystania usług stron trzecich.

Klient lightweight (lekki)

Klient lightweight przechowuje portfele użytkowników, ale wykorzystuje serwery stron trzecich w celu uzyskania dostępu do transakcji i sieci bitcoin. Klient lekki nie przechowuje pełnej wersji wszystkich transakcji i dlatego musi wykorzystywać serwery stron trzecich w celu walidacji transakcji. Jest to opcja zbliżona do klienckiej wersji poczty elektronicznej typu standalone, która łączy się z serwerem poczty w celu uzyskania dostępu do skrzynki, ponieważ wykorzystuje ona usługi strony trzeciej w celu kontaktu z siecią.

Klient web

Dostęp do klientów web odbywa się za pośrednictwem przeglądarki internetowej, a portfel użytkownika jest przechowywany na serwerze należącym do strony trzeciej. Jest to rozwiązanie zbliżone do internetowej poczty elektronicznej, ponieważ w całości uzależnione jest od serwera strony trzeciej.

Mobilny Bitcoin

Klienci mobilni korzystający ze smartfonów z systemem Android mogą korzystać z systemu jako klienci pełni, lightweight lub webowi. Niektórzy klienci mobilni są zsynchronizowani z siecią lub jako klienci typu desktop client, która stanowi portfel osadzony na wielu platformach w różnych urządzeniach, które mają dostęp do wspólnego źródła zasobów finansowych.

Wybór klienta bitcoin jest uwarunkowany stopniem kontroli jaki chce posiadać użytkownik nad funduszami. Pełny klient będzie mieć najwyższy poziom kontroli oraz niezależności, ale za to obciążony jest koniecznością tworzenia kopii zapasowych oraz zapewnienia systemu zabezpieczeń dla użytkownika. Z drugiej strony możliwości wyboru, klient web jest najłatwiejszy w ustawieniu i wykorzystywaniu, ale ceną za tego typu rozwiązanie jest to, że przejmuje na siebie część ryzyka drugiej strony, ponieważ bezpieczeństwo i kontrola są dzielone pomiędzy użytkownikiem i właścicielem usług internetowych. Jeżeli usługa portfela internetowego zostanie naruszona, jak to już miało miejsce w przeszłości, użytkownicy mogą utracić wszystkie swoje środki. I na odwrót – jeżeli użytkownicy posiadają pełnego klienta bez wykonywania kopii zapasowych mogą utracić wszystkie swoje fundusze wskutek awarii komputera.

Dla celów niniejszej książki, przedstawimy sposób wykorzystania różnych klientów bitcoin możliwych do pobrania – od oprogramowania wzorcowego (klient Satoshi) po portfele sieciowe. Niektóre przykłady będą wymagać wykorzystania klienta wzorcowego, który, poza tym, że jest pełnym klientem wystawia także API dla portfela, sieci oraz serwisów transakcyjnych. Jeżeli planujecie zapoznać się z interfejsami programatycznymi z systemem bitcoin, będzie Wam potrzebny klient wzorcowy.

Szybki start

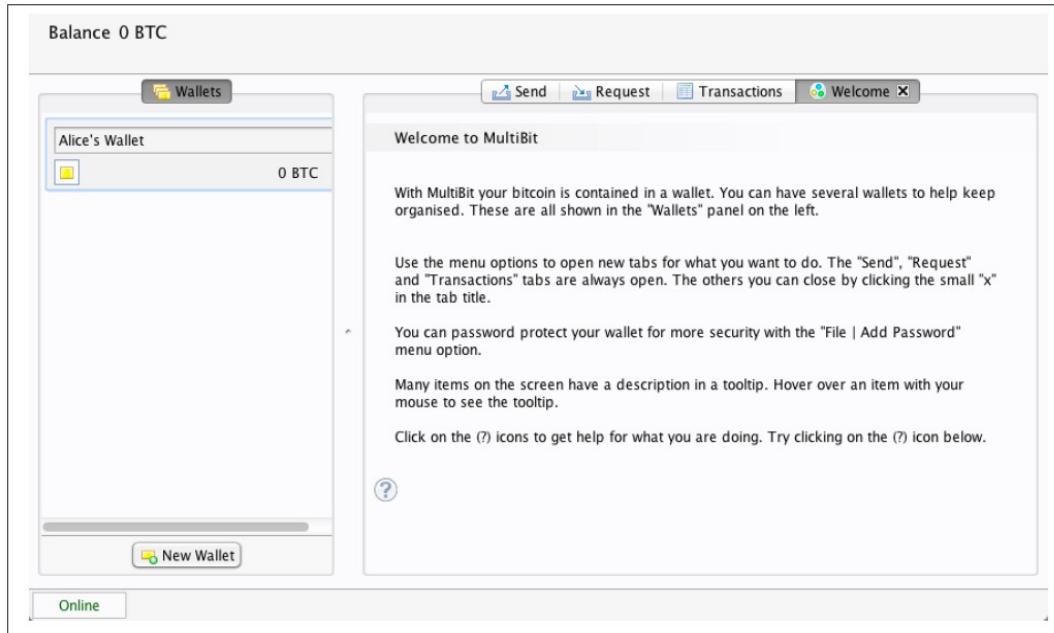
Alice, którą przedstawiliśmy w części "Wykorzystanie bitcoin, użytkownicy oraz ich historie", nie jest użytkownikiem technicznych i dopiero niedawno usłyszała od jednego z przyjaciół. Swoją podróż zaczyna od wizyty na oficjalnej stronie [bitcoin.org](#), gdzie znajduje informacje na temat różnych klientów bitcoin. Idąc za radą znaną na stronie bitcoin.org, wybiera opcję lekkiego klienta bitcoin Multibit.

Alice korzysta z linku na stronie bitcoin.org, aby pobrać i zainstalować aplikację Multibit na swoim komputerze. Multibit jest w wersji kompatybilnej z Windows, Mac OS i Linuxem.



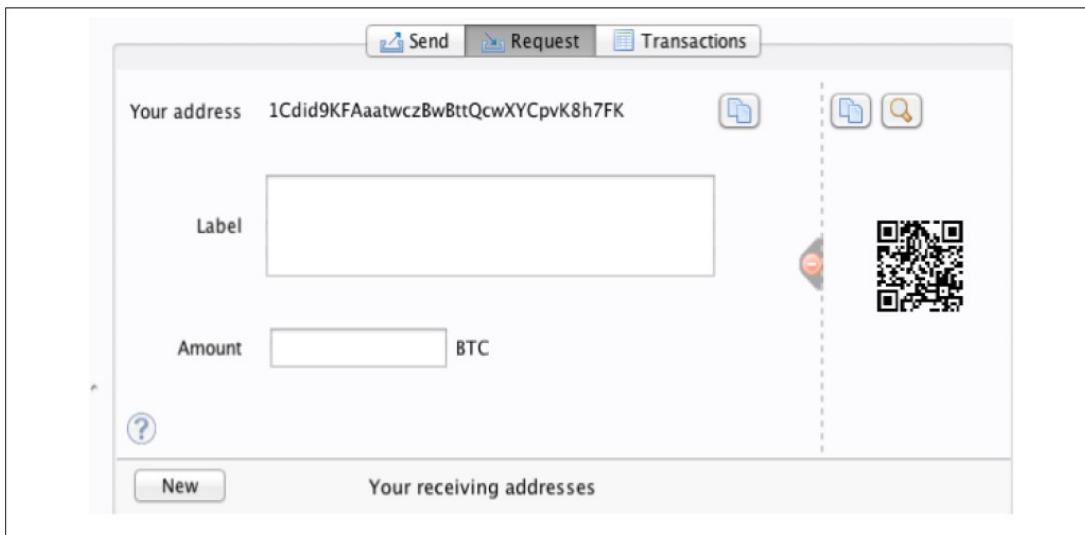
Portfel bitcoin musi być chroniony hasłem lub wyrażeniem o mocy hasła. Istnieje wielu uczestników podejmujących próbę złamania słabych haseł; należy zatem starannie wybrać hasło, które będzie trudno złamać. W tym celu warto stosować kombinacje małych i wielkich liter, liczb oraz symboli. Należy unikać danych osobowych takich jak daty urodzin, nazw zespołów sportowych oraz sów, które można znaleźć w słownikach niezależnie od języka. Jeśli to możliwe używajcie generator haseł, aby stworzyć całkowicie dowolne hasło, które będzie miało długość co najmniej 12 znaków. Pamiętajcie: bitcoin to pieniądz, który może być natychmiast przeniesiony w dowolne miejsce na świecie – bez odpowiednich zabezpieczeń jest łatwą zdobyczą złodziei.

Po pobraniu i zainstalowaniu aplikacji Multibit uruchamia ją i przechodzi do ekranu powitalnego - Rys. 1-1.



Rys. 1-1. Ekran powitalny klienta Multibit

Multibit automatycznie tworzy portfel oraz nowy adres bitcoin dla Alice, który można podejrzeć klikając na zakładkę Request na Rys. 1-2.



Rys. 1-2. Nowy adres bitcoin Alice, w zakładce Request klienta Multibit

Najważniejszą częścią tego ekranu jest *adres bitcoin* otrzymany przez Alice. Podobnie, jak w przypadku poczty elektronicznej, Alice może podawać ten adres i każdy może go wykorzystać w celu przesłania pieniędzy bezpośrednio do jej nowego portfela. Na ekranie pojawia się długie strumienie liter i liczb: 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK. Obok adresu bitcoin portfela podany jest kod QR stanowiący rodzaj kodu kreskowego, który zawiera te same informacje w formacie umożliwiającym zeskanowanie za pomocą smartfonu lub kamery (aparatu). Kod ten podany jest na czarno-białym polu po prawej stronie okna. Alice może przekopiować adres bitcoin lub kod QR do schowka klikając przycisk kopowania znajdujący się obok. Kliknięcie na kod QR spowoduje powiększenie pola, co ułatwia zeskanowanie za pomocą kamery smartfonu.

Alice może także wydrukować kod QR, aby podać swój adres i uniknąć konieczności wpisywania długich ciągów liczb i liter.



Adresy bitcoin zaczynają się od cyfry 1 lub 3. Podobnie jak w przypadku adresów email, można je podawać innym użytkownikom bitcoin w celu zdeponowania środków bitcoin bezpośrednio w Waszym portfelu. Jednak w odróżnieniu do poczty elektronicznej, takich adresów można utworzyć dowolną liczbę i wszystkie będą kierować środki do Waszego portfela. Portfel to nic innego jak zbiór adresów oraz kluczy, które odblokowują środki w nim zdeponowane. Można zwiększyć stopień prywatności wykorzystując nowy adres dla każdej kolejnej transakcji. Praktycznie liczba adresów tworzonych przez użytkownika jest nieograniczona.

Alice może teraz zacząć korzystać ze swojego nowego portfela.

Pierwsze bitcoiny

Bitcoinów nie można kupić w banku lub w kantorze. Od 2014 roku w dalszym ciągu bitcoiny są bardzo trudne do zdobycia w większości krajów. Istnieje pewna liczba kantorów prowadzących kupno i sprzedaż za walutę lokalną. Działają one na rynkach internetowej wymiany walut i są to:

[Bitstamp](#)

Europejski rynek walutowy obsługujący waluty w tym euro (EUR) i dolary amerykańskie (USD) dokonując przekazów elektronicznych.

Coinbase

Jest to amerykański portfel bitcoin i platforma, na której sprzedający i kupujący zawierają transakcję w bitcoinach. Coinbase ułatwia zakup i sprzedaż bitcoinów umożliwiając użytkownikom połączenie z amerykańskimi rachunkami za pośrednictwem systemu ACH.

Kantory wymiany kryptowalut tego typu działają na styku walut krajowych i kryptowalut. Jako takie, podlegają przepisom krajowym i międzynarodowym i często są specyficzne dla danego kraju lub strefy ekonomicznej prowadząc wymianę walut narodowych stosowanych w tej strefie. Wybór waluty wymiany będzie dotyczyć lokalnej waluty i ograniczony do kantorów prowadzących legalną działalność w Waszym kraju. Podobnie jak w przypadku założenia konta bankowego, założenie rachunków niezbędnych w przypadku tego typu usług trwa od kilku dni do kilku tygodni, ponieważ wymagane są różnego rodzaju identyfikatory w celu zapewnienia zgodności z zasadą KYC (know your customer) – poznaj swojego klienta oraz przepisami bankowymi AML (anti-money laundering) w sprawie prania pieniędzy. Po założeniu konta w kantorze bitcoin exchange, można w prosty i szybki sposób nabywać i sprzedawać bitcoin'y podobnie, jak ma to miejsce w przypadku rachunku walutowego na rachunku papierów wartościowych.

Pełna lista dostępna jest na [bitcoin charts](#) – stronie publikującej ceny oraz inne dane rynkowe na różnych rynkach walutowych.

Istnieją jeszcze cztery metody pozyskiwania bitcoinów będąc nowym użytkownikiem:

- Można je nabyć od znajomego, który je posiada. Wielu użytkowników tak właśnie zaczyna.
- Można skorzystać z sieci usług klasyfikowanych (ograniczonych) takich jak localbitcoins.com w celu znalezienia lokalnego sprzedawcy i zakupu bitcoinów za gotówkę w transakcji bezpośredniej.
- Można także dokonać sprzedaży produktu lub usługi za bitcoin'y. Jeżeli jesteś programistą możesz sprzedawać usługi programistyczne.
- Warto korzystać z bankomatów zainstalowanych w Waszej miejscowości. Znajdziecie najbliższy bankomat bitcoin na mapie online [CoinDesk](#).

Alice usłyszała o bitcoin od znajomego, więc ma łatwy dostęp do źródła swoich pierwszych bitcoinów w oczekiwaniu na założenie, weryfikację oraz aktywację rachunku na kalifornijskim rynku walutowym.

Przelewy i odbiór bitcoinów

Alice założyła swój pierwszy portfel bitcoin i może zacząć przyjmować pierwsze środki. Jej aplikacja wygenerowała przypadkowy klucz prywatny (szczegóły w części „[Klucze prywatne](#)“) oraz odpowiadający mu adres bitcoin. W tym momencie, jej adres bitcoin nie jest znany w sieci bitcoin ani „zarejestrowany“ w którymkolwiek miejscu w systemie bitcoin. Jej adres to po prostu liczba, która odpowiada kluczowi, który Alice może wykorzystać do kontroli dostępu do swoich środków. Nie istnieje żadne konto lub związek pomiędzy adresem, a rachunkiem do chwili zarejestrowania tego adresu jako odbiorcy wartości w związku z transakcją zaksięgowaną w księdze rachunkowej bitcoin (łańcuch blokowy) - jest to tylko jeden z wielu adresów, które są „ważne“ w systemie bitcoin. Po skojarzeniu adresu z transakcją staje się on częścią znanego w sieci adresu i od tego momentu Alice może sprawdzić stan konta w księdze publicznej.

Alice spotyka się ze swoim znajomym Joe, który wprowadził ją w system bitcoin, w miejscowej restauracji w celu dokonania wymiany dolarów amerykańskich i zdeponowania

Bitcoinów na jej koncie. Alice przyniosła ze sobą wydruk adresu oraz kod QR, który wyświetlił się w jej portfelu. Z punktu widzenia bezpieczeństwa dane adresowe bitcoin te nie są wrażliwe. Alice może go dowolnie udostępniać bez ryzyka dla bezpieczeństwa konta.

Alice zamierza wymienić drobną kwotę 10 dolarów amerykańskich na bitcoiny, aby nie ryzykować wielkich sum w związku z wykorzystywaniem tej nowej technologii. Wręcza Joe banknot 10-dolarowy i wydruk swojego adresu, aby umożliwić Joe przelanie odpowiednich środków w bitcoin.

Następnie Joe musi zorientować się w kursie wymiany, aby przeląć odpowiednie środki bitcoin na konto Alice. Istnieją setki aplikacji i stron podających aktualne kursy wymiany na danym rynku. Poniżej przedstawiamy kilka najbardziej popularnych:

Bitcoin Charts

Serwis notowań zawierający informacje o cenach rynkowych bitcoin w różnych kantorach na całym świecie denominowanych w walutach lokalnych

Bitcoin Average

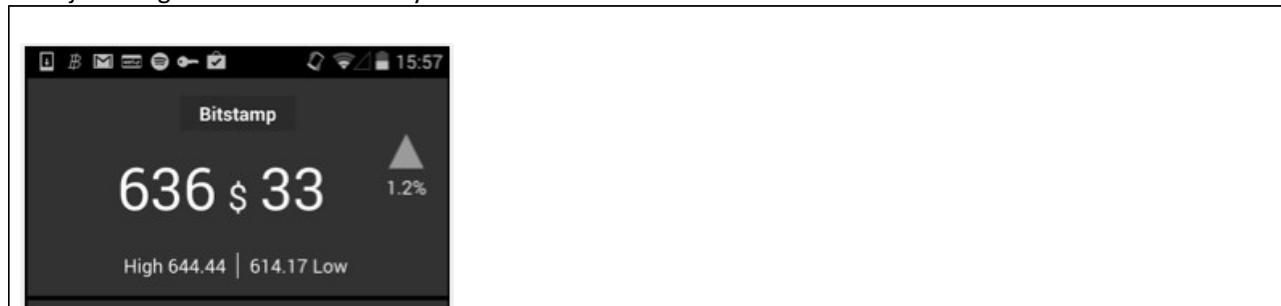
Strona zawierająca przegląd średnich ważonych wolumenem dla każdego kraju

ZeroBlock

Darmowa aplikacja na Androida i iOS podając kursy w różnych kantorach (patrz: Rys. 1-3)

Bitcoin Wisdom

Kolejna usługa notowań walut na rynkach



Rys. 1-3. ZeroBlock, aplikacja zawierająca informacje o cenach rynkowych na Androida i iOS

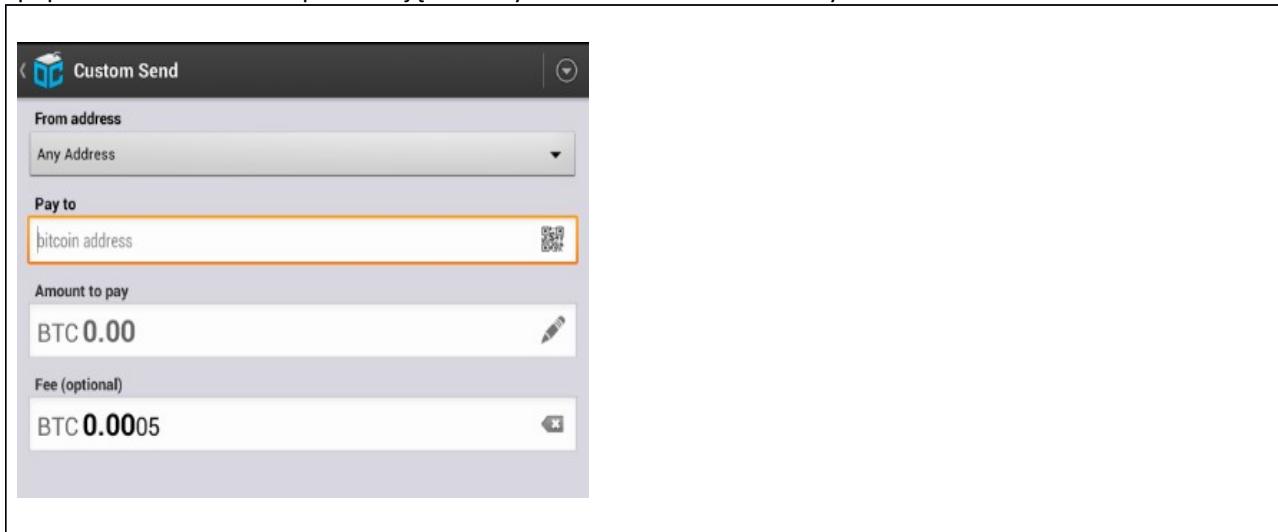
Korzystając z wyżej wymienionych aplikacji lub stron, Joe ustala cenę bitcoinów na poziomie ok. 100 US za jednostkę bitcoin. Kurs ten zapewnia Alice środki w wysokości 0,10 bitcoinów – nazywanych także 100 millibitami w zamian za 10 US, które mu wcześniej przekazała.

Po ustaleniu przez Joe godziwego kursu wymiany otwiera on aplikację w swoim telefonie mobilnym i wybiera opcję “przelej/wyślij” bitcoiny. Na przykład, korzystając z portfela mobilnego Blockchain na telefony z systemem Android na ekranie zobaczy dwa komunikaty - Rys. 1-4 -z prośbą o podanie

- Adresu docelowego transakcji bitcoin
- Kwoty, która ma być przelana

W polu adresu bitcoin znajduje się niewielka ikonka, która przypomina kod QR. Umożliwia ona Joe zeskanowanie kodu kreskowego za pomocą kamery smartfonu co eliminuje konieczność wpisywania adresu bitcoin Alice (1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK), który jest dość długi i trudny do wpisania. Joe dotyka ikonki kodu QR i uruchamia kamerę (aparat) smartfona, aby zeskanować kod z portfela Alice, który przyniosła

ze sobą w postaci wydruku. Aplikacja portfela mobilnego wypełnia adres bitcoin, a Joe może zweryfikować poprawność skanowania porównując kilka cyfr z adresu z adresem na wydruku Alice.



Rys. 1-4. Ekran przelewu mobilnego portfela bitcoin Blockchain

Następnie Joe wprowadza wartość transakcji w bitcoinach, czyli 0,10. Upewnia się, że wprowadzona wartość jest poprawna, ponieważ ma dokonać przelewu środków pieniężnych, a pomyłka może być kosztowna. W końcu zatwierdza przyciskiem Send (wyślij) i transakcja zostaje dokonana. Portfel mobilny bitcoin Joe'go tworzy transakcję, która przypisuje kwotę 0,10 bitcoinów do adresu podanego przez Alice pobierając środki z portfela Joe'go i podpisując transakcję kluczem prywatnym Joe. Jest to informacja dla sieci bitcoin, że Joe autoryzował transfer wartości z jednego ze swoich adresów na nowy adres Alice. Transmisja transakcji za pośrednictwem protokołu peer-to-peer powoduje szybkie rozprzestrzenienie się danych po sieci bitcoin. W czasie krótszym niż jedna sekunda większość węzłów sieci otrzymuje informację o transakcji i dane adresowe kontakt Alice po raz pierwszy.

Gdyby Alice miała ze sobą smartfona albo laptopa mogłaby zobaczyć tę transakcję. Księga bitcoin — stale powiększający się plik, które rejestruje wszystkie transakcje w bitcoin, jakie kiedykolwiek miały miejsce — jest publiczna, co oznacza, że wystarczy, aby Alice znalazła swój adres, aby sprawdzić, czy wpłynęły środki. Może tego łatwo dokonać wchodząc na stronę blockchain.info i wpisując swój adres w polu wyszukiwania. Na stronie pojawi się [lista](#) wszystkich transakcji kierowanych na ten adres i wychodzących z niego. Jeżeli Alice będzie obserwować stronę zobaczy na niej aktualizację salda i wpływ 0w10 bitcoinów wysłanych przez Joe'go.

Potwierdzenia

Początkowo, adres Alice przedstawi wszystkie transakcje przychodzące od Joe jako "Niepotwierdzone". Oznacza to, że transakcja została rozesłana po sieci, ale nie została jeszcze zarejestrowana w księdze transakcyjnej bitcoin – znanej, jako łańcuch blokowy. W celu rejestracji transakcja musi być „znaleziona” przez minera (górnika) i dopisana do bloku transakcji. Po utworzeniu nowego bloku, po upływie ok. 10 minut, transakcje zapisane w bloku będą uznane za "potwierdzone" przez sieć i mogą być przekazane. Transakcje są natychmiast widoczne dla wszystkich, ale jest ona uznawana przez wszystkich za „zaufaną”, jeżeli będzie zapisana w nowo utworzonym bloku.

Teraz Alice jest dumną posiadaczką 0,10 bitcoinów, które może wydać. W kolejnym rozdziale zapoznamy się z pierwszą transakcją dokonaną przez Alice w bitcoinach oraz z transakcją podstawową i technologiami powielania nieco bardziej szczegółowo.

ROZDZIAŁ 2

Jak działa Bitcoin

Transakcje, bloki, mining i łańcuchy blokowe

W odróżnieniu od tradycyjnego systemu bankowego, system bitcoin oparty jest na zasadach zdecentralizowanego zaufania. Zamiast wyznaczania centralnej jednostki jako instytucji zaufania publicznego, w bitcoin zaufanie buduje się w oparciu o interakcje zachodzące pomiędzy różnymi uczestnikami systemu. W niniejszym rozdziale, będziemy zajmować się systemem bitcoin z poziomu ogólnego śledząc poszczególne transakcje zawierane w tym systemie oraz obserwując wzrost poziomu „zaufania” i akceptacji przez mechanizm rozproszonego consensusu bitcoin oraz ostateczną rejestrację transakcji w łańcuchu blokowym – rozproszonej księdze wszystkich transakcji.

Każdy przykład oparty jest na rzeczywistych transakcjach zawartych w sieci bitcoin uruchamiających interakcje pomiędzy jej użytkownikami (Joe, Alice oraz Bob) w związku z przesyłaniem środków z jednego portfela do drugiego. Śledząc transakcje i łańcuchy blokowe w sieci bitcoin, będziemy wykorzystywać stronę *blockchain explorer* w celu wizualizacji każdego wykonywanego kroku. Blockchain explorer to aplikacja internetowa, która działa na zasadach mechanizmu wyszukiwania w tym sensie, że umożliwia wyszukiwanie adresów, transakcji oraz bloków oraz obserwację relacji oraz przepływuów pomiędzy nimi.

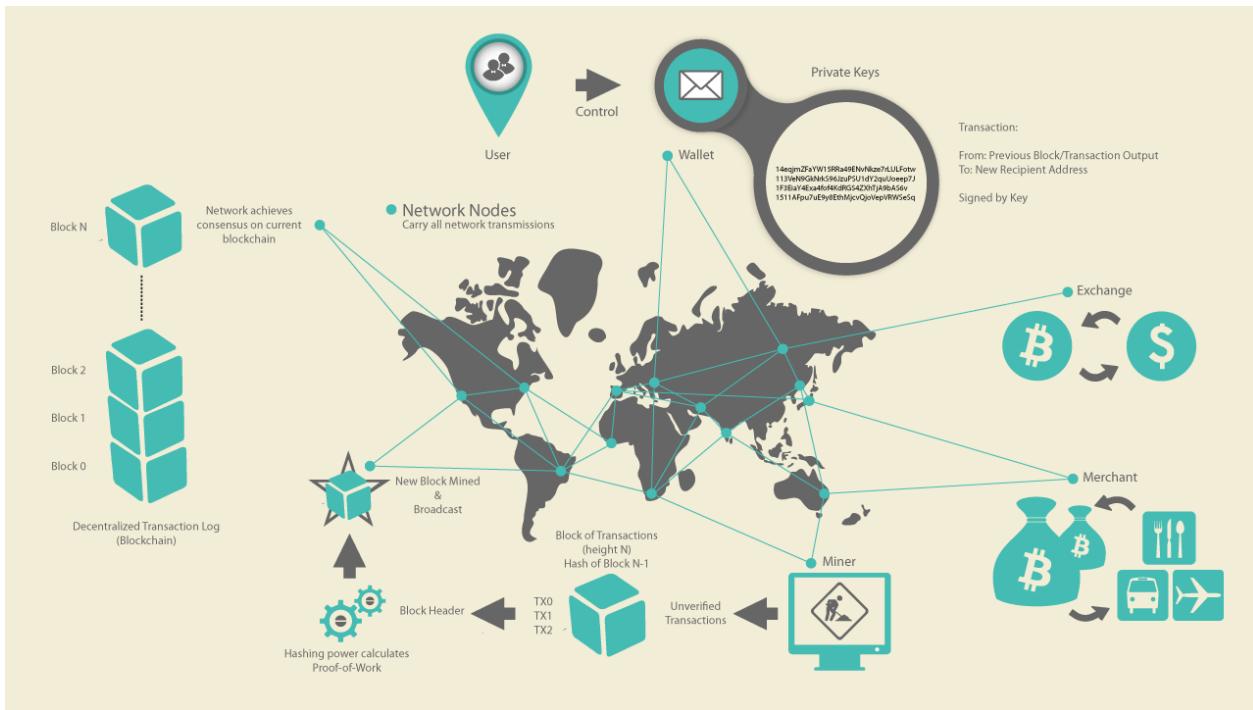
Najczęściej wykorzystywane narzędzia blockchain explorer to:

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

Każda z nich posiada funkcje wyszukiwania w oparciu o adres, hash transakcyjny lub numer bloku w celu wyszukania ekwiwalentnych danych w sieci bitcoin i łańcuchu blokowym. W każdym przykładzie podany URL, który będzie bezpośrednio prowadzić do odpowiedniego zapisu, który będzie można poddać szczegółowej analizie.

Prezentacja ogólna Bitcoin

Na wykresie na [Rys. 2-1](#) widać, że system bitcoin składa się z użytkowników posiadających portfele, w których zapisane są klucze, transakcje przesyłane w sieci oraz górnicy uzyskujący (w drodze konkurencyjnych obliczeń) consensus w ramach łańcuchów blokowych, które stanowią wiarygodną księgę z zapisem wszystkich transakcji. W tym rozdziale będziemy także śledzić pojedyncze transakcje oraz drogę jaką odbywają w sieci, a także badać wzajemne interakcje pomiędzy poszczególnymi elementami systemu bitcoin na ogólnym poziomie. W kolejnych rozdziałach omawiane są technologie wykorzystywane do obsługi portfeli, procesów miningowych oraz systemy kupieckie.



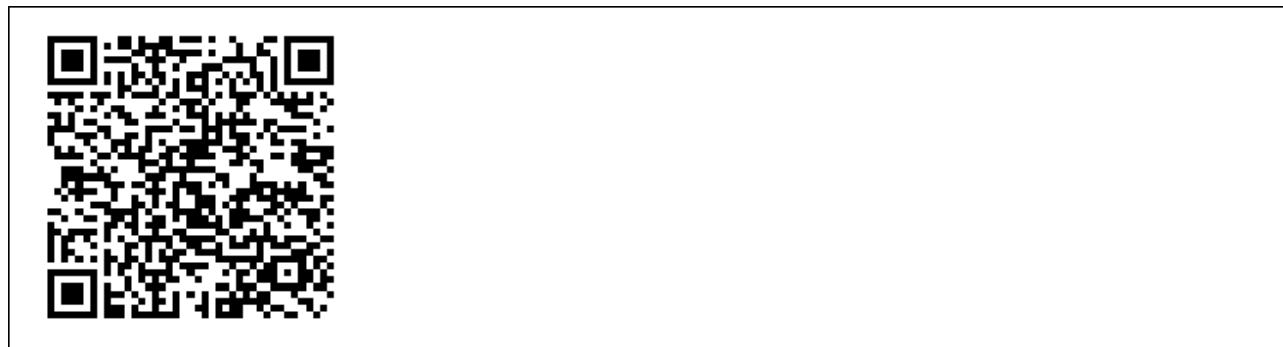
Rys. 2-1. Schemat ogólny sieci Bitcoin

Kupujemy filiżankę kawy

Alice, którą poznaliśmy w poprzednim rozdziale, jest nowym użytkownikiem, który właśnie nabył swoje pierwsze bitcoiny. W rozdziale zat. „[Pierwsze bitcoiny](#)”, Alice spotkała się ze swoim znajomym Joe w celu wymiany gotówki na bitcoiny. Transakcja utworzona przez Joe’go zasiliła portfel Alice kwotą 0,10 BTC. Obecnie Alice dokona swojej pierwszej transakcji kupując kawę w kawiarnie położonej w Palo Alto w Kalifornii, której właścicielem jest Bob. Kawiarnia ta w ostatnim czasie zaczęła przyjmować płatności w bitcoinach umieszczając opcje bitcoin w swoim systemie sprzedażowym. Ceny w Bob’s Cafe są podawane w walucie lokalne (dolary amerykańskie), ale przy kasie klienci mają do wyboru płatności w dolarach lub bitcoinach. Alice zamawia filiżankę kawy i Bob wprowadza transakcję do kaszy. System sprzedażowy przelicza cenę produktu w obu walutach wraz z kodem QR i wezwaniem do realizacji płatności dla tej transakcji (patrz: [Rys. 2-2](#)):

Suma:
\$1,50 USD
0.015 BTC

--



Rys. 2-2. Kod QR wezwania do realizacji płatności (Wskazówka; spróbujcie to zeskanować!)

Kod QR wezwania do realizacji płatności zakodowuje poniższy URL zdefiniowany w BIP0021:

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

Components of the URL

A bitcoin address: "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"

The payment amount: "0.015"

A label for the recipient address: "Bob's Cafe"

A description for the payment: "Purchase at Bob's Cafe"



W odróżnieniu od kodu QR, adres docelowy bitcoin, wezwanie do realizacji płatności to URL z kodem QR, który zawiera adres docelowy, kwotę transakcji oraz ogólny opis np. „Bob's Cafe.” To umożliwia aplikacji portfela bitcoin wstępnie wypełnić informacje umożliwiające dokonywanie płatności i wyświetlanie opisu zrozumiałego dla użytkownika. Kod QR można zeskanować za pomocą aplikacji portfela bitcoin, aby zobaczyć to samo, co Alice.

Bob mówi: „Jeden dolar pięćdziesiąt lub piętnaście milibitów.”

Alice wykorzystuje swojego smartfona, aby zeskanować kod kreskowy na ekranie. Na ekranie urządzenia pojawia się kwota 0,0150 BTC płatna na rzecz Bob's Cafe i Alice wybiera opcję Send, aby autoryzować płatność. W ciągu kilku sekund (mniej więcej tyle ile potrzeba na autoryzacje płatności kartą kredytową), Bob widzi transakcję w rejestrze – to koniec procesu.

W kolejnych rozdziałach będziemy omawiać te transakcję nieco bardziej szczegółowo i zobaczymy w jaki sposób skonstruował ją portfel Alice, jaką drogą przebyła w sieci, sposób jej weryfikacji oraz na koniec w jaki sposób Bob może wykorzystać tę kwotę w kolejnych transakcjach.



Sieć bitcoin może obsługiwać wartości ułamkowe, np. w millibitcoinach (stanowiących 1/1000 bitcoinu) aż do 1/1000000 000nej bitcoinu, które są znane jako satoshi. W tej książce słowo ‐bitcoin‐ odnosi się do ilości jednostek bitcoin

od najmniejszej jednostki (1 satoshi) aż po liczbę ogólną (21 000 000) wszystkich bitcoinów, które kiedykolwiek zostaną wykopane.

Transakcje bitcoinowe

Mówiąc najprościej, transakcja informuje sieć, że właściciel pewnej liczby bitcoinów autoryzował przelew części tych środków do innego właściciela. Nowy właściciel może teraz wydać te bitcoiny tworząc kolejne transakcje autoryzujące przelewy na rzecz kolejnych właścicieli itd. w łańcuchu właścielskim.

Transakcje przypominają linie w księdze z podwójnym zapisem. Mówiąc prościej, każda transakcja zawiera jeden lub więcej "strumienie wejściowych," które są potrącane z konta bitcoin. Z drugiej strony transakcji mamy jedną lub więcej "danych (bloków) wyjściowych", które są dopisywane do rachunku. Wszystkie dane (bloki wejściowe i wyjściowe) niekoniecznie dają tę samą kwotę po zsumowaniu. Zamiast tego, suma strumieni wyjściowych daje wartość nieco niższą niż strumienie wejściowe, a różnica stanowi "opłatę transakcyjną", która jest drobną płatnością na rzecz górnika, który zapisuje transakcję w księdze. Transakcja bitcoin jest przedstawiona jako pozycja w księdze - [Rys. 2-3](#).

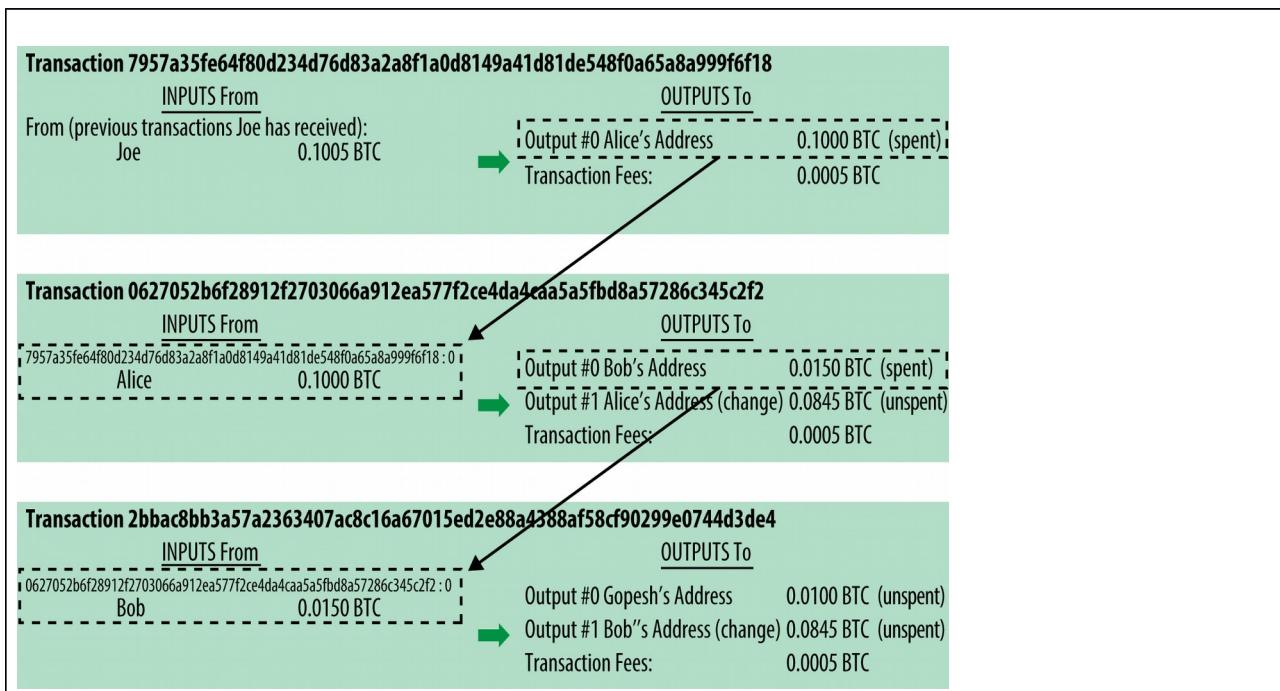
Transakcje zawierają również informacje o *proof of ownership* dla każdej kwoty wyrażonej w bitcoinach (bloki wejściowe), których wartość jest przekazywana, w formie cyfrowego podpisu właściciela, który dowolny użytkownik może zwalidować. Jeżeli chodzi i bitcoiny, "wydatki" to podpis pod transakcją, który autoryzuje przelew wartości z poprzedniej transakcji do nowego właściciela określonego w adresie bitcoinowym.



Transakcje przenoszą wartości z bloków wejściowych do wyjściowych. Strumień wejściowy określa pochodzenie wartości – reguły jest to strumień wyjściowy z poprzedniej transakcji. Blok wyjściowy przypisuje nowego właściciela do wartości kojarząc ją z odpowiednim kluczem. Klucz docelowy nazywany jest – ograniczenie z ang. *encumbrance*. Narzuca on obowiązek złożenia podpisu pod środkami finansowymi, które będą rozliczone w kolejnych transakcjach. Bloki wyjściowe jednej transakcji mogą być wykorzystane jako wejściowe w nowej, tym samym tworząc łańcuch własności w procesie przechodzenia danej wartości na kolejne adresy (patrz: [Rys. 2-4](#)).

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

Rys. 2-3. Transakcje księgowane w systemie podwójnych zapisów



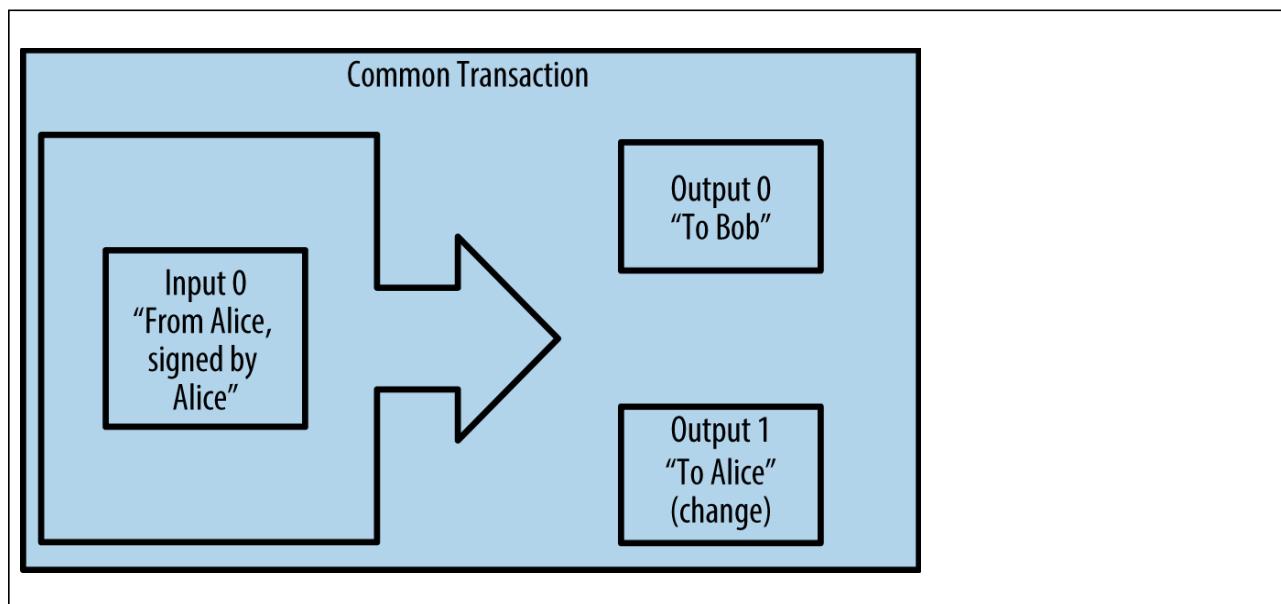
Rys. 2-4. Łącuch transakcji, w którym strumień wyjściowy jednej transakcji staje się blokiem wejściowym w kolejnej

Płatność dokonana przez Alice na rzecz Bob's Cafe wykorzystuje poprzednią transakcję jako strumień wejściowy. W poprzednim rozdziale, Alice otrzymała bitcoiny od swojego znajomego Joe w zamian za gotówkę.

Transakcja ta ma pewną liczbę zablokowanych bitcoinów (encumbered) w kluczu Alice. Nowa transakcja, której odbiorca jest Bob's Cafe zawiera informacje na temat poprzedniej transakcji jako blok wejściowy i tworzy nowe dane wyjściowe pozwalające dokonać płatności za kawę i otrzymanie reszty. Transakcje tworzą łańcuch, w którym dane wejściowe dotyczące ostatniej transakcji odpowiadają danym wyjściowym z poprzednich transakcji. Klucz Alice zawiera podpis umożliwiający odblokowanie danych wyjściowych wcześniejszych transakcji, stanowiąc potwierdzenie dla sieci bitcoin, że rzeczywiście posiada te fundusze. Alice dołącza płatność za kawę do adresu Boba i tym samym "ogranicza" dane wyjściowe wymogiem utworzenia przez Boba podpisu w celu umożliwienia wydatkowania tej kwoty. Proces ten stanowi transfer wartości pomiędzy Alice i Bobem. Ten łańcuch transakcji od Joe do Alice i następnie do Boba jest przedstawiony na Rys. 2-4.

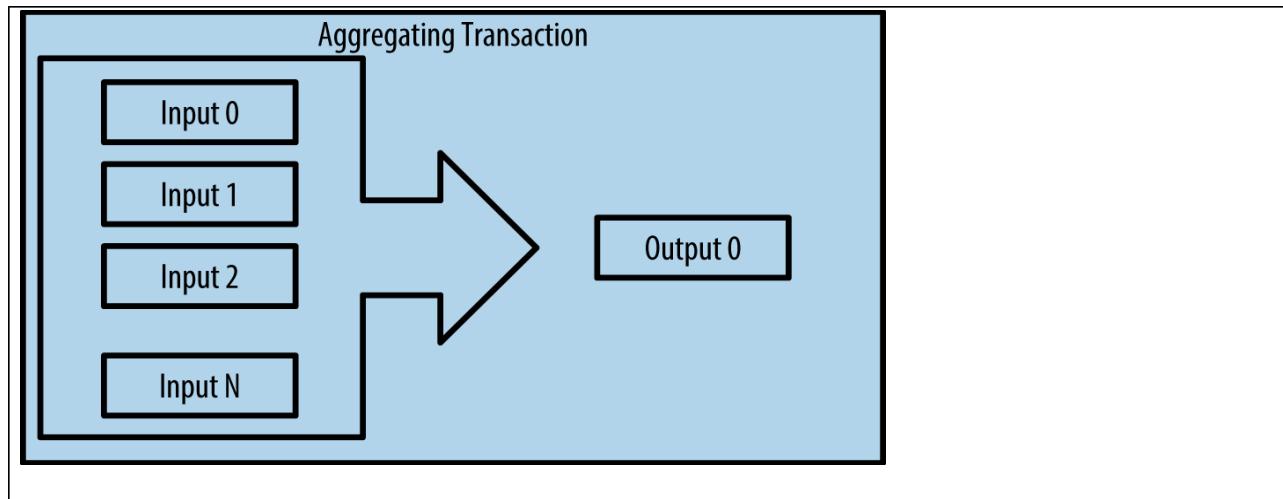
Popularne formy transakcji

Najbardziej popularne formy transakcji to prosta płatność przekazywana z jednego adresu na drugi, która często wymaga zwrotu „reszty” do oryginalnego właściciela. Ten rodzaj transakcji ma jedno uznanie i jeden blok wyjściowy zgodnie z Rys. 2-5.



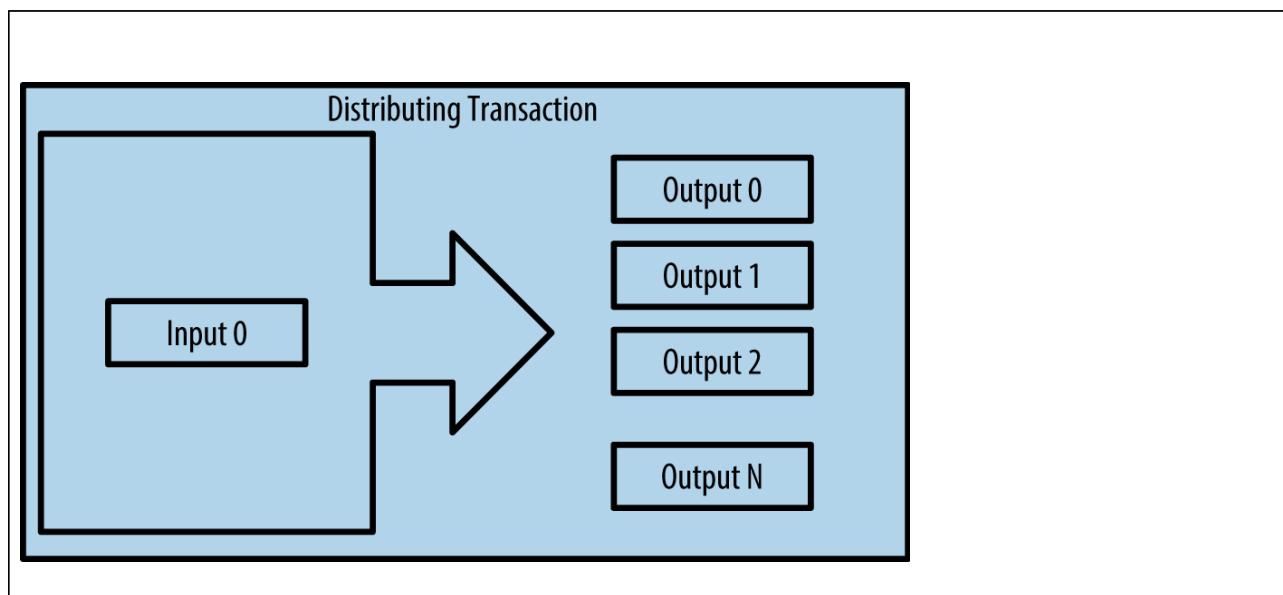
Rys. 2-5. Najczęściej spotykane typy transakcji

Kolejna często spotykana forma transakcji to zagregowanie kilku danych wejściowych w jednym zestawie wyjściowym (patrz: Rys. 2-6). Są to ekwiwalenty wymiany monet oraz banknotów w świecie rzeczywistym na jeden banknot o dużym nominału. Tego typu transakcje są niekiedy generowane przez aplikacje portfelowe w celu usunięcia dużej liczby drobnych kwot otrzymanych jako reszta z transakcji.



Rys. 2-6. Zagregowane środki z transakcji

Na koniec jeszcze jeden rodzaj transakcji, który jest bardzo często odnotowywany w ksiągach bitcoin to transakcje, które kieruje jedne rodzaj danych wejściowych na wiele danych wejściowych reprezentujących wielu odbiorców (patrz: Rys. 2-7). Ten rodzaj transakcji jest niekiedy wykorzystywany przez jednostki komercyjne w celu dystrybuowania środków np. w związku z wypłatą wynagrodzenia dla wielu pracowników.



Rys. 2-7. Transakcja dystrybuowania środków

Konstrukcja transakcji

Aplikacja portfelowa Alice zawiera całą logikę wyboru odpowiednich bloków wejściowych i wyjściowych pozwalających zbudować transakcję zgodnie ze specyfikacjami określonymi przez Alice. Wystarczy, że określi cel oraz kwotę; pozostałe funkcje są wykonywane przez aplikację, a Alice nawet nie widzi ich szczegółów. Ważne jest, że aplikacja portfelowa potrafi konstruować transakcje nawet gdy nie ma dostępu do Internetu (jest w trybie offline). Przypomina to trochę wystawianie czeku w domu i przynoszenie do banku w kopercie –

transakcja nie musi być skonstruowana i podpisana w trakcie połączenia z siecią bitcoin. Wystarczy ją przesłać do sieci, aby została zrealizowana.

Otrzymywanie odpowiednich strumieni wejściowych

Aplikacja portfelowa Alice najpierw musi znaleźć bloki wejściowe, które będą w stanie opłacić kwotę, którą ona chce przekazać Bobowi. Większość aplikacji portfelowych posiada niewielką bazę "danych wyjściowych transakcji niezrealizowanych", które są zablokowane (ograniczone) własnymi kluczami portfela. Dlatego, portfel Alice może zawierać kopię danych wyjściowych z transakcji Joe'go, która została utworzona w zamian za gotówkę (patrz: ["Getting Your First Bitcoins"](#)). Aplikacja portfelowa bitcoin, która działa jako klient z pełnym indeksem w rzeczywistości zawiera kopię każdej niewydatkowanej kwoty z każdej transakcji stanowiącej część łańcucha blokowego. To rozwiązanie umożliwia portfelowi utworzenie danych wejściowych transakcji oraz szybko zweryfikować transakcje przychodzące jako poprawne. Nie mniej jednak ponieważ klient z pełnym indeksem zajmuje dużo miejsca na dysku, większość portfeli użytkowników działa jako klienci "lightweight", którzy śledzą jedynie niewydatkowane strumienie wyjściowe (unspent outputs).

Jeżeli aplikacja portfelowa nie przechowuje kopii niewydatkowanych kwot transakcji, może przesłać do sieci bitcoin zapytanie o odzyskanie tych informacji za pomocą różnych, dostępnych API wysłanych przez różnych użytkowników lub zwrócić się do w pełni zindeksowanego węzła wykorzystującego JSON RPC API. [Przykład 2-1](#) przedstawia prośbę RESTful API skonstruowaną jako polecenie HTTP GET dla danego URL. URL następnie przekazuje informacje o wszystkich niewydatkowanych kwotach w danym adresie przekazując dowolnej aplikacji informacji, które są jej niezbędne do zbudowania obciążen. Wykorzystujemy prostą linię komend HTTP client *cURL*, aby uzyskać odpowiedź.

Przykład 2-1. Wyszukiwanie wszystkich niewydatkowanych kwot dla adresu bitcoin należącego do Alice

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

Przykład 2-2. Odpowiedź na polecenie wyszukania

```
{
  "unspent_outputs": [
    {
      "tx_hash": "186f9f998a5...2836dd734d2804fe65fa35779",
      "tx_index": 10481020,
      "tx_output_n": 0,
      "script": "76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
      "value": 10000000,
      "value_hex": "00989680",
      "confirmations": 0
    }
  ]
}
```

Odpowiedź w [Przykładzie 2-2](#) pokazuje jeden niewydatkowany blok wyjściowy (ten, który nie został jeszcze umorzony) jako należący do adresu Alice 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK. Odpowiedź obejmuje odniesienie do transakcji, w której ten niewydatkowany output jest zapisany (płatność dokona przez Joe'go) oraz jego wartość w satoshi, przy 10 milionach, co stanowi odpowiednik 0,10 bitcoinów. Dzięki tej informacji aplikacja portfelowa Alice może skonstruować transakcję, która przeniesie tę wartość na adres nowego właściciela.



Zapoznajcie się z przebiegiem transakcji [od Joe do Alice](#).

Jak widać portfel Alice zawiera wystarczającą ilość bitcoinów w jednym niewydatkowanym bloku wyjściowym, aby zapłacić za filiżankę kawy. Gdyby tak nie było, aplikacja portfelowa Alice musiałaby “przedrzeć się” przez stos drobniejszych niewydatkowanych strumieni wyjściowych, niczym wybieranie monet z portmonetki, aż znajdzie wystarczającą kwotę, aby zapłacić za kawę. W obu przypadkach, może okazać się konieczny zwrot reszty – z opcją ta zapoznamy się w kolejnym, rozdziale, ponieważ aplikacja portfelowa tworzy dane wyjściowe transakcji (płatności).

Tworzenie bloków wyjściowych

Dane wyjściowe transakcji są tworzone w postaci skryptu, który ogranicza wartość i może być umorzony poprzez wprowadzanie rozwiązań dla skryptu. Mówiąc prościej, dane wyjściowe transakcji Alice będą zwierać skrypt zawierający mniej więcej takie informacje: “Te dane są należne każdemu, kto przedstawi podpis odpowiadający adresowi publicznemu Boba.” Ponieważ tylko Bob posiada portfel z kluczami odpowiadającymi temu adresowi, tylko jego portfel może przedstawić odpis umarzający ten blok wyjściowy. Dlatego Alice będzie “ograniczać” wartość wyjściową żądaniem podpisu Boba.

Transakcja ta będzie także zawierać drugi blok wyjściowy, ponieważ środki Alice są w formie danych wyjściowych 0,10 BTC - zbyt dużo w stosunku do ceny filiżanki kawy, która wynosi 0,015 BTC. Alice należy się reszta w kwocie 0,085 BTC. Reszta dla Alice jest tworzona przez portfel Alice w tej samej transakcji co płatność dla Boba. Zasadniczo portfel Alice dzieli jej środki na dwie płatności – jedna dla Boba, a druga dla niej. Będzie mogła wykorzystać pozostałą resztę w kolejnej transakcji – wydać ją później.

Na koniec, aby transakcja mogła być przetworzona przez sieć bez straty czasu, aplikacja portfelowa Alice doliczy niewielką opłatę. Jest to element nie wprost tej transakcji; jest on implikowany jako różnica pomiędzy danymi wejściowymi i wyjściowymi. Jeżeli zamiast pobrać kwotę reszty 0,085, Alice utworzy jedynie 0,0845 jako drugi strumień wyjściowy, zostanie reszta w wysokości 0,0005 BTC (pół milibitcoina). Blok wejściowy 0,10 BTC nie zostanie w pełni wydatkowany w dwóch blokach wyjściowych, ponieważ ich suma wynosi mniej niż 0,10. Wynikająca różnica to *opłata transakcyjna*, która jest pobierana przez górnika jako opłata za przypisanie transakcji do bloku i zaksięgowanie w księdze łańcucha blokowego.

Można uzyskać podgląd transakcji wynikowej za pomocą aplikacji internetowej blockchain explorer web - [Rys. 2-8](#).

Transaction		View information about a bitcoin transaction	
0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f1			
1Cd1d9KFAaatwczBwBttQcwXYCpvK8h7FK	(0.1 BTC - Output)	1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA - (Unspent) 0.015 BTC	
		1Cd1d9KFAaatwczBwBttQcwXYCpvK8h7FK - (Unspent) 0.0845 BTC	
		97 Confirmations	0.0995 BTC
Summary		Inputs and Outputs	
Size	258 (bytes)	Total Input	0.1 BTC
Received Time	2013-12-27 23:03:05	Total Output	0.0995 BTC
Included In Blocks	277316 (2013-12-27 23:11:54 +9 minutes)	Fees	0.0005 BTC
		Estimated BTC Transacted	0.015 BTC

Rys. 2-8. Transakcja Alice skierowana do Bob's Cafe



Zapoznajcie się z przebiegiem transakcji od Alice do Bob's Cafe.

Dodawanie transakcji do księgi

Transakcja utworzona przez aplikację portfelową Alice ma długość 258 bajtów i zawiera wszystkie informacje niezbędne do potwierdzenia własności środków oraz przypisania nowych właścicieli. Teraz transakcja musi być przesłana do sieci bitcoin, gdzie stanie się częścią rozproszonej księgi (łańcuch blokowy). W kolejnym rozdziale zobaczymy w jaki sposób transakcja staje się częścią nowego bloku oraz w jaki sposób ten jest wydobywany("mined"). Na koniec zobaczymy w jaki sposób nowy blok po dodaniu do łańcucha blokowego zyskuje zaufanie sieci w miarę dodawania kolejnych bloków.

Wysyłanie transakcji

Ponieważ transakcja zawiera wszystkie informacje niezbędne do jej przetworzenia nie ma znaczenia w jaki sposób lub skąd jest przesyłania do sieci bitcoin. Sieć bitcoin to sieć peer-to-peer, w której każdy klient bitcoin uczestniczy łącząc się z innymi klientami bitcoin. Celem sieci bitcoin jest wysyłanie transakcji oraz bloków do wszystkich uczestników.

Powielanie

Aplikacja portfelowa Alice może wysłać nową transakcję do dowolnych klientów bitcoin, z którymi jest połączona za pośrednictwem Internetu: przewodowo, bezprzewodowo lub za pośrednictwem urządzeń mobilnych. Jej portfel bitcoin nie musi być podłączony do portfela bitcoin należącego do Boba i nie musi korzystać z połączenia internetowego dostępnego w kawiarni, chociaż obie opcje są także możliwe. Wszystkie węzły sieci bitcoin (inni klienci), które otrzymują ważne transakcje wcześniej przez nie niewidziane przesyłają je

natychmiast dalej do kolejnych węzłów, z którymi był podłączony. W związku z tym transakcja zaczyna szybko obiegać całą sieć peer-to-peer docierając do dużej liczby węzłów ciągu zaledwie kilku sekund.

Perspektywa Boba

Jeżeli aplikacja portfelowa bitcoin należąca do Boba jest bezpośrednio podłączona do aplikacji portfelowej Alice, może stanowić pierwszy węzeł, który otrzyma informacje o transakcji. Nie mniej jednak nawet jeśli portfel Alice wysyła informacje o transakcjach za pośrednictwem pozostałych węzłów i tak dotrze do portfela Boba w ciągu zaledwie kilku sekund. Portfel Boba natychmiast zidentyfikuje transakcję Alice jako płatność przychodząą, ponieważ zawiera dane umarzalne za pomocą klucza Boba. Aplikacja portfelowa Boba może także w sposób niezależny dokonywać weryfikacji poprawności transakcji w oparciu o wcześniejsze niewydatkowane dane (środki) oraz zawiera wystarczającą ilość opłat transakcyjnych, aby umieścić je w kolejnym bloku. W tym momencie Bob może uznać przy znikomym ryzyku, że transakcja zostanie wkrótce umieszczona w bloku i potwierdzona.



Bardzo popularnym przekonaniem dotyczącym transakcji bitcoin jest to, że muszą być potwierdzone w ciągu 10 minut oczekiwania na kolejny blok lub w ciągu 60 minut na pełne potwierdzenie sześciu konfirmacji. Choć potwierdzenie gwarantuje akceptację transakcji przez całą sieć, takie opóźnienie nie jest konieczne w przypadku niewielkich kwot, jak np. płatność za kawę. Sprzedający może przyjąć transakcję na niewielką kwotę bez potwierdzania ryzykując nie więcej niż ryzykuje w przypadku płatności kartą kredytową bez potwierdzenia i bez podpisu – co obecnie jest działaniem rutynowym.

Bitcoin Mining

Transakcja jest teraz rozsyłana po sieci bitcoin. Nie staje się częścią wspólnej księgi (łańcucha blokowego) do chwili weryfikacji zapisania jej w procesie nazywanym mining. Patrz: [Rozdział 8](#) w celu zapoznania się ze szczegółowym opisem.

System bitcoin jako system zaufania oparty jest na obliczeniach. Transakcje są łączone w bloki, które wymagają dużej ilości obliczeń, aby potwierdzić wiarygodność niewielkiej kwoty. Proces mining pełni w bitcoin dwie funkcje:

- Tworzy nowe bitcoiny w każdym bloku, podobnie jak bank centralny drukujący nowe pieniądze. Kwota bitcoinów utworzonych dla danego bloku jest stała i z czasem zmniejsza się.
- Mining tworzy relacje zaufania zapewniając, że transakcje są potwierdzane wyłącznie, gdy do bloku, który je zawiera zostanie zastosowana odpowiednia moc obliczeniowa. Im więcej bloków, tym więcej obliczeń, tym wyższy poziom zaufania.

Dobrym sposobem opisania miningu jest porównanie go do ogromnej planszy sudoku, która się resetuje za każdym razem, kiedy ktoś znajdzie rozwiązanie; trudność jest regulowana automatycznie w taki sposób, że znalezienie rozwiązania zajmuje ok. 10 minut. Wyobraźcie sobie ogólną układankę sudoku z kilkoma tysiącami rzędów i kolumn. Jeżeli pokaże Wam wypełnioną planszę, szybko będziecie w stanie zweryfikować jej poprawność. Nie mniej jednak, jeżeli układanka ma tylko kilka pól wypełnionych, a reszta jest pusta, potrzeba dużo czasu, aby zadanie rozwiązać! Stopień trudności sudoku można regulować zmieniając wielkość planszy (więcej lub mniej rzędów i kolumn), w dalszym ciągu jednak proces weryfikacji jest dość łatwy nawet jeśli będzie ona duża. "Układanka" wykorzystywana w bitcoin jest oparta na szyfrowaniu i wykazuje się podobnymi

cechami: trudno ją rozwiązać asymetrycznie, ale za to łatwo zweryfikować, a stopień trudności można regulować.

W części „[Wykorzystanie bitcoinów, użytkownicy oraz ich historie](#)”, przedstawiliśmy Jinga – studenta inżynierii komputerowej z Shanghai. Jing jest uczestnikiem sieci bitcoin, w której pełni rolę górnika. Co mniej więcej 10 minut, Jing wraz z tysiącami innych górników bierze udział w globalnym wyścigu, aby znaleźć rozwiązanie dla transakcji blokowych. Znalezienie takiego rozwiązania, tzw. *proof of work* wymaga kwadrylionów operacji hashowania na sekundę wykonywanych w całej sieci bitcoin. Algorytm *proof of work* wymaga powtarzalnego hashowania nagłówka bloku oraz losowo wybranej liczby z algorytmem SHA256 do chwili znalezienia rozwiązania pasującego do wcześniej określonego wzorca. Ten górnik, który znajdzie rozwiązanie jako pierwszy wygrywa rundę i umieszcza dany blok w łańcuchu.

Jing zaczął pracę, jaką miner w 2010 r. korzystając z komputera stacjonarnego z funkcją szybkiego przesyłu danych, aby znajdować odpowiednie *proof of work* dla nowych bloków. W miarę przyłączania się coraz to nowych górników do sieci bitcoin, zaczęły pojawiać się liczne trudności. Wkrótce Jing oraz inni górnicy musieli wymienić sprzęt na bardziej specjalistyczny – najwyższej jakości dedykowane karty jednostki przetwarzania grafiki (graphical processing units - GPUs) podobne do tych, które są wykorzystywane w komputerach lub konsolach do gier. W momencie opracowywania niniejszej publikacji, stopień trudności jest tak duży, że wydobycie jest opłacalne wyłącznie w oparciu o układy zintegrowane specyficzne dla aplikacji (application-specific integrated circuits - ASIC); zasadniczo, setki algorytmów wydobycia (górnictwa) jest wpisanych w urządzenia i działają one równolegle w jednym układzie scalonym. Jing także przyłączył się do “puli górniczej”, która podobnie jak pula loteryjna umożliwia różnym uczestnikom dzielenie się zadaniami i nagrodami. Obecnie, Jing korzysta z dwóch urządzeń ASIC połączonych przez USB, aby wydobywać bitcoiny 24 h na dobę. Koszty energii elektrycznej opłaca zysków ze sprzedaży bitcoinów, które uzyskuje dzięki wydobyciu generując jednocześnie dochód. Na komputerze ma zainstalowaną kopię bitcoind, referencyjnego klienta bitcoin w charakterze rozwiązania backendowego dla wyspecjalizowanego oprogramowania górnictwa, jakim dysponuje.

Bloki transakcji miningowych

Transakcja przesyłana przez sieć nie jest weryfikowana do chwili zarejestrowania w globalnej rozproszonej księdze - łańcuchu blokowym. Średnio, co 10 minut, górnicy generują nowy blok, który zawiera wszystkie transakcje wprowadzone od powstania poprzedniego bloku. Nowe transakcje stale napływają do sieci z portfeli użytkowników oraz innych aplikacji. Po wychwyceniu przez węzły sieci bitcoin są dodawane do tymczasowej puli transakcji niezweryfikowanych przechowywanych w każdym węźle. Górnicy budując nowy blok dodają niezweryfikowane transakcje z tej puli i podejmują próbę rozwiązania trudnego problemu (tzn. *proof of work*), aby potwierdzić ważność nowego bloku. Proces ten jest szczegółowo opisany we „[Wstępie](#)”.

Transakcje są dodawane do nowego bloku otrzymując priorytety - transakcje z najwyższymi opłatami wchodzą jako pierwsze oraz w oparciu o kilka innych kryteriów. Każdy górnik zaczyna proces wydobycia nowego bloku transakcji, gdy tylko otrzyma poprzedni blok wiedząc, że poprzednią rundę wyścigu przegrał. Natychmiast przystępuje do utworzenia nowego bloku, który uzupełnia nowymi transakcjami oraz odciskami palca na wcześniejszych blokach i zaczyna obliczać *proof of work* dla nowego bloku. Każdy górnik umieszcza szczególne transakcje w swoim bloku, tzn. te które przydzielają nagrody jego własnemu adresowi bitcoin w postaci nowo utworzonych bitcoinów (obecnie 25 BTC na blok). Jeżeli górnik znajdzie rozwiązanie, które sprawi, że jego blok stanie się ważny zdobywa nagrodę, ponieważ jego blok zostaje dodany do globalnego łańcucha blokowego, a transakcja z nagrodą, którą dołączył staje się możliwa do wydania. Oprogramowanie Jinga, będącego członkiem puli górników, jest tak ustawione, że tworzy nowe bloki, które przypisują nagrody

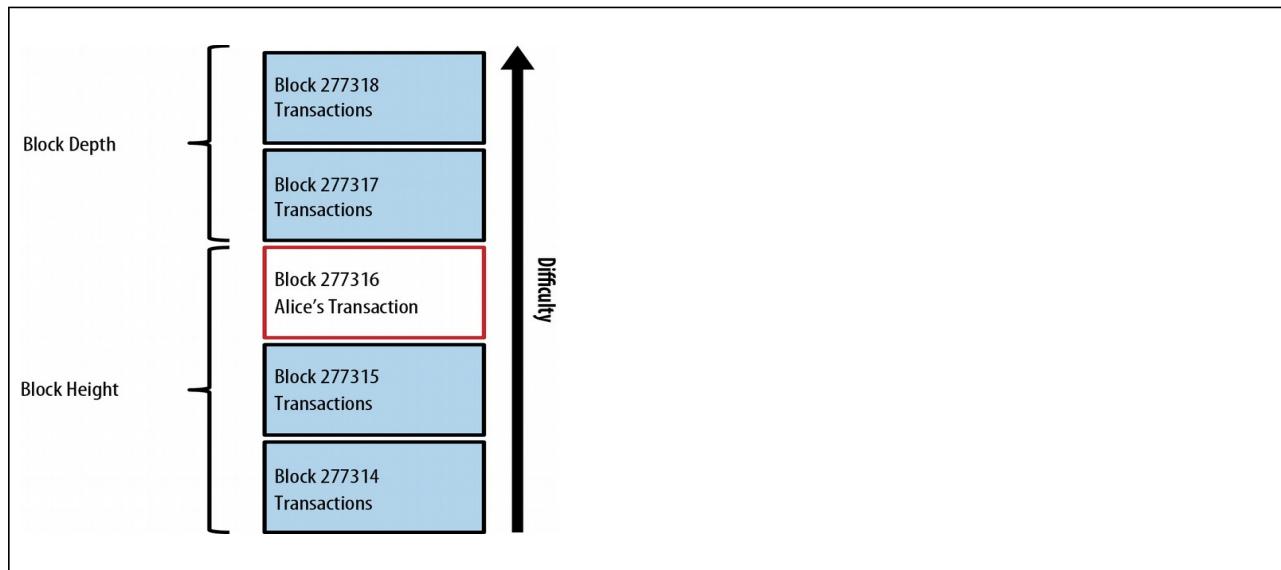
do adresów puli. Z tego miejsca udział w nagrodzie jest przypisywany Jingowi oraz innym górnikom proporcjonalnie do wkładu pracy wniesionego do ostatniej rundy.

Transakcja Alice została wybrana przez sieć i umieszczona w puli transakcji niezweryfikowanych. Ponieważ ma wystarczające opłaty, została dołączona do nowego bloku wygenerowanego przez pulę górniczą Jinga. Po upływie około pięciu minut od wysłania transakcji z portfela Alice, górnik ASIC Jinga znalazł rozwiązanie dla bloku i opublikował je jako blok #277316 zawierający 419 innych transakcji. Górnik ASIC Jinga opublikował nowy blok w sieci bitcoin, gdzie pozostali górnicy zwalidowali i uruchomili nowy wyścig w celu wygenerowania kolejnego bloku.

Możecie zobaczyć blok, który zawiera [transakcję Alice](#).

Kilka minut później nowy blok #277317 wydobyty przez kolejnego górnika. Ponieważ ten nowy blok jest oparty o poprzedni (#277316) - ten z transakcją Alice, dodaje on jeszcze więcej obliczeń do tego bloku jednocześnie wzmacniając zaufanie do tych transakcji. Blok zawierający transakcję Alice jest liczony jako jedno potwierdzenie transakcji. Każdy blok wydobyty na bloku zawierającym transakcję jest dodatkowym potwierdzeniem. W miarę gromadzenia bloków trudność związana z odwróceniem transakcji narasta wykładniczo zwiększając zaufanie ze strony sieci.

Na wykresie na [Rys. 2-9](#) widzimy blok #277316, który zawiera transakcję Alice. Poniżej znajduje się 277 316 bloków (wraz z blokiem #0 połączonym z innymi blokami w łańcuch (łańcuch blokowy) aż do bloku #0, nazywanego *blokiem oryginalnym*). Z biegiem czasu i wzrostu "wysokości" bloków trudności obliczeniowe dla każdego bloku i całego łańcucha rosną. Bloki wydobyte po bloku z transakcją Alice stanowią zabezpieczenia gromadząc się na kolejnych obliczeniach i w coraz dłuższym łańcuchu. Zgodnie z przyjętą konwencją, każdy blok z więcej niż sześcioma potwierdzeniami jest uważany za nieodwoływalny, ponieważ będzie wymagał coraz większego nakładu obliczeniowego w celu unieważnienia i ponownego przeliczenia sześciu bloków. Proces wydobycia oraz budowania zaufania będzie bardziej szczegółowo omówiony w [Rozdziale 8](#).



Rys. 2-9. Transakcja Alice umieszczona w bloku #277316

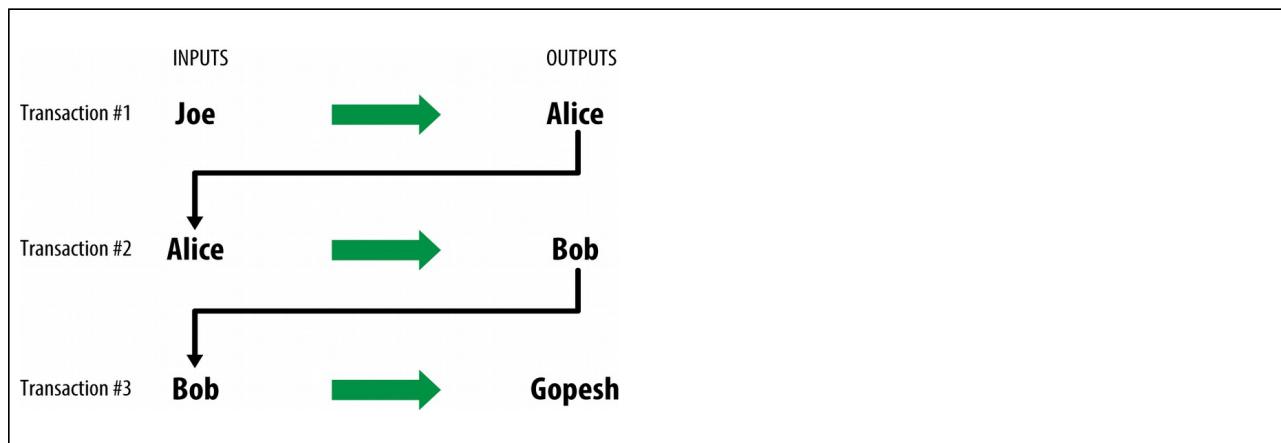
Wydatkowanie transakcji

Pod dodaniu transakcji Alice do łańcucha blokowego jako jego elementu składowego, stała się ona częścią rozproszonej księgi bitcoin i jest widoczna dla wszystkich aplikacji bitcoin. Każdy klient bitcoin może niezależnie zweryfikować transakcje jako ważne i zawierające środki. Klienci fullindex mogą śledzić źródło tych środków od momentu pierwszego wygenerowania bitcoinów w bloku narastająco poprzez poszczególne transakcje aż do chwili dotarcia do adresu Boba. Klienci lightweight mogą wykonać tzw. uproszczoną weryfikację płatności (patrz: "[Węzły uproszczonej weryfikacji płatności \(SPV\)](#)") potwierdzając, że transakcja jest elementem łańcucha blokowego podobnie jak kilka bloków wydobytych po niej tym samym stanowiąc gwarancję uznania przez sieć za ważną.

Bob może teraz wydać dane z tej i innych transakcji tworząc swoje własne transakcje odnoszące się do tych danych jako danych wejściowych i przypisać im nowego właściciela.

Na przykład, Bob może zapłacić dostawcy lub kontrahentowi przesyłając kwotę wpłaconą przez Alice za kawę do nowych właścicieli. Prawdopodobnie, oprogramowanie bitcoin Boba zagreguje kilka drobnych płatności w jedną większą a nawet skoncentruje wszystkie dienne wpłaty w bitcoinach w jedną transakcję. Funkcja ta spowoduje wysłanie różnych płatności na jeden adres wykorzystywany jako konto sklepu. Wykres wszystkich zagregowanych transakcji przedstawia [Rys. 2-6](#).

Bob wykorzystując wpłaty dokonane przez Alice i innych klientów rozbudowuje łańcuch transakcji, które z kolei są dodawane do globalnej księgi łańcucha blokowego, który wszyscy mogą zobaczyć i któremu mogą ufać. Założymy, że Bob płaci projektantowi stron Gopeshowi, który pracuje w Bangalore za opracowanie nowej strony internetowej. Sieć transakcji zmienia wygląd w następujący sposób - [Rys. 2-10](#).



Rys. 2-10. Transakcja Alice jako część łańcucha transakcji od Joe do Gopesha

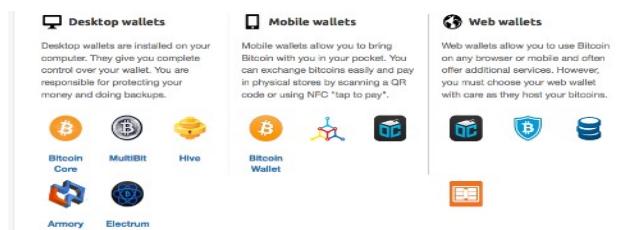
ROZDZIAŁ 3

Bitcoin Client

Rdzeń Bitcoin: oprogramowanie wzorcowe

Wzorcowe oprogramowane *Bitcoin Core*, znane również pod nazwą „Satoshi client” można pobrać ze strony bitcoin.org. Zawiera ono wszystkie aspekty systemu bitcoin, w tym portfele, mechanizm weryfikacji transakcji z pełną kopią księgi transakcyjnej (łańcuch blokowy) oraz kompletny węzeł sieci bitcoin *peer-to-peer*.

Na stronie [Bitcoin wybrać stronę portfela](#), następnie zaznaczyć system rdzeniowy Bitcoin, aby pobrać plik wzorcowego oprogramowania klienta. W zależności od posiadanego systemu operacyjnego pobierany jest odpowiedni instalator wykonywalny. W przypadku Windows jest to albo plik spakowany ZIP lub plik z rozszerzeniem .exe. W przypadku Mac OS jest to obraz dysku .dmg. Wersje na Linuxa zawierają pakiet PPA dla Ubuntu lub plik spakowany tar.gz. Strona bitcoin.org zawiera listę zalecanych klientów bitcoin i przedstawiona jest na [Rys. 3-1](#).

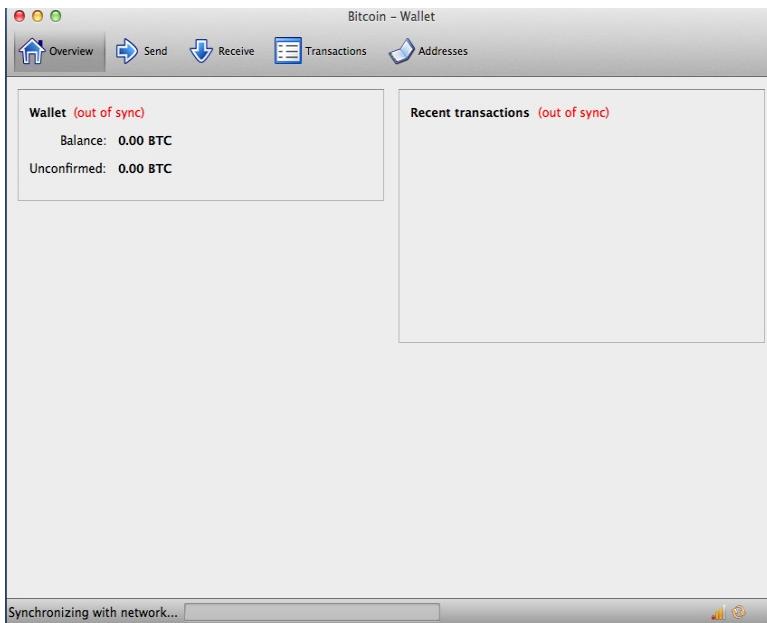


Rys. 3-1. Wybór odpowiedniego klienta bitcoin na stronie bitcoin.org

Pierwsze uruchomienie oprogramowania podstawowego Bitcoin

Po zainstalowaniu pakietu instalacyjnego w postaci plików z rozszerzeniami .exe, .dmg lub PPA, w ten sam sposób można zainstalować dowolną aplikację w systemie operacyjnym. W przypadku Windows należy uruchomić plik .exe i stosować się do instrukcji krok po kroku pojawiających się na ekranie. W przypadku Mac OS, należy uruchomić plik z rozszerzeniem .dmg i przeciągnąć ikonkę Bitcoin-QT do folderu aplikacji (*Applications*). Systemy Ubuntu wymagają dwukrotnego kliknięcia na PPA w pliku przeglądarki (Explorer) co spowoduje otwarcie menadżera pakietu umożliwiającego instalację. Po wykonaniu instalacji, lista Waszych aplikacji zostanie wzbogacona o nowy program Bitcoin-Qt. Podwójne kliknięcie na ikonkę uruchomi aplikację bitcoin client.

Pierwsze uruchomienie Bitcoin Core spowoduje pobranie łańcucha blokowego – proces ten może potrwać kilka dni (patrz: [Rys. 3-2](#)). Nie przerywać działania urządzenia, dopóki na ekranie nie pojawi się komunikat „Zsynchronizowano” i zniknie informacja o braku synchronizacji “out of sync” w oknie obok salda.



Rys. 3-2. Ekran Bitcoin Core podczas inicjalizacji łańcucha blokowego



Bitcoin Core przechowuje pełną kopię księgi transakcyjnej (łańcuch blokowy), zawierającej wszystkie transakcje, jakie pojawiły się w sieci bitcoin od dnia jej powstania w roku 2009. Ten zbiór danych ma wielkość kilkunastu gigabajtów (ok. 16 GB pod koniec 2013r.) i pobierany jest przyrostowo przez kilka dni. Klient nie będzie w stanie przetworzyć transakcji ani zaktualizować sald rachunków do chwili pobrania kompletnego zbioru danych łańcucha blokowego. W tym czasie, na ekranie będzie pojawiać się komunikat o braku synchronizacji "out of sync" w dolnej części ekranu będzie podany status "Synchronizing" (synchronizacja w toku). Należy sprawdzić, czy na dysku jest dostateczna ilość miejsca, szerokość wiązki oraz czas do zakończenia wstępnej synchronizacji.

Kompilacja Bitcoin Core w oparciu o kod źródłowy

Programiści mogą także skorzystać z opcji pobrania pełnego kodu źródłowego w postaci skompresowanej (ZIP) lub poprzez sklonowanie autorytatywnego repozytorium źródłowego z GitHub. Na stronie [GitHub bitcoin](#) zaznaczyć Download ZIP na pasku bocznym. Można także użyć linii poleceń w celu utworzenia lokalnej kopii kodu źródłowego w Waszym systemie. W przykładzie przedstawionym poniżej klonujemy kod źródłowy z linii poleceń Unix dla systemów Linux lub Mac OS:

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



Instrukcje oraz dane wyjściowe mogą różnić się w zależności od wersji. Należy zapoznać się z instrukcjami w dokumentacji dołączonej do kodu nawet jeżeli odbiega ona od instrukcji zwartych w niniejszej książce oraz nie dziwić się, jeśli ekran wynikowy będzie nieco różnić się od przykładów podanych w niniejszym materiale.

Po zakończeniu operacji klonowania przez git zostaje utworzona pełna kopia lokalna repozytorium kodu źródłowego w katalogu *bitcoin*. Należy użyć tego katalogu wpisując cd bitcoin w oknie:

```
$ cd bitcoin
```

Domyślnie, kopia lokalna będzie zsynchronizowana z najnowszym kodem, który może być niestabilny lub stanowić wersję beta programu bitcoin. Przed komplikacją kodu należy określić wybraną wersję sprawdzając odpowiednie informacje w *tagu*. Czynność ta pozwoli zsynchronizować lokalną kopię z migawkowym obrazem repozytorium kodu określonym przez słowo kluczowe. Tagi są wykorzystywane przez programistów w celu określenia wersji kodu w oparciu o numer wersji. Aby znaleźć dostępne tagi należy użyć polecenia git tag:

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

[... many more tags ...]

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

Lista tagów pokazuje wszystkie wersje bitcoin. Zgodnie z konwencją, *release candidates* przeznaczone do testowania mają rozszerzenie "rc". Wersje stabilne, które mogą być instalowane i wykorzystywane w systemach produkcyjnych nie mają żadnego rozszerzenia. Z poprzednio pokazanej listy wybrać najnowszą wersję, która w momencie opracowywania niniejszego materiału ma numer v0.9.0rc1. Aby zsynchronizować wersję oprogramowania z lokalnym kodem należy użyć polecenia git checkout:

```
$ git checkout v0.9.0rc1  
Note: checking out 'v0.9.0rc1'.  
HEAD is now at 15ec451... Merge pull request #3605  
$
```

Ten kod źródłowy zawiera dokumentację, którą można znaleźć w różnych plikach. Należy zapoznać się z treścią dokumentacji głównej w pliku *README.md* w katalogu *bitcoin* wpisując more *README.md* w oknie oraz zatwierdzając polecenie spacją, aby przejść do kolejnej strony. W tym rozdziale zbudujemy wiersz poleceń klienta *bitcoin* znany także jako *bitcoind* dla Linuxa. Należy zapoznać się z instrukcjami dotyczącymi kompilowania wierszy poleceń *bitcoind* klienta na Waszej platformie wpisując more *doc/build-unix.md*. Alternatywne polecenia dla Mac OS X oraz Windows dostępne są w katalogu *doc* jako *build-osx.md* lub *buildmsw.md*.

Należy dokładnie zapoznać się z instrukcjami znajdującymi się w pierwszej części dokumentacji. Są to biblioteki, które muszą obowiązkowo znaleźć się w Waszym systemie zanim przystąpić do kompilacji *bitcoin*.

Jeżeli określone warunki wstępne nie zostaną spełnione, proces budowy zakończy się błędem. W takim przypadku, jeżeli został pominięty warunek wstępny, można go zainstalować, a następnie wznowić proces budowy od ostatniego zakończonego kroku. Zakładając, że warunki zostały zainstalowane, można przystąpić do procesu budowy generując zestaw budujących wykorzystując skrypt *autogen.sh*.



Proces budowy Bitcoin Core został zmieniony i będzie wykorzystywać system autogen/configure/make zaczynając od wersji 0.9. Starsze wersje wykorzystują prosty plik Makefile i działają nieco inaczej niż pokazano na przykładzie poniżej. Należy przestrzegać instrukcji dla wersji, którą chcecie utworzyć. System autogen/configure/make wprowadzony w wersji 0.9 jest bardzo prawdopodobna opcją dla przyszłych wersji kodu i jest to system, który został wykorzystany w celach ilustracyjnych w niniejszym materiale.

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

Skrypt *autogen.sh* tworzy zestaw automatycznych skryptów konfiguracji, które zleca Waszemu systemowi ustalenie poprawnych ustawień oraz zapewnia obecność bibliotek niezbędnych w celu skompilowania kodu. Najważniejszy z nich to skrypt *configure*, umożliwiający różne opcje kustomizacji procesu budowy. Aby zapoznać się z różnymi opcjami należy wpisać polecenie *./configure--help*:

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

Usage: ./configure [OPTION]... [VAR=VALUE]...
To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE. See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

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

[... many more options and variables are displayed below ...]

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]

[... more options ...]

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

Report bugs to <info@bitcoin.org>.
```

\$

Skrypt configure umożliwia uaktywnienie lub wyłączenie niektórych cech bitcoind za pomocą flag --enable-FEATURE oraz --disable-FEATURE, gdzie FEATURE jest zastępowana przez nazwę zgodnie z listą w danych wyjściowych pomocy. W tym rozdziale będziemy budować klienta bitcoind uwzględniając wszystkie cechy domyślne. Nie będziemy wykorzystywać flag konfiguracji, ale warto się z nimi zapoznać, aby zrozumieć, jakie opcjonalne cechy stanowią elementy składowe klienta. Następnie uruchomimy skrypt configure umożliwiający automatyczne wykrycie wszystkich niezbędnych bibliotek oraz utworzenie skustomizowanych skryptów budujących Wasz system:

```
$ ./configure
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
Bitcoin Core: The Reference Implementation | 35
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
```

[... many more system features are tested ...]

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

Jeśli wszystko pójdzie dobrze, wykonywanie polecenia configure zakończy się utworzeniem skustomizowanych skryptów umożliwiających komplikację bitcoind. Jeżeli zabraknie bibliotek lub wystąpią błędy, wykonywanie polecenia configure zakończy się błędem zamiast utworzenia odpowiednich skryptów. W przypadku błędów jest to bardzo prawdopodobne z uwagi na brak lub niewłaściwą bibliotekę. Należy zapoznać się z dokumentacją budowy ponownie oraz upewnić się, że będą zainstalowane wszystkie brakujące warunki. Następnie ponownie uruchomić configure i sprawdzić, czy błąd został usunięty. Teraz można przejść do komplikacji kodu źródłowego – jest to proces, który może zająć nawet kilka godzin. W tym czasie należy sprawdzać dane wyjściowe co kilka sekund lub sprawdzić błędy w przypadku niepowodzenia. Przerwany proces komplikacji można wznowić w dowolnym czasie. Aby zacząć komplikację wystarczy wpisać polecenie make:

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

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

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

Jeśli nie wystąpiły błędy, bitcoind został skompilowany. Ostatnim krokiem jest instalacja pliku wykonywalnego bitcoind w ścieżce systemu za pomocą polecenia make:

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

Teraz można potwierdzić poprawność instalacji bitcoin wydając polecenie wyświetlenia dwóch plików wykonywalnych zgodnie z poniższym:

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

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

Instalacja domyślna bitcoind umieszcza ten element w `/usr/local/bin`. Podczas pierwszego uruchomienia bitcoind, przypomni Wam o konieczności utworzenia pliku konfiguracji z silnym hasłem dla interfejsu JSONRPC. Uruchomić bitcoind wpisując w terminalu polecenie bitcoind:

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

Zacząć edycję pliku konfiguracji korzystając z preferowanego edytora oraz zestawu parametrów zastępując hasło silniejszym zgodnie z zaleceniem bitcoind. *Nie wykorzystywać haseł przedstawionych w niniejszym materiale.* Utworzyć plik wewnątrz katalogu `.bitcoin` pod nazwą `.bitcoin/bitcoin.conf`, a następnie wpisać nazwę użytkownika i hasło:

```
rpcuser=bitcoinrpc  
rpcpassword=2XA4DuKNCbtZXsBQRRNDEwEY2nM6M4H9Tx5dFjoAVVbK
```

Edytując plik konfiguracji można ustawić kilka opcji takich jak np. txindex (patrz: "Indeks baz danych transakcji oraz opcja txindex"). Aby zapoznać się z pełną listą dostępnych opcji należy wpisać polecenie bitcoind --help.

Teraz można uruchomić klienta Bitcoin Core. Podczas pierwszego uruchomienia, program przebuduje łańcuch blokowy bitcoin pobierając wszystkie bloki. Jest to duży, wielogigowy plik i pobranie zajmuje z reguły dwa dni. Czas inicjalizacji łańcucha można skrócić pobierając kopię częściową za pomocą klienta BitTorrent z [SourceForge](#).

Uruchomić bitcoind w tle z opcją -daemon:

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

Korzystanie z JSON-RPC API Bitcoin Core w linii poleceń

Klient Bitcoin Core pobiera interfejs JSON-RPC, który także jest uruchamiany za pomocą linii poleceń pomocnika bitcoin-cli. Linia ta umożliwia prowadzenie interaktywnych eksperymentów z możliwościami, które są także dostępne programatycznie za pośrednictwem

API. Aby uruchomić, należy uruchomić help, aby zobaczyć listę wszystkich dostępnych poleceń RPC:

```
$ bitcoin-cli help  
addmultisigaddress nrequired ["key",...] ( "account" )  
addnode "node" "add|remove|onetry"  
backupwallet "destination"  
createmultisig nrequired ["key",...]  
createrawtransaction [{"txid":"id","vout":n,...} {"address":amount,...}]  
decoderawtransaction "hexstring"  
decodescript "hex"  
dumpprivkey "bitcoinaddress"  
dumpwallet "filename"  
getaccount "bitcoinaddress"  
getaccountaddress "account"  
getaddednodeinfo dns ( "node" )  
getaddressesbyaccount "account"  
getbalance ( "account" minconf )  
getbestblockhash  
getblock "hash" ( verbose )  
getblockchaininfo
```

```

getblockcount
getblockhash index
getblocktemplate ( "jsonrequestobject" )
getconnectioncount
getdifficulty
getgenerate
gethashespersec
getinfo
getmininginfo
getnettotals
getnetworkhashps ( blocks height )
getnetworkinfo
getnewaddress ( "account" )
getpeerinfo
getrawchangeaddress
getrawmempool ( verbose )
getrawtransaction "txid" ( verbose )
getreceivedbyaccount "account" ( minconf )
getreceivedbyaddress "bitcoinaddress" ( minconf )
gettransaction "txid"
gettxout "txid" n ( includemempool )
gettxoutsetinfo
getunconfirmedbalance
getwalletinfo
getwork ( "data" )
help ( "command" )
importprivkey "bitcoinprivkey" ( "label" rescan )
importwallet "filename"
keypoolrefill ( newsize )
listaccounts ( minconf )
listaddressgroupings
listlockunspent
listreceivedbyaccount ( minconf includeempty )
listreceivedbyaddress ( minconf includeempty )
listsinceblock ( "blockhash" target-confirmations )
listtransactions ( "account" count from )
listunspent ( minconf maxconf ["address",...] )
lockunspent unlock [{"txid":"txid","vout":n},...]
move "fromaccount" "toaccount" amount ( minconf "comment" )
Using Bitcoin Core's JSON-RPC API from the Command Line | 39
ping
sendfrom "fromaccount" "tobitcoinaddress" amount ( minconf "comment" "commentto"
)
sendmany "fromaccount" {"address":amount,...} ( minconf "comment" )
sendrawtransaction "hexstring" ( allowhighfees )
sendtoaddress "bitcoinaddress" amount ( "comment" "comment-to" )
setaccount "bitcoinaddress" "account"
setgenerate generate ( genproclimit )
settfee amount
signmessage "bitcoinaddress" "message"
signrawtransaction "hexstring" ( [{"txid":"id","vout":n,"scriptPubKey":"hex","redeemScript":"hex"},...] ["privatekey1",...], sighamstype )
stop
submitblock "hexdata" ( "jsonparametersobject" )
validateaddress "bitcoinaddress"
verifychain ( checklevel numblocks )
verifymessage "bitcoinaddress" "signature" "message"
walletlock
walletpassphrase "passphrase" timeout
walletpassphrasetchange "oldpassphrase" "newpassphrase"

```

Informacje o statusie Bitcoin Core Client

Polecenie: getinfo

Polecenie RPC bitcoin getinfo pozwala wyświetlić podstawowe informacje dotyczące statusu węzła sieci bitcoin, portfela oraz bazy danych łańcucha blokowego. Polecenie bitcoin-cli uruchamia opcję:

```
$ bitcoin-cli getinfo
```

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

Dane są zwracane w notacji obiektowej - JavaScript Object Notation (JSON) – formacie łatwo przyswajalnym dla wszystkich języków programowania, który jest także czytelny dla użytkownika. Dane te zawierają numery wersji oprogramowania klienta bitcoin (90000), protokołu (70002) oraz portfela (60000). Widzimy także saldo bieżące portfela, które wynosi zero. Widzimy także bieżącą wysokość portfela pokazującą, ile bloków jest znanych klientowi (286216). Dostępne są także statystyki sieci bitcoin oraz ustawienia klienta. W dalszej części tego rozdziału omówimy te ustawienia nieco bardziej szczegółowo.



Aktualizacja do bieżącej wysokości łańcucha zapewne trochę potrwa – być może więcej niż jeden dzień, ponieważ system musi pobrać bloki od innych klientów bitcoin. Postęp i liczbę pobranych bloków można sprawdzić korzystając z opcji getinfo.

Ustawienia portfela oraz szyfrowanie

Polecenia: encryptwallet, wallet passphrase

Przed przystąpieniem do tworzenia kluczy oraz innych poleceń należy najpierw zaszyfrować portfel i opatrzyć hasłem. W tym przykładzie używamy polecenia encryptwallet oraz hasła "foo". Oczywiście "foo" należy zastąpić mocnym, złożonym hasłem!

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

Można sprawdzić, czy portfel został zaszyfrowany ponownie uruchamiając getinfo. Teraz na ekranie pojawi się informacja unlocked_until. Jest to licznik pokazujący jak długo hasło odblokowujące będzie przechowane w pamięci udostępniając portfel. Na początku wartość wyniesie zero, co oznacza, że portfel jest zablokowany:

```
$ bitcoin-cli getinfo  
{
```

```

    "version" : 90000,
    [... other information...]
        "unlocked_until" : 0,
        "errors" : ""
    }
$
```

Aby odblokować portfel należy wpisać polecenie `walletpassphrase`, które składa się z dwóch parametrów — hasła oraz czasu (w sekundach) do ponownego, automatycznego zablokowania portfela (licznik czasowy):

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

Odblokowanie portfela można potwierdzić sprawdzając jednocześnie czas ponownie uruchamiając opcję `getinfo`:

```

$ bitcoin-cli getinfo
{
    "version" : 90000,
    [... other information ...]
        "unlocked_until" : 1392580909,
        "errors" : ""
}
```

Aktualizacja portfela, usuwanie oraz odzyskiwanie tekstu Plain-text Dump

Polecenia: `backupwallet`, `importwallet`, `dumpwallet`

W dalszej części będziemy ćwiczyć tworzenie pliku zapasowego portfela oraz odzyskiwanie portfela z pliku zapasowego. Aby utworzyć kopię zapasową należy użyć polecenia `backupwallet`, podając nazwę pliku jako parametr. W tym miejscu można utworzyć kopię zapasową portfela w pliku `wallet.backup`:

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

Teraz można odzyskać plik zapasowy wpisując polecenie `importwallet`. Jeżeli Wasz portfel jest zablokowany, najpierw trzeba będzie odblokować go (patrz: `walletpassphrase` w poprzedniej części) w celu zaimportowania pliku zapasowego:

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

Polecenie `dumpwallet` można wykorzystać w celu przeniesienia portfela do kolejnego pliku tekstowego, który może być odczytany przez użytkownika:

```

$ bitcoin-cli dumpwallet wallet.txt
$ more wallet.txt
# Wallet dump created by Bitcoin v0.9.0rc1-beta (2014-01-31 09:30:15 +0100)
# * Created on 2014-02- 8dT20:34:55Z
# * Best block at time of backup was 286234
(000000000000000f74f0bc9d3c186267bc45c7b91c49a0386538ac24c0d3a44),
# mined on 2014-02- 8dT20:24:01Z

KzTg2wn6Z8s7ai5NA9MVX4vstHRsqP26QJCzLg4JvFrp6mMaGB9 2013-07- 4dT04:30:27Z
change=1 # addr=16pJ6XkwSCqv5ma5FSXMRPaXEYrENCEg47F
Kz3dVz7R6mUpXzdZy4gJEVZxJwA15f198eVui4CUivXotzLBDKY 2013-07- 4dT04:30:27Z
change=1 # addr=17oJds8kaN8LP8kuAkWTco6ZM7BGXFC3gk
[... many more keys ...]

$
```

Adres portfela i odbieranie transakcji

Polecenia: `getnewaddress`, `getreceivedbyaddress`, `listtransactions`, `getaddressesbyaccount`,

getbalance

Klient wzorcowy zachowuje pulę adresów, której wielkość jest dostępna w opcji keypoolsizw przypadku stosowania polecenia getinfo. Adresy te są generowane automatycznie i mogą być wykorzystywane jako publiczne adresy odbiorców lub adresy zmian. Aby uzyskać jeden z tych adresów należy wpisać polecenie getnewaddress:

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

Teraz można wykorzystać adres, aby wysłać drobną kwotę bitcoinów do naszego portfela z portfela zewnętrznego (zakładając, że dysponujemy bitcoinami w kantorze, portfelu sieciowym lub w innym portfelu przechowywanym w innym miejscu). W tym przykładzie, dokonamy przelewu 50 milibitów (0,050 bitcoinów) na poprzedni adres.

Teraz możemy zadać zapytanie klientowi bitcoind o kwotę przekazaną na ten adres i określić, ile potwierdzeń jest niezbędnych zanim kwota zostanie uznana w saldzie. W tym przykładzie ustawimy zero potwierdzeń. Kilka sekund po wysłaniu Bitcoinów z innego portfela, zobaczymy odpowiednia informacje w naszym portfelu. Stosujemy `getreceivedbyaddress` z adresem oraz liczbą potwierdzeń zero (0):

```
$ bitcoin-cli getreceivedbyaddress 1hvzSofGwT8cjB8JU7nBsCSfEVQX5u9CL0  
0.05000000
```

Pomijając zero na końcu polecenia zobaczymy tylko te kwoty, które mają co najmniej minconf potwierdzeń, gdzie minconf oznacza minimalną liczbę potwierdzeń zanim transakcja zostanie dopisana do salda. Ustawienia minconf są zapisywane w pliku konfiguracji bitcoind. Ponieważ transakcja wysyłająca te bitcoiny została wysłana w ciągu ostatnich kilku sekund, nie jest jeszcze potwierdzona i dlatego saldo będzie zerowe:

```
$ bitcoin-cli getreceivedbyaddress 1hvzSofGwT8cjB8JU7nBsCSfEVQX5u9CL  
0.00000000
```

Transakcje otrzymane przez cały portfel mogą być wyświetlane po wpisaniu polecenia `listtransactions`:

```
$ bitcoin-cli listtransactions
[
  {
    "account" : "",
    "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
    "category" : "receive",
    "amount" : 0.05000000,
    "confirmations" : 0,
    "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309ac
bae2c14ae3",
    "time" : 1392660908,
    "timereceived" : 1392660908
  }
]
```

Teraz można wyświetlić listę wszystkich adresów dostępnych w portfelu wpisując polecenie `getaddressesbyaccount:`

```
$ bitcoin-cli getaddressesbyaccount ""  
[  
    "1LQoTPYy1TyerBnV4zZbhEmgyfAipC6eqL",  
    "17vrg8uwMQuibkvS2ECRX4zpvcVI78iFaZS",  
    "1FvrHWhHBBA8cGRRsGIaeqEzUmjKJQWR",  
    "1NVlK3lsI4tBE1KvxrlJwW5XHiunifn02iz".
```

```

        "14MZqqzCxjc99M5ipsQSRfieT7qPZcM7Df",
        "1BhrGvtKFjTAhGdPGbrEwP3xvfjkBuFCa",
        "15nem8CX91XtQE8B1Hdv97jE8X44H3DQMT",
        "1Q3q6taTsUiv3mMemEuQQJ9sGLEGaSjo81",
        "1HoSiTg8sb16oE6SrmazQEwcGEv8obv9ns",
        "13fE8BGhBvnoy68yZKuWJ2hheYKovSDjqM",
        "1hvzSofGwT8cj8JU7nBsCSfEVQX5u9CL",
        "1KHUmVfCJteJ21LmRXHSpPoe23rXKifAb2",
        "1LqJZz1D9yHxG4clkdujnqG5jNNGmPeAMD"
    ]

```

Na koniec, za pomocą polecenia getbalance można zapoznać się z saldem łącznym portfela sumując wszystkie potwierdzone transakcje z co najmniej minconf potwierdzeń:

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



Jeżeli transakcja nie została jeszcze potwierdzona saldo będzie mieć wartość zero po wpisaniu polecenia getbalance. Opcja konfiguracyjna “minconf” określa minimalną liczbę potwierdzeń wymaganych zanim pojawi się informacja o saldzie.

Sprawdzanie i odszyfrowywanie transakcji

Polecenia: gettransaction, getrawtransaction, decoderawtransaction

Obecnie zapoznamy się z transakcjami przychodzącyimi, które zostały uprzednio wyświetcone po wpisaniu polecenia gettransaction. Transakcje można odzyskać według wartości hash danej transakcji pokazanej wcześniej w txid wpisując polecenie gettransaction:

```
$ bitcoin-cli gettransaction
9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3
{
    "amount" : 0.05000000,
    "confirmations" : 0,
    "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
    "time" : 1392660908,
    "timereceived" : 1392660908,
    "details" : [
        {
            "account" : "",
            "address" : "1hvzSofGwT8cj8JU7nBsCSfEVQX5u9CL",
            "category" : "receive",
            "amount" : 0.05000000
        }
    ]
}
```



Identyfikatory transakcji nie są autorytywne do chwili potwierdzenia transakcji. Brak hasha transakcji w łańcuchu blokowym nie oznacza, że transakcja nie została przetworzona. Proces ten nazywany jest “przekształcalnością

transakcji”, ponieważ wartości hash transakcji można zmodyfikować przed potwierdzeniem w bloku. Po potwierdzeniu txid ma charakter ostateczny i nie może być zmieniony.

Transakcja wyświetlana za pomocą polecenia gettransaction ma postać uproszczoną. Aby odzyskać pełny kod transakcji oraz odszyfrować ją stosujemy dwa rodzaje polecień: getrawtransaction oraz decoderawtransaction. Pierwsze polecenie, getrawtransaction wykorzystuje wartość *hash transakcji* (*txid*) jako parametr i zwraca pełną transakcję w postaci surowego strumienia szesnastkowego, w postaci w jakiej jest dostępny w sieci bitcoin:

```
$ bitcoin-cli getrawtransaction 9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc6 ↵
```

```
6c309acbae2c14ae30100000001d717279515f88e2f56ce4e8a31e2ae3e9f00ba1d0add648e80c4 ↵
```

```
80ea22e0c7d3000000008b483045022100a4ebbeec83225dedead659bbde7da3d026c8b8e12e61a ↵
```

```
2df0dd0758e227383b302203301768ef878007e9ef7c304f70ffaf1f2c975b192d34c5b9b2ac1bd ↵
```

```
193dfba2014104793ac8a58ea751f9710e39aad2e296cc14daa44fa59248be58ede65e4c4b884ac ↵
```

```
5b5b6dede05ba84727e34c8fd3ee1d6929d7a44b6e111d41cc79e05dbfe5ceafffffff02404b4c ↵
```

```
00000000001976a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac1f312906000000001 ↵
```

```
976a914107b7086b31518935c8d28703d66d09b3623134388ac00000000
```

Aby odszyfrować ten strumień należy użyć polecenia decoderawtransaction. Skopiować i wkleić strumień szesnastkowy jako pierwszy parametr decoderawtransaction aby zinterpretować pełną treść jako strukturę danych JSON (z przyczyn formatowych strumień szesnastkowy jest skrócony zgodnie z przykładem poniżej):

```
$ bitcoin-cli decoderawtransaction 0100000001d717...388ac00000000
{
    "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "txid" : "d3c7e022ea80c4808e64dd0a1dba009f3eaee2318a4ece562f8ef815",
            "vout" : 0,
```

```

    "scriptSig": {

        "asm": "3045022100a4ebbeec83225dedead659bbde7da3d026c8b8e12e4
61a2df0dd0758e227383b302203301768ef878007e9ef7c304f70ffaf1f2c975b192d34c5b9b2e4

ac1bd193dfba20104793ac8a58ea751f9710e39aad2e296cc14daa44fa59248be58ede65e4c4b2e4
884ac5b5b6dede05ba84727e34c8fd3ee1d6929d7a44b6e111d41cc79e05dbfe5cea",
        "hex": "483045022100a4ebbeec83225dedead659bbde7da3d026c8b8e1e4
2e61a2df0dd0758e227383b302203301768ef878007e9ef7c304f70ffaf1f2c975b192d34c5b9e4

b2ac1bd193dfba2014104793ac8a58ea751f9710e39aad2e296cc14daa44fa59248be58ede65e4
4c4b884ac5b5b6dede05ba84727e34c8fd3ee1d6929d7a44b6e111d41cc79e05dbfe5cea"
    },
    "sequence": 4294967295
},
],
"vout": [
{
    "value": 0.05000000,
    "n": 0,
    "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 07bdb518fa2e6089fd810235cf1100c9c4
13d1fd2 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
            "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL"
        ]
    }
},
{
    "value": 1.03362847,
    "n": 1,
    "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 107b7086b31518935c8d28703d66d09b36e4
231343 OP_EQUALVERIFY OP_CHECKSIG",
    }
}
]
}

```

```

        "hex" : "76a914107b7086b31518935c8d28703d66d09b3623134388ac",
        "reqSigs" : 1,
        "type" : "pubkeyhash",
        "addresses" : [
            "12W9goQ3P7Waw5JH8fRVs1e2rVAKoGnvoy"
        ]
    }
}
]
}

```

Po odkodowaniu transakcji widzimy wszystkie elementy w tym dane wyjściowe i wejściowe (uznania i obciążenia). W tym przypadku widzimy, że transakcja dopisana do naszego nowego adresu w kwocie 50 milibitów wykorzystuje jedno wejście i generuje dwa wyjścia. Dane wejściowe to były dane wyjściowe z wcześniejszej potwierzonej transakcji (przedstawione jako vin txid i zaczynające się od d3c7). Dwa wyjścia odpowiadają uznaniu o wartości 50 milibitów oraz danym wyjściowym z informacją o wydaniu reszty dla nadawcy.

Następnie możemy prześledzić łańcuch blokowy zapoznając się z poprzednią transakcją wymienioną w txid w tej transakcji stosując te same polecenia (np., gettransaction). Przejście od jednej transakcji do drugiej umożliwia śledzenie łańcucha transakcji wstecz w miarę transferu kwoty od jednego właściciela adresu do drugiego.

Po potwierdzeniu otrzymanej transakcji i umieszczeniu jej w bloku, polecenie gettransaction spowoduje zwrot dodatkowych informacji oraz wartości *hash bloku (identyfikator)*, w którym transakcja została zapisana:

```

$ bitcoin-cli gettransaction 9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66<
c309acbae2c14ae3
{
    "amount" : 0.05000000,
    "confirmations" : 1,
    "blockhash" : "00000000000000000051d2e759c63a26e247f185ecb7926ed7a6624bc31c<
2a717b",
    "blockindex" : 18,
    "blocktime" : 1392660808,
    "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
    "time" : 1392660908,
    "timereceived" : 1392660908,
    "details" : [
        {
            "account" : "",
            "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
            "category" : "receive",
            "amount" : 0.05000000
        }
    ]
}

```

W tym miejscu widzimy nowe informacje w polach blockhash (wartość hash bloku, w którym została zapisana transakcja) oraz blockindex o wartości 18 (wskazujący, że nasza transakcja jest 18-tą transakcją umieszczoną w bloku).

Indeks bazy danych transakcji oraz opcji txindex

Domyślnie Bitcoin Core buduje bazę zawierającą *tylko* transakcje związane z portfelem użytkownika. Jeżeli chcecie mieć możliwość dostępu do dowolnej innej transakcji za pomocą polecenia gettransaction, trzeba skonfigurować Bitcoin Core w taki sposób, aby aplikacja zbudowała pełny indeks transakcji, który będzie dostępny wraz z opcją txindex. Ustawić wartości txindex=1 w pliku konfiguracji Bitcoin Core (zwykle znajduje się on w katalogu głównym tutaj: *.bitcoin/bitcoin.conf*). Po zmianie tego parametru należy zrestartować bitcoind i oczekać, aż indeks zostanie odbudowany.

Wyszukiwanie bloków

Polecenie: getblock, getblockhash

Teraz wiemy, który blok naszej transakcji został dodany i możemy zapytać o ten blok. W tym celu wykorzystujemy polecenie getblock z wartością hash bloku jako parametrem:

```
$ bitcoin-cli getblock 000000000000000051d2e759c63a26e247f185ecb7926ed7a6624bd
c31c2a717b true
{
  "hash" : "000000000000000051d2e759c63a26e247f185ecb7926ed7a6624bc31c2a717d
  "b",
    "confirmations" : 2,
    "size" : 248758,
    "height" : 286384,
    "version" : 2,
  "merkleroot" : "9891747e37903016c3b77c7a0ef10acf467c530de52d84735bd555387d
  19f9916",
  "tx" : [
    "46e130ab3c67d31d2b2c7f8fbc1ca71604a72e6bc504c8a35f777286c6d89bf0",
    "2d5625725b66d6c1da88b80b41e8c07dc5179ae2553361c96b14bcf1ce2c3868",
    "923392fc41904894f32d7c127059bed27dbb3cf550d87b9a2dc03824f249c80",
    "f983739510a0f75837a82bfd9c96cd72090b15fa3928efb9cce95f6884203214",
    "190e1b010d5a53161aa0733b953eb29ef1074070658aa656f933ded1a177952",
    "ee791ec8161440262f6e9144d5702f0057cef7e5767bc043879b7c2ff3ff5277",
    "4c45449ff56582664abfadef1907756d9bc90601d32387d9cf4f1ef813b46be",
    "3b031ed886c6d5220b3e3a28e3261727f3b4f0b29de5f93bc2de3e97938a8a53",
    "14b533283751e34a8065952fd1cd2c954e3d37aaa69d4b183ac6483481e5497d",
    "57b28365adaff61aaf60462e917a7cc9931904258127685c18f136eeabd5d35",
    "8c0cc19fff6b66980f90af39bee20294bc745baf32cd83199aa83a1f0cd6ca51",
    "1b408640d54a1409d66ddaf3915a9dc2e8a6227439e8d91d2f74e704ba1cdae2",
    "0568f4fad1fdeff4dc70b106b0f0ec7827642c05fe5d2295b9deba4f5c5f5168",
    "9194bfe5756c7ec04743341a3605da285752685b9c7eebb594c6ed9ec9145f86",
    "765038fc1d444c5d5db9163ba1cc74bba2b4f87dd87985342813bd24021b6faf",
    "bff1caa9c20fa4eeff38877765ee0a7d599fd1962417871ca63a2486476637136",
    "d76aa89083f56fcce4d5bf7fcf20c0406abdac0375a2d3c62007f64aa80bed74",
    "e57a4c70f91c8d9ba0ff0a55987ea578affb92daaa59c76820125f31a9584dfc",
    "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
```

Jak widać blok zawiera 367 transakcji, a 18ta transakcja (9ca8f9...) to txid transakcji uznającej 50 milibitów na naszym adresie. Pole wysokości mówi nam, że jest to 286384-ty blok w łańcuchu.

W tym miejscu możemy także odzyskać blok po wysokości za pomocą polecenia `getblockhash`, które przenosi wysokość bloku jako parametr i zwraca wartość hash dla tego bloku:

```
$ bitcoin-cli getblockhash 0000000000019d6689c085ae165831e934ff763ae46a2a6c174
```

2b3f1b60a8ce26f

W tym miejscu odzyskujemy wartość hash "bloku oryginalnego", pierwszego wydobytego przez Satoshi Nakamoto na wysokość zero. Odzyskanie tego bloku powoduje wyświetlenie:

```
$ bitcoin-cli getblock 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f14
```

b60a8ce26f
{

"hash": "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce264"

```
f",
"confirmations" : 286388,
"size" : 285,
"height" : 0,
"version" : 1,
"merkleroot" : "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7af4
deda33b"
```

```

    "tx" : [
      "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
    ],
    "time" : 1231006505,
    "nonce" : 2083236893,
    "bits" : "1d00ffff",
    "difficulty" : 1.00000000,
    "chainwork" : "0000000000000000000000000000000000000000000000000000000000000001004
010001",
    "nextblockhash" : "00000000839a8e6886ab5951d76f411475428afc90947ee320161b4
bf18eb6048"
  }
}

```

Polecenia getblock, getblockhash oraz gettransaction mogą być wykorzystywane w celu zapoznania się z bazą danych łańcucha programatycznie.

Tworzenie, podpisywanie oraz przekazywanie transakcji w oparciu o niezrealizowane kwoty wyjściowe

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

Transakcje Bitcoin są oparte na koncepcji wydawania „bloków wyjściowych”, które stanowią wynik poprzednich transakcji w celu tworzenia łańcucha transakcji przenoszących własność z adresu na adres. Nasz portfel został zasilony transakcją, która przypisała dany blok wyjściowy do naszego adresu. Po potwierdzeniu może być wydatkowany.

Najpierw wykorzystujemy polecenie listunspent, aby pokazać wszystkie niewydane, ale potwierdzone bloki wyjściowe w naszym portfelu:

```

$ bitcoin-cli listunspent
[
  {
    "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c4
14ae3",
    "vout" : 0,
    "address" : "1hvzSofGwT8cjB8JU7nBsCSfEVQX5u9CL",
    "account" : "",
    "scriptPubKey" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
    "amount" : 0.05000000,
    "confirmations" : 7
  }
]

```

Widzimy, że transakcja 9ca8f9... utworzyła dane wyjściowe (z indeksem 0) przypisane do adresu 1hvzSo... w kwocie 50 milibitów, które w tym punkcie otrzymały siedem potwierdzeń. Transakcje wykorzystują wcześniej

utworzone bloki wejściowe odnosząc się do nich poprzez wcześniejsze txid oraz indeksy vout. Teraz utworzymy transakcję, która wyśle 0-wy vout txid 9ca8f9... jako dane wejściowe i przypisz je do nowego bloku wyjściowego wysyłającego wartość na nowy adres.

Najpierw przyjrzyjmy się danym wyjściowym nieco bliżej. W tym celu wykorzystujemy gettxout aby przejść do szczegółów tego niewydatkowanego uznania. Uznania transakcji są zawsze zapisane w txid i vout w te właśnie parametry podajemy do gettxout:

```
$ bitcoin-cli gettxout 9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309ac<  
bae2c14ae3 0  
{  
  
    "bestblock" : "0000000000000001405ce69bd4ceebcdedb537749cebe89d371eb37e13<  
  
    "899fd9",  
    "confirmations" : 7,  
    "value" : 0.05000000,  
    "scriptPubKey" : {  
        "asm" : "OP_DUP OP_HASH160 07bdb518fa2e6089fd810235cf1100c9c13d1fd2  
OP_EQUALVERIFY OP_CHECKSIG",  
        "hex" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",  
        "reqSigs" : 1,  
        "type" : "pubkeyhash",  
        "addresses" : [  
            "1hvzSofGwT8cjB8JU7nBsCSfEVQX5u9CL"  
        ]  
    },  
    "version" : 1,  
    "coinbase" : false  
}
```

Widzimy tutaj dane wyjściowe, które przypisały 50 milibitów do naszego adresu 1hvz.... Aby wydatkować tę kwotę tworzymy nową transakcję. Najpierw trzeba utworzyć adres, na który dokonamy przelewu:

```
$ bitcoin-cli getnewaddress  
1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JD
```

Zrobimy przelew 25 milibitów na nowy adres 1LnfTn..., który właśnie utworzyliśmy w naszym portfelu. W naszej nowej transakcji, wydamy 50 milibitów z uznania i wyślemy 25 milibitów na ten nowy adres. Ponieważ musimy wydatkować *cały* z wcześniejszej transakcji i musimy także wygenerować resztę. Wygenerujemy resztę, która będzie przekazana z powrotem na adres 1hvz... przekazując resztę na adres, z którego kwota jest oryginalnie wysłana. Na koniec dokonamy opłaty za realizację transakcji. Aby dokonać tej opłaty będziemy musieli pomniejszyć resztę o 0,5 milibitów i przelać 24,5 milibita tytułem reszty. Różnica pomiędzy sumą nowych danych wejściowych (25 mBTC + 24,5 mBTC = 49,5 mBTC) a danymi wejściowymi (50 mBTC) zostanie pobrana jako opłata transakcyjna przez górników.

Aby utworzyć transakcję stosujemy polecenie createrawtransaction. Jako parametry createrawtransaction podajemy dane wejściowe transakcji (50 milibitów) niewydanej kwoty z potwierzonej transakcji) oraz dwa zestawy danych wyjściowych transakcji (pieniądze przesypane na nowy adres oraz reszta przekazana na poprzedni adres):

```

$ bitcoin-cli createrawtransaction '[{"txid": "9ca8f969bd3ef5ec2a8685660fdbf",
  "vout": 0}, {"txid": "7a8bd365524c2e1fc66c309acbae2c14ae3", "vout": 0}]' '{"1LnfTndy3qzXGN19Jwscj1": 0.025,
  "8LR3MVe3JDb": 0.025, "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL": 0.0245}'

0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c000

0000000fffffff02a0252600000000001976a914d90d36e98f62968d2bc9bbd68107564a156a

9bcf88ac50622500000000001976a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac0
```

Polecenie createrawtransaction powoduje utworzenie nieprzetworzonego strumienia szesnastkowego z zakodowanymi danymi przekazanej przez nas transakcji. Potwierdzmy, że wszystko jest poprawne odszyfrując ten nieprzetworzony strumień za pośrednictwem polecenia decoderawtransaction:

```

$ bitcoin-cli decoderawtransaction 0100000001e34ac1e2baac09c366fce1c2245536bd
a8f7db0f6685862aecf53ebd69f9a89c0000000000fffffff02a0252600000000001976a914d
90d36e98f62968d2bc9bbd68107564a156a9bcf88ac50622500000000001976a91407bdb518fa
2e6089fd810235cf1100c9c13d1fd288ac00000000
{
  "txid": "0793299cb26246a8d24e468ec285a9520a1c30fcb5b6125a102e3fc05d4f3cb",
  "a",
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acb",
      "ae2c14ae3",
      "vout": 0,
      "scriptSig": {

```

```

        "asm" : "",
        "hex" : ""
    },
    "sequence" : 4294967295
}
],
"vout" : [
{
    "value" : 0.02500000,
    "n" : 0,
    "scriptPubKey" : {

        "asm" : "OP_DUP OP_HASH160 d90d36e98f62968d2bc9bbd68107564a154"
        "6a9bcf OP_EQUALVERIFY OP_CHECKSIG",
        "hex" : "76a914d90d36e98f62968d2bc9bbd68107564a156a9bcf88ac",
        "reqSigs" : 1,
        "type" : "pubkeyhash",
        "addresses" : [
            "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb"
        ]
    }
},
{
    "value" : 0.02450000,
    "n" : 1,
    "scriptPubKey" : {

        "asm" : "OP_DUP OP_HASH160 07bdb518fa2e6089fd810235cf1100c9c14"
        "3d1fd2 OP_EQUALVERIFY OP_CHECKSIG",
        "hex" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
        "reqSigs" : 1,
        "type" : "pubkeyhash",
        "addresses" : [
            "1hvzSofGwT8cjB8JU7nBsCSfEVQX5u9CL"
        ]
    }
}
]
}

```

Wygląda, że wszystko jest poprawne! Nasz nowa transakcja “konsumuje” niewydatkowane bloki wyjściowe z naszej potwierzonej transakcji a następnie wydaje ją w dwóch transakcjach wyjściowych – jednej w kwocie 25 milibitów na nasz nowy adres oraz drugiej na 24,5 milibita, jako reszta zwracana na oryginalny adres. Różnica w kwocie 0,5 milibita to opłata transakcyjna, która zostanie dopisana do rachunku górnika, który znalazł blok zawierający naszą transakcję.

Jak widać, transakcja zawiera pusty scriptSig, ponieważ nie złożyliśmy jeszcze podpisu. Bez podpisu, transakcja nie ma znaczenia; jeszcze nie udowodniliśmy, że jesteśmy właścicielami adresu, z którego niewydatkowane kwoty pochodzą. Składając podpis usuwamy ograniczenia nałożone na dane wychodzące i udowadniamy, że jesteśmy właścicielami tych danych oraz że możemy nimi dysponować. Podpisujemy transakcję za pomocą polecenia signrawtransaction. Parametrem w tym procesie jest nieprzetworzony strumień szesnastkowy:



Zaszyfrowany portfel musi być odblokowany zanim transakcja zostanie podpisana, ponieważ podpis wymaga dostępu do sekretnego klucza w portfelu.

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

```
$ bitcoin-cli signrawtransaction 010000001e34ac1e2baac09c366fce1c2245536bda8d
```

```
f7db0f6685862aecf53ebd69f9a89c000000000fffffff02a0252600000000001976a914d90d
```

```
d36e98f62968d2bc9bbd68107564a156a9bcf88ac50622500000000001976a91407bdb518fa2e
```

```
6089fd810235cf1100c9c13d1fd288ac00000000
{
```

```
    "hex" : "010000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53e
```

```
bd69f9a89c00000006a47304402203e8a16522da80cef66bacfb0c800c6d52c4a26d1d86a54
```

```
e0a1b76d661f020c9022010397f00149f2a8fb2bc5bca52f2d7a7f87e3897a273ef54b277e4af
```

```
52051a06012103c9700559f690c4a9182faa8bed88ad8a0c563777ac1d3f00fd44ea6c71dc512
```

```
7fffffffff02a0252600000000001976a914d90d36e98f62968d2bc9bbd68107564a156a9bcf88d
```

```
ac50622500000000001976a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac00000000",
    "complete" : true
}
```

Po wykonaniu signrawtransaction zwraca kolejną nieprzetworzoną transakcję szesnastkową. Odszyfrowujemy ją za pomocą polecenia decoderawtransaction:

```
$ bitcoin-cli decoderawtransaction010000001e34ac1e2baac09c366fce1c2245536bda
```

```
8f7db0f6685862aecf53ebd69f9a89c00000006a47304402203e8a16522da80cef66bacfb0c
```

800c6d52c4a26d1d86a54e0a1b76d661f020c9022010397f00149f2a8fb2bc5bca52f2d7a7f87↵

e3897a273ef54b277e4af52051a06012103c9700559f690c4a9182faa8bed88ad8a0c563777ac↵

1d3f00fd44ea6c71dc5127ffffffff02a0252600000000001976a914d90d36e98f62968d2bc9b↵

bd68107564a156a9bcf88ac50622500000000001976a91407bdb518fa2e6089fd810235cf1100↵

```
c9c13d1fd288ac00000000
{
```

"txid" : "ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584ac17b34↵

```
6",
  "version" : 1,
  "locktime" : 0,
  "vin" : [
{

```

"txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acb↵

```
ae2c14ae3",
  "vout" : 0,
  "scriptSig" : {
```

"asm" : "304402203e8a16522da80cef66bacfb0c800c6d52c4a26d1d86↵

a54e0a1b76d661f020c9022010397f00149f2a8fb2bc5bca52f2d7a7f87e3897a273ef54b277e↵

4af52051a0601 03c9700559f690c4a9182faa8bed88ad8a0c563777ac1d3f00fd44ea6c71dc5↵

127",

"hex" : "47304402203e8a16522da80cef66bacfb0c800c6d52c4a26d1d↵

86a54e0a1b76d661f020c9022010397f00149f2a8fb2bc5bca52f2d7a7f87e3897a273ef54b27↵

```

7e4af52051a06012103c9700559f690c4a9182faa8bed88ad8a0c563777ac1d3f00fd44ea6c71<
dc5127"
    },
    "sequence" : 4294967295
}
],
"vout" : [
{
    "value" : 0.02500000,
    "n" : 0,
    "scriptPubKey" : {

        "asm" : "OP_DUP OP_HASH160 d90d36e98f62968d2bc9bbd68107564a15<
6a9bcf OP_EQUALVERIFY OP_CHECKSIG",
            "hex" : "76a914d90d36e98f62968d2bc9bbd68107564a156a9bcf88ac",
            "reqSigs" : 1,
            "type" : "pubkeyhash",
            "addresses" : [
                "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb"
            ]
        }
    },
    {
        "value" : 0.02450000,
        "n" : 1,
        "scriptPubKey" : {

            "asm" : "OP_DUP OP_HASH160 07bdb518fa2e6089fd810235cf1100c9c1<
3d1fd2 OP_EQUALVERIFY OP_CHECKSIG",
                "hex" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
                "reqSigs" : 1,
                "type" : "pubkeyhash",
                "addresses" : [
                    "1hvzSofGwT8cjB8JU7nBsCsFEVQX5u9CL"
                ]
            }
        }
    }
]
}

```

Teraz dane wejściowe wykorzystywane w transakcji zawierają scriptSig, który stanowi podpis cyfrowy potwierdzający własność adresu 1hvz... oraz likwidując ograniczenie na danych wyjściowych i umożliwiający wydatkowanie kwot. Podpis sprawia, że transakcja staje się weryfikowalna dla każdego węzła w sieci bitcoin.

Teraz można przekazać nowo utworzoną transakcję do sieci. Robimy to za pomocą polecenia sendrawtransaction, które wykorzystuje nieprzetworzony strumień szesnastkowy wygenerowany przez signrawtransaction. Jest to ten sam strumień, który odszyfrowaliśmy:

```
$ bitcoin-cli sendrawtransaction010000001e34ac1e2baac09c366fce1c2245536bda8f<
```

```
7db0f6685862aecf53ebd69f9a89c000000006a47304402203e8a16522da80cef66bacfb0c80↵
```

```
0c6d52c4a26d1d86a54e0a1b76d661f020c9022010397f00149f2a8fb2bc5bca52f2d7a7f87e3↵
```

```
897a273ef54b277e4af52051a06012103c9700559f690c4a9182faa8bed88ad8a0c563777ac1d↵
```

```
3f00fd44ea6c71dc5127ffffffff02a0252600000000001976a914d90d36e98f62968d2bc9bbd↵
```

```
68107564a156a9bcf88ac50622500000000001976a91407bdb518fa2e6089fd810235cf1100c9↵
```

```
c13d1fd288ac0000000ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584a↵
```

c17b346

Polecenie sendrawtransaction powoduje podanie wartości *hash transakcji (txid)*, gdyż umieszcza transakcję w sieci. Teraz możemy wyszukać identyfikator transakcji stosując komendę gettransaction:

```
$ bitcoin-cli gettransaction ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584a17b346↵
```

```
21dff584ac17b346
{
    "amount" : 0.00000000,
    "fee" : -0.00050000,
    "confirmations" : 0,
    "txid" : "ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584ac17b346",
    "time" : 1392666702,
    "timereceived" : 1392666702,
    "details" : [
        {
            "account" : "",
            "address" : "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb",
            "category" : "send",
            "amount" : -0.02500000,
            "fee" : -0.00050000
        },
        {
            "account" : "",
            "address" : "1hvzSofGwT8cjB8JU7nBsCSfEVQX5u9CL",
            "category" : "send",
            "amount" : -0.02450000,
            "fee" : -0.00050000
        },
        {
            "account" : "",
            "address" : "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb",
            "category" : "receive",
            "amount" : 0.05000000,
            "fee" : 0.00000000
        }
    ]
}
```

```

        "amount" : 0.02500000
    },
    {
        "account" : "",
        "address" : "1hvzSofGwT8cjB8JU7nBsCSfEVQX5u9CL",
        "category" : "receive",
        "amount" : 0.02450000
    }
]
}

```

Podobnie jak poprzedni, możemy także zbadać ten proces szczegółowo za pomocą polecień getrawtransaction oraz decodetransaction. Polecenia te spowodują wyświetlenie tego samego strumienia szesnastkowego, który opracowaliśmy i odszyfrowaliśmy poprzednio zanim został przekazany do sieci.

Alternatywni klienci, biblioteki oraz zestawy narzędzi

Poza klientem wzorcowym (bitcoind) inni klienci oraz biblioteki mogą być wykorzystywane do współdziałania z siecią bitcoin oraz strukturami danych. Są one wdrażane w różnych językach programowania umożliwiając programistom znane im interfejsy w ich własnych językach.

Dostępne są następujące programy alternatywne:

libbitcoin and sx tools

C++ multithreaded full-node client oraz biblioteka z narzędziami linii poleceń

bitcoinvj

Biblioteka Java full-node client

btcd

Go language full-node bitcoin client

Bits of Proof (BOP)

Program bitcoin Java enterprise-class

picocoin

Oprogramowanie w C lightweight client library dla bitcoin

pybitcointools

Bibliotek Python bitcoin

pycoin

Kolejna biblioteka Python bitcoin

Istnieje znacznie więcej bibliotek w innych językach programowania i równie wiele co chwila jest tworzonych.

Libbitcoin i narzędzia sx

Biblioteka libbitcoin to oprogramowanie w C++, skalowalne, wielordzeniowe, modułowe wspierające klienta typu full-node oraz zestaw narzędzi w linii poleceń nazywanych sx, które oferują wiele tego samego rodzaju możliwości co polecenia bitcoind client, które przedstawiliśmy w tym rozdziale. Narzędzia sx także oferują pewne możliwości w zakresie zarządzania kluczami oraz narzędzia do wprowadzania zmian, które nie są oferowane przez bitcoind z uwzględnieniem kluczy deterministycznych typu 2 oraz skrótów mnemonicznych kluczy.

Installing sx

Aby zainstalować sx oraz wspierające biblioteki libbitcoin, należy pobrać i uruchomić instalator online dla systemu Linux:

```
$ wget http://sx.dyne.org/install-sx.sh  
$ sudo bash ./install-sx.sh
```

Teraz powinniście mieć zainstalowane narzędzia sx. Wpisać sx bez parametrów, aby wyświetlić tekst pomocy, który zawiera listę wszystkich dostępnych poleceń (patrz: [Załącznik D](#)).



Zestaw narzędzi sx zawiera wiele użytecznych poleceń do szyfrowania i deszyfrowania adresów oraz konwersji z różnych formatów oraz reprezentacji. Należy je wykorzystywać w celu badania różnych formatów takich, jak Base58, Base58Check, hex, etc.

pycoin

Biblioteka Python [pycoin](#), pierwotnie napisana i prowadzona przez Richarda Kissa jest to biblioteka oparta na Python, która wspiera manipulacje kluczami bitcoin oraz transakcjami a nawet wspiera język skryptów w wystarczającym stopniu, aby obsługiwać transakcje niestandardowe.

Biblioteka wspiera zarówno Python 2 (2.7.x) jak i Python 3 (po 3.3) i zawiera użyteczne narzędzia wierszy poleceń - ku i tx. Aby zainstalować pycoin 0.42 w Python 3 w środowisku wirtualnym (venv), należy stosować następujące polecenia:

```
$ python3 -m venv /tmp/pycoin  
$ ./tmp/pycoin/bin/activate  
$ pip install pycoin==0.42  
  Downloading/unpacking pycoin==0.42  
    Downloading pycoin-0.42.tar.gz (66kB): 66kB downloaded  
    Running setup.py (path:/tmp/pycoin/build/pycoin/setup.py) egg_info for package  
      pycoin  
  
  Installing collected packages: pycoin  
    Running setup.py install for pycoin  
      Installing tx script to /tmp/pycoin/bin  
      Installing cache_tx script to /tmp/pycoin/bin  
      Installing bu script to /tmp/pycoin/bin  
      Installing fetch_unspent script to /tmp/pycoin/bin  
      Installing block script to /tmp/pycoin/bin  
      Installing spend script to /tmp/pycoin/bin  
      Installing ku script to /tmp/pycoin/bin  
      Installing genwallet script to /tmp/pycoin/bin  
  Successfully installed pycoin  
Cleaning up...  
$
```

Poniżej przedstawiony jest przykład skryptu Python pozwalający na pobieranie i wydawanie bitcoinów w oparciu od bibliotekę pycoin:

```
#!/usr/bin/env python  
from pycoin.key import Key  
from pycoin.key.validate import is_address_valid, is_wif_valid  
from pycoin.services import spendables_for_address  
from pycoin.tx.tx_utils import create_signed_tx  
  
def get_address(which):  
    while 1:  
        print("enter the %s address=> " % which, end="")  
        address = input()
```

```

is_valid = is_address_valid(address)
if is_valid:
    return address
    print("invalid address, please try again")
src_address = get_address("source")
spendables = spendables_for_address(src_address)
print(spendables)
while 1:
    print("enter the WIF for %s=> " % src_address, end="")
    wif = input()
    is_valid = is_wif_valid(wif)
    if is_valid:
        break
    print("invalid wif, please try again")
key = Key.from_text(wif)
if src_address not in (key.address(use_uncompressed=False), key.address(use_uncompressed=True)):
    print(** WIF doesn't correspond to %s" % src_address)
    print("The secret exponent is %d" % key.secret_exponent())
dst_address = get_address("destination")
tx = create_signed_tx(spendables, payables=[dst_address], wifs=[wif])
print("here is the signed output transaction")
print(tx.as_hex())

```

Aby zapoznać się z przykładami narzędzi wierszy poleceń ku and tx należy odwołać się do treści Załącznika B.

btcd

btcd to oprogramowanie dla kompletnych węzłów bitcoin na pisane w Go. Obecnie pobiera, weryfikuje oraz obsługuje łańcuch blokowy w oparciu o precyzyjne(dokładne) reguły (w tym błędy) w celu akceptacji bloku jako oprogramowania wzorcowego bitcoind. Zapewnia także poprawne przekazywanie nowo odkrytych bloków), utrzymuje pulę transakcyjną oraz przekazuje indywidualne transakcje, które jeszcze nie dotarły do bloku. Zapewnia, że wszystkie indywidualne transakcje dopuszczone do puli będą zgodne z większością ścisłych weryfikacji sprawdzających transakcje w oparciu o wymogi górników (transakcje “standardowe”).

Jedna z zasadniczych różnic pomiędzy btcd oraz bitcoind jest taka, że btcd nie obejmuje funkcji portfelowych i decyzja ta była świadomie podjęta. Oznacza to, że nie można wykonać lub odebrać płatności bezpośrednio za pomocą btcd. Funkcja ta znajduje się w projektach btcwallet oraz btcgui, które są obecnie w fazie intensywnego opracowywania. Pozostałe istotne różnice pomiędzy btcd oraz bitcoind obejmują wsparcie btcd dla zgłoszeń HTTP POST

(takich jak bitcoind) oraz preferowanych Websockets oraz fakt, że połączenia RPC btcd mają domyślnie uruchomiony TLS.

Instalacja btcd

Aby zainstalować btcd na Windows, należy pobrać i uruchomić msi dostępne w [GitHub](#) lub uruchomić następujące polecenie w systemie Linux zakładając, że już macie zainstalowany język Go:

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

To update btcd to the latest version, just run:

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

Sterowanie btcd

btcd posiada pewną liczbę opcji konfiguracji, których podgląd można uzyskać wpisując polecenie:

```
$ btcd --help
```

btcd posiada kilka użytecznych kąsków takich jak btcctl, które jest narzędziem wierzy poleceń, które można wykorzystać zarówno do kontroli jak i zapytań kierowanych do btcd za pośrednictwem RPC. btcd nie

uruchamia swojego serwera RPC domyślnie; trzeba skonfigurować co najmniej nazwę użytkownika RPC oraz hasło w następujących plikach konfiguracji:

- *btcd.conf*:

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

- *btcctl.conf*:

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

Jeśli chcecie zastąpić pliki konfiguracji z linii poleceń:

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

Aby wyświetlić listę dostępnych opcji należy uruchomić następującą funkcję:

```
$ btcctl --help
```

ROZDZIAŁ 4

Klucze, adresy, portfele

Wstęp

Własność bitcoin jest ustalana w oparciu o *klucze cyfrowe, adresy bitcoin* oraz *podpisy cyfrowe*. Klucze cyfrowe nie są przechowywane w sieci, ale tworzone i przechowywane przez użytkowników pliku lub po prostu w bazie danych nazywanej *portfelem*. Klucze cyfrowe w portfelu użytkownika stanowią całkowicie niezależny protokół bitcoin, który zasadniczo może być zarządzany przez oprogramowanie portfela użytkownika bez konieczności odnoszenia się do łańcucha blokowego lub dostępu do Internetu. Klucze uaktywniają wiele interesujących funkcji sieci bitcoin włącznie ze zdecentralizowanym zaufaniem oraz kontrolą, poświadczaniem własności oraz modelem bezpieczeństwa w oparciu o szyfry.

Każda transakcja bitcoin wymaga umieszczenia ważnego podpisu w łańcuchu blokowym, który może być wygenerowany w oparciu o ważne klucze cyfrowe; dlatego każdy użytkownik posiadający kopię tych kluczy kontroluje bitcoin na danym rachunku. Klucze tworzone są jako pary składające się z klucza prywatnego (sekretnego) oraz publicznego. Klucz publiczny można porównać z numerem konta bankowego a klucz prywatny do tajnego kodu PIN lub podpisu na czeku, który pozwala kontrolować zarządzać rachunkiem. Te klucze cyfrowe są rzadko widziane przez użytkowników bitcoin. Przez większość czasu są one przechowywane wewnętrz pliku portfela i zarządzane przez oprogramowanie portfela.

W części płatności transakcji bitcoin, odbiorca klucza publicznego jest reprezentowany przez cyfrowy odcisk palca nazywany *adresem bitcoin*, który jest wykorzystywany w ten sam sposób, co nazwisko odbiorcy na czeku (tzn. "Wypłacić na rzecz"). W większości przypadków, adres bitcoin jest generowany na podstawie i odpowiada kluczowi publicznemu. Nie mniej jednak, nie wszystkie adresy bitcoin to klucze publiczne; mogą one także reprezentować innych beneficjentów takich jak skrypty - przekonamy się o tym w dalszej części tego rozdziału. W ten sposób adresy bitcoin abstrahują odbiorcę środków i sprawiają, że adres docelowy transakcji jest elastyczny podobnie jak ten na czekach papierowych: pojedynczy instrument płatniczy, który może być wykorzystywany do wpłat na rachunki osób fizycznych firmowych, w celu dokonania płatności faktur za dostawę mediów lub po prostu płatności gotówkowych. Adres bitcoin to jedyna reprezentacja klucza, który użytkownicy rutynowo widzą, ponieważ jest to ta część, która jest przekazywana pozostałym użytkownikom.

W tym rozdziale będziemy omawiać portfele, które zawierają klucze szyfrowania. Zobaczmy w jaki sposób są generowane, przechowywane oraz zarządzane. Zapoznamy się również z różnymi formatami szyfrowania wykorzystywanymi jako reprezentacje kluczy prywatnych i publicznych, adresów oraz adresów skryptowych. Na koniec zapoznamy się także ze szczególnym zastosowaniem kluczy, tj. do podpisywania wiadomości oraz udowodnienia własności, a także do budowania adresów vanity oraz portfeli papierowych.

Szyfrowanie przy użyciu klucza publicznego oraz kryptowaluty

Szyfrowanie w oparciu o klucz publiczny wymyślono w latach 1970-tych i jest to matematyczna podwalina bezpieczeństwa komputerowego i informatycznego.

Od momentu wynalezienia szyfrowania w oparciu o klucz publiczny, wiele funkcji matematycznych takich jak potęgowanie liczb pierwszych, czy mnożniki krzywej eliptycznej zostały odkryte. Te funkcje matematyczne są praktycznie nieodwracalne co oznacza, że łatwo je obliczyć w jednym kierunku, ale obliczenia w kierunku przeciwnym nie są możliwe. Dzięki tym funkcjom matematycznym, szyfrowanie umożliwia tworzenie

cyfrowych tajnych informacji oraz niezmiennych podpisów cyfrowych. Bitcoin wykorzystuje mnożniki krzywych eliptycznych jako podstawę dla szyfrowania w oparciu o klucz publiczny.

W bitcoin wykorzystujemy szyfrowanie w oparciu o klucz publiczny w celu utworzenia pary kluczy, które kontrolują dostęp do bitcoinów. Para kluczy zawiera klucz prywatny oraz utworzony na jego podstawie unikalny klucz publiczny. Klucz publiczny jest wykorzystywany w celu odbierania bitcoinów; klucz prywatny jest wykorzystywany do podpisywania transakcji i wydatkowania bitcoinów.

Istnieje relacja matematyczna pomiędzy kluczem publicznym a kluczem prywatnym, która umożliwia wykorzystanie klucza prywatnego do wygenerowania podpisów na wiadomości. Podpis ten może być zweryfikowany bez konieczności ujawniania klucza prywatnego.

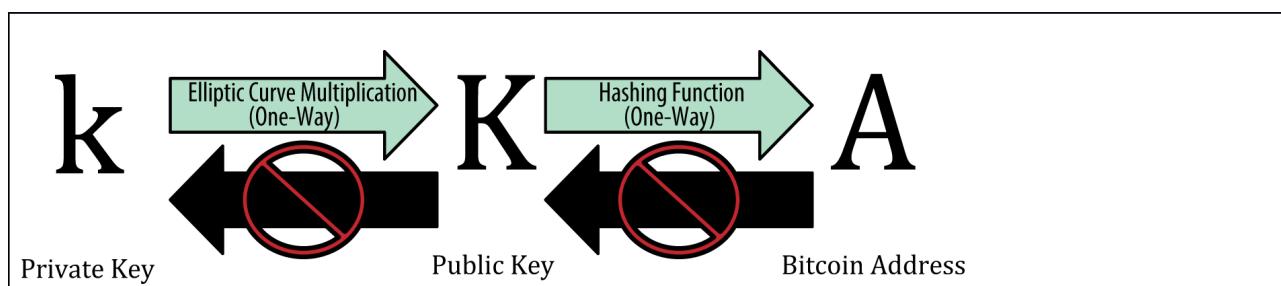
Wydając bitcoiny, obecny właściciel przedstawia swój klucz publiczny oraz podpis (za każdym razem inny, ale tworzony w oparciu o ten sam klucz prywatny) w transakcji związanej z wydatkowaniem tych bitcoinów. Prezentując klucz publiczny oraz podpis, każdy uczestnik sieci bitcoin może zweryfikować i zatwierdzić transakcję jako ważną, potwierdzając jedocześnie, że osoba dokonująca transferu bitcoinów była ich właścicielem w chwili transferu.



W większości oprogramowań portfelowych, klucze prywatne i publiczne są przechowywane razem jako *para kluczy* dla wygody. Nie mniej jednak, klucz publiczny można obliczyć w oparciu o klucz prywatny, zatem przechowywanie jedynie prywatnego klucza jest także możliwe.

Klucze prywatne i publiczne

Portfel bitcoin zawiera zbiór par kluczy, gdzie każda para składa się z klucza prywatnego i publicznego. Klucz prywatny (k) to liczba zwykle wybierana losowo. Z klucza prywatnego wybieramy mnożnik krzywej eliptycznej, aby wygenerować klucz publiczny (K). Z publicznego klucza (K), wykorzystujemy jednokierunkową funkcję szyfrowania hashem, aby wygenerować adres bitcoin (A). W tej części zacznijmy generować klucze prywatne oraz przyjrzymy się obliczeniom w oparciu o krzywą eliptyczną wykorzystywaną w celu jego przekształcenia na klucz publiczny; na koniec wygenerujemy adresy bitcoin w oparciu o klucze publiczne. Relacja pomiędzy kluczem prywatnym, publicznym i adresem bitcoin jest przedstawiona na Rys. 4-1.



Rys. 4-1. Klucz prywatny, publiczny oraz adres bitcoin

Klucze prywatne

Klucz prywatny to po prostu losowo wybrana liczba. Własność oraz kontrola nad kluczem prywatnym to fundament kontroli użytkownika nad wszystkimi funduszami skojarzonymi z danym adresem bitcoin. Klucz prywatny jest wykorzystywany do utworzenia podpisów, które są niezbędne do wydatkowania bitcoinów weryfikując saldo środków wykorzystywanych w danej transakcji. Klucz prywatny musi pozostać sekretny w każdej sytuacji, ponieważ jego ujawnienie stronom trzecim jest równoważne przekazaniu kontroli nad bitcoinami zabezpieczonymi tym kluczem. Klucz prywatny musi być także wspierany i chroniony przed przypadkową utratą, ponieważ jego zagubienie uniemożliwia jego odzyskanie i nieodwracalną utratę środków, które chroni.



Klucz prywatny bitcoin to po prostu liczba. Można losowo dokonać wyboru klucza prywatnego wykorzystując monetę, ołówek oraz papier: wystarczy podrzuścić monetę 256

Razy, aby uzyskać cyfry binarne losowego klucza prywatnego, który można wykorzystać w portfelu bitcoin. Teraz można wygenerować klucz publiczny na podstawie klucza prywatnego.

Generowanie klucza prywatnego w oparciu o liczbę losową

Pierwszy i najważniejszy krok w generowaniu kluczy to znalezienie bezpiecznego źródła entropii, lub inaczej losowości/przypadkowości. Tworzenie klucza bitcoin jest zasadniczo takie samo, jak "wybieranie liczby od 1 do 2^{256} ". Metoda wybrana w celu wybrania tych liczb nie ma znaczenia o ile jest nieprzewidywalna lub niepowtarzalna. Oprogramowanie bitcoin wykorzystuje generator liczb losowych stanowiące część systemów operacyjnych, aby stworzyć 256 bitów entropii (przypadkowości). Z reguły, generator liczb losowych OS jest inicjalizowany przez człowieka i dlatego system może zwrócić się z prośbą o poruszanie myszką przez kilka chwil. Dla paranoików, nic nie równa się z kostką, ołówkiem i papierem.

Mówiąc dokładniej, klucz prywatny to może być dowolna liczba z przedziału od 1 do $n - 1$, gdzie n oznacza stałą ($n = 1,158 * 10^{77}$, nieco mniej niż 2^{256}) definiowaną jako szereg krzywych eliptycznych wykorzystywanych w bitcoin (patrz: "[Elliptic Curve Cryptography Explained](#)"). Aby utworzyć taki klucz, losowo wybieramy liczbę 256-bitową oraz sprawdzamy, czy jest mniejsza niż $n - 1$. W kategoriach programistycznych, z reguły osiągamy to wprowadzając dłuższy strumień losowych bitów, zebranych z zabezpieczonego szyfrem źródła bitów do algorytmów hash SHA256, który w wygodny sposób wygeneruje liczbę 256-bitową. Jeżeli wynik wynosi mniej niż $n - 1$, mamy właściwy klucz prywatny. W przeciwnym razie po prostu wyprobujemy kolejną liczbę losową.



Nie opracujcie swojego własnego kodu w celu utworzenia liczby losowej i nie wykorzystujcie "prostego" generatora liczb losowych oferowanego przez Wasz własny język programowania. Wykorzystujcie szyfrowo zabezpieczony generator liczb losowych (CSPRNG) ze źródłem odpowiednio przypadkowym. Zapoznajcie się z treścią dokumentacji bibliotek generator liczb losowych przez Was wybranych w stworzenia odpowiednio bezpiecznego poziomu szyfrowania. Poprawne zaimplementowanie CSPRNG jest krytyczne dla bezpieczeństwa kluczy.

Poniżej przedstawiamy losowo wygenerowany klucz prywatny (k) przedstawiony w formacie szesnastkowym (256 - liczby binarne przedstawione w postaci 64 grup szesnastkowych każda o wielkości 4 bitów):

1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD



Ilość miejsca niezbędna dla klucza prywatnego bitcoin 2^{256} jest niewyobrażalnie duża. W kategoriach liczb dziesiętnych wynosi 10^{77} . Widoczne spektrum zawiera 10^{80} atomów.

Aby wygenerować klucz za pomocą klienta Bitcoin Core client (patrz: [Rozdział 3](#)), można użyć polecenia `getnewaddress`. Ze względów bezpieczeństwa na ekranie wyświetla się jedynie klucz publiczny, a nie prywatny. Aby poprosić o klucz prywatny należy użyć polecenia `dumpprivatekey`. Polecenie `dumpprivatekey` pokazuje klucz prywatny w formacie sumy weryfikującej Base58 nazywanej *Wallet Import Format (WIF)*, z którym zapoznamy się nieco bardziej szczegółowo w części „[Formaty kluczy prywatnych](#)”. Poniżej znajduje się przykład generowania i wyświetlania klucza prywatnego w oparciu o te dwa polecenia:

```
$ bitcoind getnewaddress  
1J7mdg5rbQyUHENYdx39WWK7fsLpEoXZy  
$ bitcoind dumpprivatekey 1J7mdg5rbQyUHENYdx39WWK7fsLpEoXZy  
KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawrtJ
```

Polecenie `dumpprivatekey` otwiera portfel i ekstrahuje z niego klucz prywatny, który został wygenerowany przez polecenie `getnewaddress`. Nie ma innej możliwości rozróżnienia przez `bitcoind` klucza prywatnego od klucza publicznego, chyba że obydwa są zapisane w portfelu.



Polecenie `dumpprivatekey` nie generuje klucza prywatnego z publicznego, ponieważ nie jest to możliwe. Polecenie po prostu ujawnia klucz prywatny, który jest już znany portfelowi i który został wygenerowany przez polecenie `getnewaddress`.

Można także wykorzystać narzędzie polecień `sx` (patrz: „[Libbitcoin and sx Tools](#)” na stronie 56), aby wygenerować i wyświetlić klucze prywatne używając polecenia `sx newkey`:

```
$ sx newkey  
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

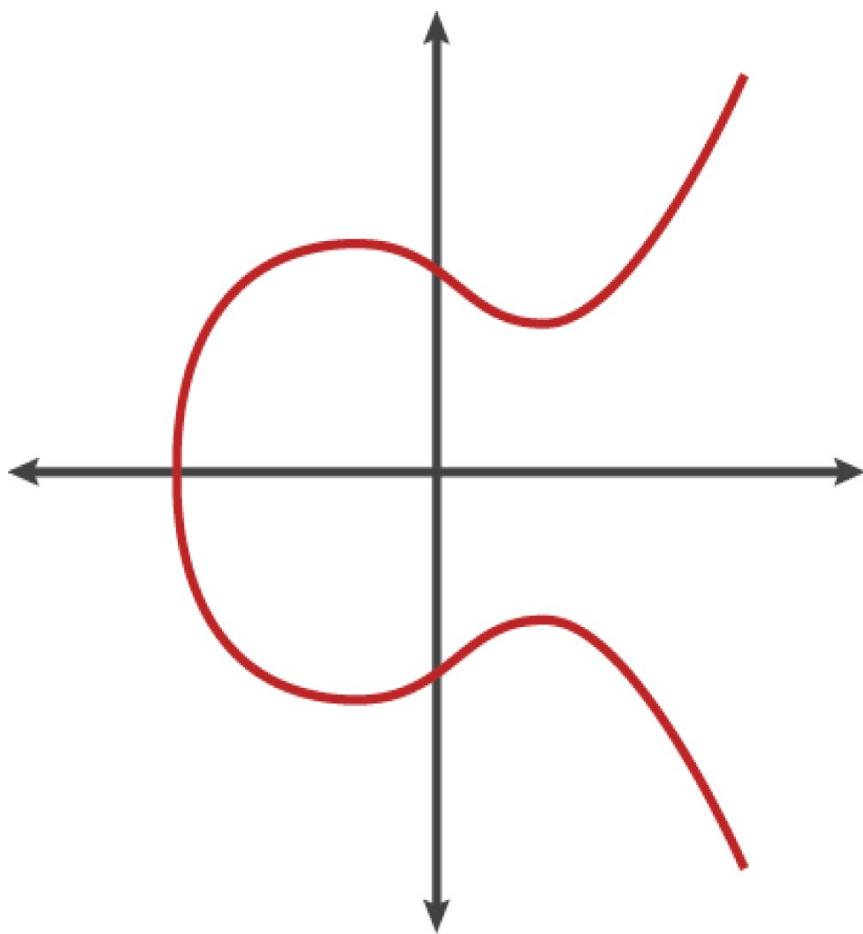
Klucze publiczne

Klucz publiczny jest obliczany na podstawie klucza prywatnego wykorzystującego mnożnik krzywej eliptycznej, który jest nieodwracalny: $K = k * G$, gdzie k to klucz prywatny, G to punkt stały nazywany *punktem generatora* a K to wynikowy klucz publiczny. Operacja odwrotna znana jako “znajdowanie unikalnego algorytmu” — obliczająca k , jeśli K jest znane jest tak trudnie jak wypróbowywanie wszystkich potencjalnych wartości k , tzn., wyszukiwanie w oparciu o siłę fizyczną. Zanim pokażemy w jaki sposób generować klucz publiczny z klucza prywatnego przyjrzymy się szyfrowaniu krzywą eliptyczną nieco bliżej.

Wyjaśnienie szyfrowania krzywą eliptyczną

Szyfrowanie krzywą eliptyczną to rodzaj asymetrycznego lub w oparciu o szyfrowanie kluczem publicznym, w oparciu o problem odrębnych logarytmów wyrażony jako dodanie lub pomnożenie punktów na krzywej eliptycznej.

Rys. 4-2 to przykład krzywej eliptycznej zbliżony do tego, który jest wykorzystywany przez bitcoin.



Rys. 4-2. Krzywa eliptyczna

Bitcoin wykorzystuje szczególny typ krzywej eliptycznej oraz zestaw stałych matematycznych zdefiniowanych w standardzie nazywanym secp256k1, opracowanym przez National Institute of Standards and Technology (NIST).

Krzywa secp256k1 jest zdefiniowana przez niżej wymienioną funkcję, która produkuje krzywą eliptyczną:

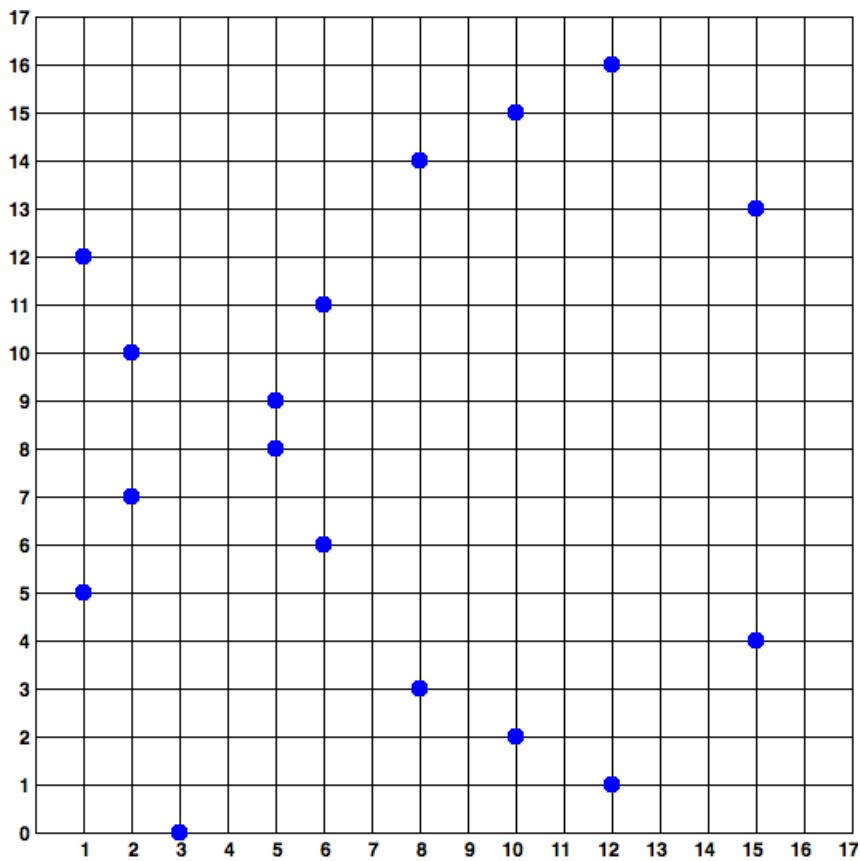
$$y^2 = x^3 + 7 \text{ nad } (\mathbb{F}_p)$$

lub

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

$\bmod p$ (liczba pierwsza modułu liczb pierwszych p) oznacza, że krzywa jest ponad skończony polem szeregu głównego p , także zapisywana jako \mathbb{F}_p , gdzie $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$, i jest bardzo dużą liczbą pierwszą.

Ponieważ krzywa ta jest zdefiniowana skończonym polem liczb pierwszych zamiast liczbami rzeczywistymi, przypomina wzór kropek rozproszonych w dwóch wymiarach utrudnia wizualizację. Nie mniej jednak, matematyka jest identyczna z krzywą eliptyczną dla liczb rzeczywistych. Jako przykład, Rys. 4-3, pokazujemy tę samą krzywą eliptyczną nad znacznie mniejszym skończonym polem szeregu 17, pokazując wzór kropek na siatce. Krzywa eliptyczna secp256k1 bitcoin może być uznana za o wiele bardziej złożony wzór kropek na niewyobrażalnie dużej siatce.



Rys. 4-3. Kryptografia krzywych eliptycznych: wizualizacja krzywej eliptycznej nad $F(p)$, z $p=17$

Dla przykładu: poniżej mamy punkt P o współrzędnych (x, y) który znajduje się na krzywej secp256k1. Można sprawdzić samemu korzystając z Python:

```
P =
(55066263022277343669578718895168534326250603453777594175500187360389116729240,
32670510020758816978083085130507043184471273380659243275938904335757337482424)
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
```

W matematyce krzywych eliptycznych istnieje punkt nazywany "punktem w nieskończoności", który w przybliżeniu spełnia rolę w zera w dodawaniu. W przypadku komputerów jest on czasami wyrażony jako $x = y = 0$ (co nie spełnia równania krzywej eliptycznej, ale jest to prosty, oddzielny przypadek, który można sprawdzić).

Istnieje także wyrażenie $a +$ operator nazywane "dodawaniem", które ma pewne cechy zbliżone do tradycyjnego dodawania liczb rzeczywistych, których uczą się dzieci w szkole. Mają dwa punkty P_1 oraz P_2 na krzywej eliptycznej można wyznaczyć trzeci punkt $P_3 = P_1 + P_2$ na tej samej krzywej.

Z geometrycznego punktu widzenia, ten trzeci punkt P_3 jest obliczany poprzez połączenie punktów P_1 i P_2 linią, która będzie przecinać krzywą dokładnie w jednym dodatkowym miejscu. Miejsce to nazywamy $P_3' = (x, y)$. Następnie nanosimy go na oś x i uzyskujemy $P_3 = (x, -y)$.

Istnieje kilka szczególnych przypadków, które wyjaśniają potrzebę stosowania punktu w nieskończoności".

Jeżeli P_1 oraz P_2 to ten sam punkt, linia "pomiędzy" P_1 i P_2 powinna tworzyć styczną na tej krzywej w punkcie P_1 . Będzie ona przecinać krzywą dokładne w jednym, nowym punkcie. Można wykorzystać techniki stosowane w calculus w celu określenia nachylenia krzywej. Co dziwne – techniki te sprawdzają się nawet, jeśli ograniczymy nasze zainteresowanie do punktów na krzywej od współrzędnych wyrażonych dwiema liczbami całkowitymi!

W niektórych przypadkach (np. jeżeli P_1 i P_2 mają te same wartości x, ale różne y), styczna będzie biec pionowo, w którym to przypadku $P_3 = \text{"punkt w nieskończoności"}$.

Jeżeli P_1 jest "punktem w nieskończoności", to suma $P_1 + P_2 = P_2$. Podobnie, jeżeli P_2 jest punktem w nieskończoności, to $P_1 + P_2 = P_1$. Przykład ten ilustruje, dlaczego punktem w nieskończoności pełni rolę 0.

Okazuje się, że + ma charakter łączny, co oznacza, że $(A+B) C = A(B+C)$, czyli że możemy zapisać wyrażenie, jako $A+B+C$ nie stosując nawiasów bez jakiegokolwiek dwuznaczności.

Zdefiniowawszy dodawanie możemy zdefiniować mnożenie w sposób standardowy, który będzie rozszerzać działanie dodawania. Dla punktu P na krzywej eliptycznej, jeżeli k stanowi liczbę całkowitą, to $kP = P + P + P + \dots + P$ (k razy). Należy zauważać, że k jest niekiedy mylnie nazywane „wykładnikiem” w tym przypadku.

Generowanie klucza publicznego

Zaczynając od klucza prywatnego w postaci losowo wygenerowanej liczby k , mnożymy ją przez wcześniej określony punkt na krzywej nazywany *generatorem punktu G*, aby wyznaczyć kolejny punkt w innym miejscu krzywej, który będzie odpowiadać kluczowi publicznemu K . Punkt generatora jest określony jako część standardu secp256k1 i ma zawsze tę samą wartość dla wszystkich kluczy w bitcoin:

$$K = k * G$$

Gdzie k jest kluczem prywatnym, G jest punktem generatora, a K to wynikowy klucz publiczny – punkt na krzywej. Ponieważ punkt generatora jest zawsze taki sam dla wszystkich użytkowników, klucz prywatny k pomnożonym przez G będzie zawsze generować ten sam klucz publiczny K . Związek pomiędzy k i K jest stały, ale może być obliczony wyłącznie w jednym kierunku, tzn. k do K . Dlatego adresy bitcoin (obliczone w oparciu o K) mogą być ujawniane różnym osobom bez ujawniania klucza prywatnego użytkownika (k).



Klucz prywatny można zamienić na klucz publiczny, ale klucz publicznego nie można zamienić z powrotem na klucz prywatny, ponieważ matematyka działa tylko w jedną stronę.

Stosując mnożenie w oparciu o krzywą eliptyczną, wykorzystujemy klucz prywatny k wcześniej wygenerowany i mnożymy go przez generator punktu G w celu znalezienia klucza publicznego K :

$$K = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD * G$$

Klucz publiczny jest zdefiniowany jako punkt $K = (x, y)$:

$$K = (x, y)$$

Gdzie:

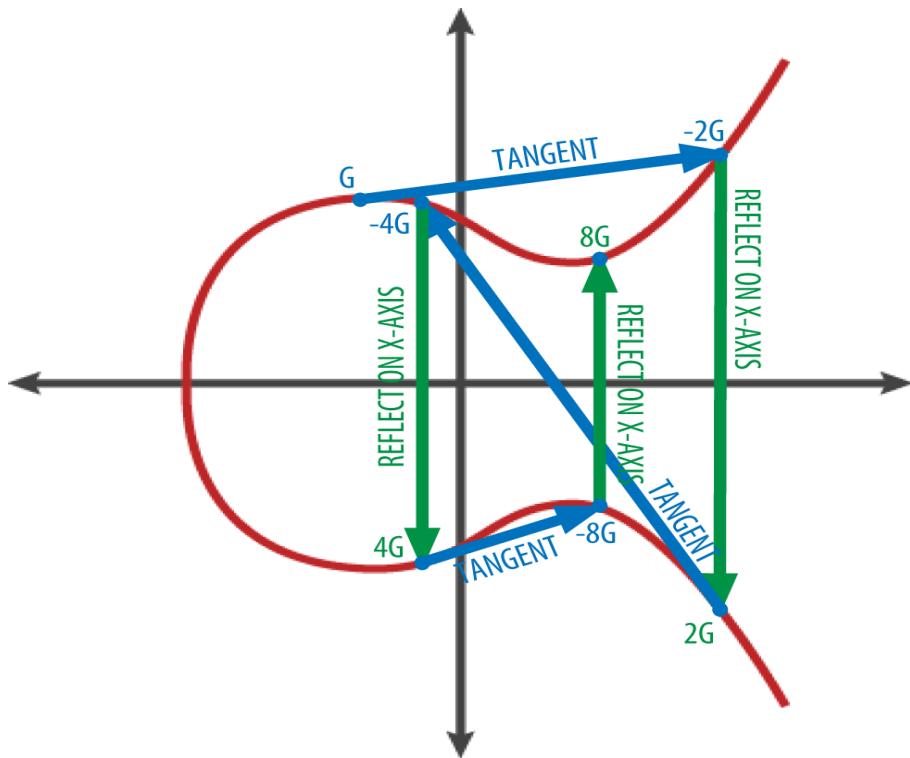
$x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2EFFF579DC341A$
 $y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB$

Aby zwizualizować mnożenie punktu przez liczbę całkowitą, zastosujemy prostszą krzywą eliptyczną z liczbami rzeczywistymi — należy jednak pamiętać, że reguły matematyczne są zawsze takie same. Naszym celem jest znalezienie wielokrotności kG generatora punktu G . Jest to czynność identyczna, jak dodawanie G do siebie kilka razy z rzędu. W przypadku krzywych eliptycznych, dodawanie punktu do siebie samego jest tożsame z kreślenie linii stycznej do punktu i znalezienie punktu ponownego przecięcia z krzywą i następnie odwzorowanie tego punktu na osi x.

Rys. 4-4 przedstawia proces wyznaczania punktów $G, 2G, 4G$ jako działanie geometryczne na krzywej.



Większość programów wykorzystuje [OpenSSL cryptographic library](#) w celu wykonania obliczeń w oparciu o krzywą eliptyczną. Na przykład, aby obliczyć klucz publiczny wykorzystuje się funkcję `EC_POINT_mul()`.



Rys. 4-4. Kryptografia krzywej eliptycznej: wizualizacja mnożenia punktu G przez liczbę całkowitą na krzywej eliptycznej

Adresy bitcoin

Adres bitcoin to strumień cyfr i znaków, które są udostępniane każdej osobie, która chce dokonać płatności na nasze konto. Adresy te są generowane w oparciu o klucze publiczne składające się ze strumienia liczb i liter zaczynających się od cyfry "1". Poniżej przedstawiony jest przykład adresu bitcoin:

1J7mdg5rbQyUHENYdx39WWK7fsLpEoXZy

Adres bitcoin to element, który w transakcji występuje w charakterze “odbiorcy” środków. Porównując transakcje bitcoin do czeku, adres bitcoin to beneficjent, czyli treść wpisana po zwrocie „Płatność na żądanie”. W przypadku tej formy rozliczeń beneficjentem może być posiadacz rachunku bankowego, czyli firma lub instytucja, niekiedy transakcja może być czysto gotówkowa. Ponieważ czek nie zawsze muszą odnosić się do szczególnego konta, ale raczej wykorzystują nazwę abstrakcyjną, jako odbiorcę środków, rozwiązanie to czyni czek bardzo elastycznym instrumentem płatniczym. Transakcje bitcoin wykorzystują podobny typ abstrakcji – adresy bitcoin, aby zapewnić wysoki poziom elastyczności. Adres bitcoin może reprezentować właściciela pary kluczy prywatnych/publicznych lub reprezentować coś innego, np. skrypt płatności, jak zobaczymy w rozdziale **“Wartość Pay-to-Script-Hash (P2SH)”**. W tej chwili jednak rozważmy prosty przypadek adresu bitcoin stanowiącego i utworzonego w oparciu o klucz publiczny.

Adres bitcoin jest obliczany w oparciu o klucz publiczny w oparciu o jednokierunkowe funkcje hashujące. „Algorytm hashujący” lub po prostu „algorytm hash” to funkcja jednokierunkowa generująca odcisk palca lub „hash” bloku wejściowego dowolnej wielkości. Szyfrowanie w oparciu o funkcje hash są powszechnie stosowane w systemie bitcoin: do budowania adresów bitcoin, w adresów skryptowych oraz w górnictwem algorytmie proof-of-work. Algorytmy wykorzystywane w celu zbudowania adresów bitcoin w oparciu o klucze publiczne to Secure Hash Algorithm (SHA) oraz RACE Integrity Primitives Evaluation Message Digest (RIPEMD), w szczególności SHA256 i RIPEMD160.

Zaczynając od klucza publicznego K obliczamy wartość hash SHA256, a następnie hash RIPEMD160 wyniku uzyskując 160-bitową (20-bajtową) liczbę:

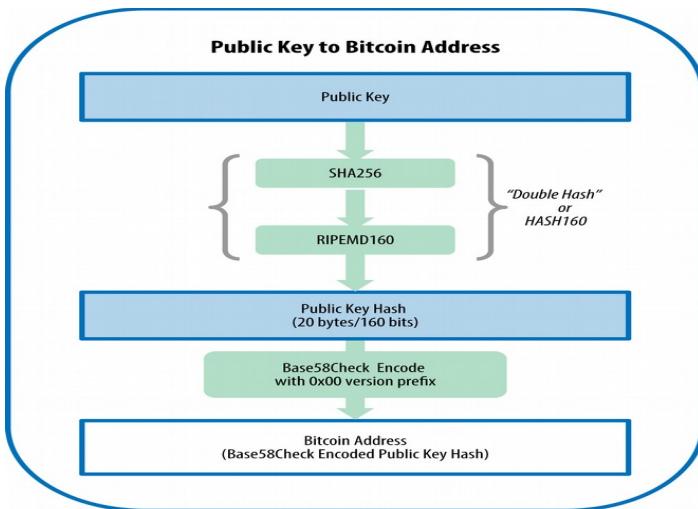
$$A = \text{RIPEMD160}(\text{SHA256}(K))$$

Gdzie K to klucz publiczny, a A to wynikowy adres bitcoin.



Adres bitcoin *nie* jest tożsamy z kluczem publicznym. Adresy bitcoin są budowane w oparciu o klucz publiczny wykorzystując funkcję jednokierunkową.

Adresy bitcoin są niemal zawsze ujawniane użytkownikom w postaci szyfrowanej “Base58Check” (patrz: **„Szyfrowanie Base58 i Base58Check”**), która oparta jest na 58 znakach (system liczbowy Base58) oraz sumie kontrolnej, co ułatwia odczyt, pozwala uniknąć dwuznaczności oraz chroni przed błędami w transkrypcji adresów oraz zapisów. Base58Check jest także wykorzystywany do innych celów w bitcoin, zawsze, gdy pojawia się potrzeba odczytania przez użytkownika i poprawnego zapisania liczby, np. adresu bitcoin, klucza prywatnego, klucza szyfrowanego, czy wartości hash skryptu. W kolejnym rozdziale zapoznamy się z mechaniką szyfrowanie i deszyfrowania Base58Check oraz z wynikowymi reprezentacjami. **Rys. 4-5** ilustruje zamianę klucza publicznego na adres bitcoin.



Rys. 4-5. Klucz publiczny dla adresu bitcoin: konwersja klucza publicznego na adres bitcoin

Szyfrowanie Base58 i Base58Check

Aby przedstawić długie ciągi liczb w sposób skrócony przy wykorzystaniu mniejszej liczby symboli, wiele systemów komputerowych wykorzystuje mieszankę reprezentacji alfanumerycznych o podstawie (lub pozycji) wyższej niż 10. Na przykład, ponieważ konwencjonalny system dziesiętny wykorzystuje 10 liczb od 0 do 9, system szesnastkowy wykorzystuje 16 i litery od A do F pełniące rolę dodatkowych symboli. Liczba przedstawiona w formacie szesnastkowym jest krótsza niż odpowiadająca jej reprezentacja dziesiętna. Jest krótsza, ponieważ reprezentacja Base-64 wykorzystuje 26 małych liter, 26 wielkich liter, 10 cyfr oraz dodatkowe dwa znaki w postaci "+" oraz "/" w celu przesłania danych binarnych na nośniku tekstowym takim, jak email. Base-64 jest najczęściej stosowanym systemem w celu dodania załącznika binarnego do poczty elektronicznej. Base58 to oparty na tekście format szyfrowania binarnego opracowany do wykorzystania w systemie bitcoin oraz innych kryptowalutach. Zapewnia on równowagę pomiędzy skróconą reprezentacją, możliwością odczytywania, a wykrywaniem i zapobieganiem powstawaniu błędów. Base58 to podzbiór Base64, wykorzystujący litery małe i wielkie oraz liczby z pominięciem niektórych znaków, które są często mylone i po wyświetleniu w pewnym typie czcionek mogą być uznane za jednakowe. W szczególności, Base58 to Base64 bez 0 (liczba zero), O (wielkiego o), l (małego L), I (wielkiego i) oraz znaków "\+" oraz "/". Mówiąc prościej, jest to zestaw wielkich i małych liter oraz liczb bez wymienionych czterech znaków (0, O, l, I).

Przykład 4-1. Alfabet Base58 bitcoin

123456789ABCDEFHJKLMNPQRSTUWXYZabcdefghijklmnopqrstuvwxyzijklmnopqrstuvwxyz

Aby zapewnić wyższy stopień zabezpieczeń w związku z błędami literowymi lub transkrypcyjnymi, Base58Check to format szyfrowania Base58, często wykorzystywany w bitcoin, który posiada wbudowany kod weryfikacji błędów. Suma kontrolna to dodatkowe cztery bajty dodawane na końcu szyfrowanych danych. Suma ta jest obliczana na podstawie wartości hash kodowanych danych i w związku z tym może być wykorzystywana do wykrywania i zapobiegania błędów w transkrypcji lub literowym. Dołączona do kodu Base58Check powoduje, że oprogramowanie deszyfrujące obliczy tę sumę danych, a następnie porówna ją z sumą kontrolną wpisaną do kodu. Jeżeli obie sumy nie będą zgodne, oznacza to, że do danych wkradł się błąd a dane Base58Check są nieważne. Pozwala to, na przykład, uniknąć akceptacji błędnie wpisanych adresów

bitcoin przez oprogramowanie portfela jako adresów poprawnych, który w przeciwnym razie spowodowałby utratę środków finansowych.

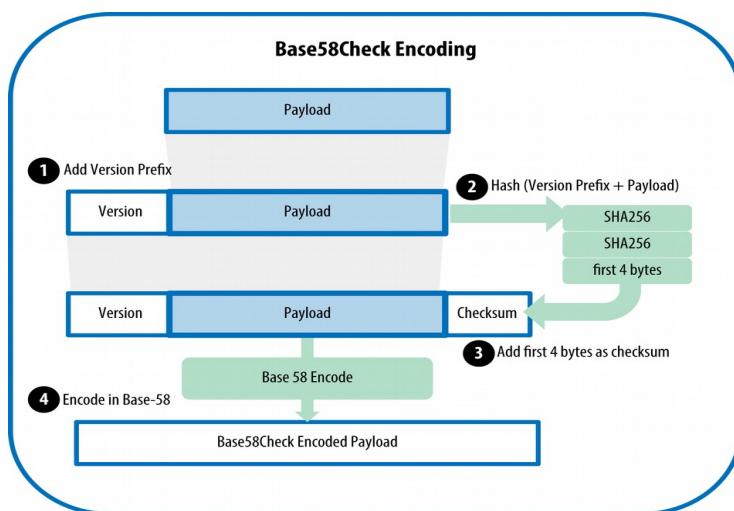
Aby dokonać konwersji danych (liczby) na format Base58Check, musimy najpierw dodać prefiks do danych, nazywany "bajtem wersji", który pomaga w sposób prosty zidentyfikować rodzaj szyfrowanych danych. W odniesieniu do adresów bitcoin, prefiks ten ma wartość zero (0x00 w systemie szesnastkowym), podczas, gdy prefiks wykorzystywany w szyfrowaniu klucza prywatnego ma wartość 128 (0x80 w systemie szesnastkowym). Lista najczęściej stosowanych prefiksów jest przedstawiona w [Tabeli 4-1](#).

Następnie, obliczamy podwójną sumę kontrolną SHA, co oznacza, że stosujemy algorytm hash SHA256 dwukrotnie w odniesieniu do poprzednio uzyskanego wyniku (prefiks oraz dane):

$$\text{checksum} = \text{SHA256}(\text{SHA256}(\text{prefix+data}))$$

Z uzyskanej 32-bajtowej wartości hash (hash-of-a-hash), wykorzystujemy pierwsze cztery bajty. Wykorzystywane są one jako kod weryfikacji błędów lub inaczej suma kontrolna. Wartość ta jest dodawana na końcu strumienia.

Wynik składa się zatem z trzech elementów: prefiksu, danych oraz sumy kontrolnej. Wynik ten jest kodowany w oparciu o wcześniej opisany alfabet Base58. [Rys. 4-6](#) ilustruje proces szyfrowania Base58Check.



Rys. 4-6. Szyfrowanie w systemie Base58Check: Base58, wersja oraz format sumy kontrolnej w celu jednoznacznego szyfrowania danych bitcoin

W bitcoin, większość danych przedstawianych użytkownikowi jest kodowanych w Base58Check, w celu skrócenia zapisu, ułatwienia odczytu oraz wykrywania błędów. Prefiks wersji w szyfrowaniu Base58Check jest wykorzystywany w celu prostego tworzenia rozróżnialnych formatów, które następnie szyfrowane w Base58 zawierają szczególne znaki na początku zaszyfrowanej w Base58Check treści. Znaki te ułatwiają odczyt oraz identyfikację zakodowanych danych oraz możliwości ich wykorzystania. To właśnie odróżnia na przykład adres bitcoin kodowany w Base58Check zaczynający się od 1 od formatu WIF klucza prywatnego szyfrowanego w Base58Check, który zaczyna się od liczby 5. Niektóre przykłady prefiksów wersji oraz wynikowe znaki w Base58 przedstawiono w [Tabeli 4-1](#).

Tabela 4-1. Prefiks wersji w Base58Check i przykładowe wyniki kodowania

Type Typ	Prefiks wersji (hex)	Prefiks wynikowy w Base58
----------	-------------------------	------------------------------

Adres Bitcoin	0x00	1
Adres Pay-to-Script-Hash	0x05	3
Adres Bitcoin Testnet	0x6F	m lub n
Klucz prywatny WIF	0x80	5, K lub L
Klucz prywatny kodowany w BIP38	0x0142	6P
Rozszerzony klucz publiczny BIP32	0x0488B21E	xpub

Prześledźmy kompletny proces budowania adresu bitcoin – od utworzenia klucza prywatnego (punkt na krzywej eliptycznej), aż po adres z podwójnym hashem i na koniec szyfrowanie w Base58Check. Kod C++ w [Przykładzie 4-2](#) pokazuje pełny proces krok po kroku – od klucza prywatnego, aż po adres bitcoin szyfrowany w Base58Check. Przykładowy kod wykorzystuje bibliotekę libbitcoin, która została mówiona w rozdziale „[Alternatywni klienci, Biblioteki oraz zestawy narzędzi](#)” zawierającą funkcje pomocnicze.

Przykład 4-2. Tworzenie adresu bitcoin kodowane w Base58Check z oparciu o klucz prywatny

```
#include <bitcoin/bitcoin.hpp>
int main()
{
    // Private secret key.
    bc::ec_secret secret = bc::decode_hex(
        "038109007313a5807b2ecc082c8c3fb988a973cacf1a7df9ce725c31b14776");
    // Get public key.
    bc::ec_point public_key = bc::secret_to_public_key(secret);
    std::cout << "Public key: " << bc::encode_hex(public_key) << std::endl;

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

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

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

Kod wykorzystuje wcześniej zdefiniowane klucze prywatne i pozwala uzyskać ten sam adres bitcoin za każdym razem, kiedy jest uruchamiany zgodnie z [Przykładem 4-3](#).

Przykład 4-3. Kompilowanie i uruchamianie kodu addr

```
# Compile the addr.cpp code
```

```

$ g++ -o addr addr.cpp ${pkg-config --cflags --libs libbitcoin}
# Run the addr executable
$ ./addr
Public key: 0202a406624211f2abbdc68da3df929f938c3399dd79fac1b51b0e4ad1d26a47aa
Address: 1PRTTajesdNovgne6Ehcd1fpEdX7913CK

```

Formaty kluczy

Zarówno klucze prywatne jak i publiczne można przedstawiać korzystając z różnych formatów. Reprezentacje te zawsze kodują tę samą liczbę nawet jeśli wyglądają inaczej. Formaty są głównie wykorzystywane, aby ułatwić odczyt i transkrypcję kluczy bez popełniania błędów.

Formaty kluczy prywatnych

Klucz prywatny może być przedstawiony w różnych formatach, z których każdy odpowiada tej samej 256-bitowej liczbie. **Tabela 4-2** przedstawia trzy najczęściej stosowane formaty wykorzystywane w celu reprezentacji kluczy prywatnych.

Tabela 4-2. Reprezentacje kluczy prywatnych (format szyfrowania)

Typ	Prefiks	Opis
Hex	Brak	64 cyfry w systemie szesnastkowym
WIF	5	Szyfrowanie w Base58Check: Base58 z prefiksem wersji 128 i sumą kontrolną 32-bitową
WIF-compressed	K lub L	Zgodnie z powyższym z sufiksem 0x01 dodanym przez kodowaniem

Tabela 4-3 przedstawia klucz prywatny generowany w tych trzech formatach.

Table 4-3. Przykład: ten sam klucz, różne formaty

Format	Klucz prywatny
Hex	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
WIF-compressed	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawrtJ

Wszystkie z przedstawionych reprezentacji pokazują różne sposoby przedstawiania tej samej liczby i tego samego klucza prywatnego. Wyglądają inaczej, ale zasadniczo każdy format można przekształcić na inny format.

Deszyfrowanie z Base58Check na hex

Pakiet narzędzi sx (patrz: ["Libbitcoin i narzędzia sx Tools"](#)) ułatwia napisanie skryptów powłoki oraz "rurek" wierszy poleceń, które zmieniają klucze, adresy oraz transakcje bitcoin. Narzędzia sx mogą być wykorzystywane do deszyfrowania formatu Base58Check w wierszu poleceń.

Używamy komendy base58check-decode:

```
$ sx base58check-decode 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd 128
```

W wyniku uzyskujemy klucz szesnastkowy, po którym następuje prefiks wersji Wallet Import Format (WIF)128.

Szyfrowanie z hex do Base58Check

Aby zaszyfrować dane w Base58Check (czynność odwrotna w stosunku do poprzedniej), podajemy szesnastkowy klucz prywatny, po którym następuje prefiks wersji Wallet Import Format (WIF)128:

```
$ sx base58check-encode  
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd 128  
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

Szyfrowanie z systemu szesnastkowego (klucz skompresowany) na Base58Check

Aby zakodować dane w Base58Check w postaci skompresowanego klucza prywatnego (patrz rozdział: „[Skompresowany klucz prywatny](#)”) dodajemy sufiks 01 na końcu klucza szesnastkowego, a następnie kodujemy w sposób przedstawiony powyżej:

```
$ sx base58check-encode  
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01 128  
KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawrtJ
```

Uzyskany w ten sposób format skompresowanego WIF zaczyna się od litery “K”. Oznacza ona, że klucz prywatny ma sufiks “01” i będzie wykorzystany wyłącznie w celu utworzenia skompresowanego klucza publicznego (patrz część: „[Skompresowane klucze publiczne](#)”).

Formaty kluczy publicznych

Klucze publiczne są także przedstawiane na różne sposoby, przeważnie jako *skompresowane* lub *nieskompresowane* klucze publiczne.

Jak widzieliśmy wcześniej, klucz publiczny to punkt na krzywej eliptycznej składający się z pary współrzędnych (x, y). Z reguły jest on przedstawiany z prefiksem 04, po którym następują dwie liczby 256-bitowe, jedna dla współrzędnej x punktu, a druga dla współrzędnej y . Prefiks 04 jest wykorzystywany rozróżniania nieskompresowanych kluczy publicznych zaczynających się od 02 lub 03.

Poniżej przedstawiono klucz publiczny wygenerowany przez wcześniej utworzony klucz prywatny przedstawiony jako współrzędne x i y :

```
x = F028892BAD7ED57D2FB57BF33081D5CF6F9ED3D3D7F159C2E2FFF579DC341A  
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Poniżej przedstawiony jest ten sam klucz publiczny w postaci liczby 520-bitowej (szesnastkowe cyfry 130) z prefiksem 04, po którym następują współrzędne x oraz y , zapisane jako 04 x y:

```
K = 04F028892BAD7ED57D2FB57BF33081D5CF6F9ED3D3D7F159C2E2FFF579DC341A<?pdf-cr?  
>07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

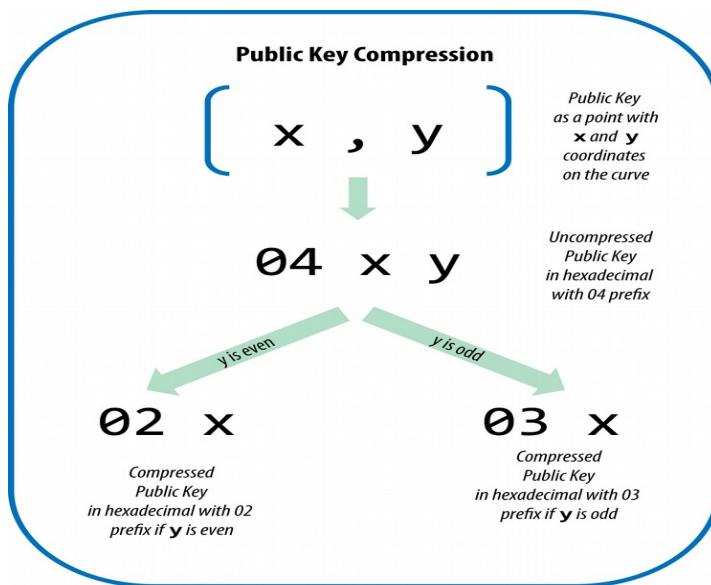
Skompresowane klucze publiczne

Skompresowane klucze publiczne zostały wprowadzone do bitcoin w celu zredukowania rozmiaru transakcji i zachowania przestrzeni na dysku na węzły przechowujące bazę danych łańcucha bloków bitcoin. Większość transakcji zawiera klucz publiczny niezbędny do zweryfikowania właściciela oraz wydatkowania bitcoinów. Każdy klucz publiczny wymaga 520 bitów (prefiks \+ x \+ y), które pomnożone przez kilkaset transakcji występujących w bloku lub dziesiątki tysięcy transakcji realizowanych dziennie znaczco powiększają zasoby danych w łańcuchu blokowym.

Zgodnie z treścią części „[Klucze publiczne](#)”, klucz publiczny to punkt (x, y) na krzywej eliptycznej. Ponieważ krzywa ta wyraża funkcje matematyczne, punkt na krzywej reprezentuje rozwiązanie dla równania li dlatego znając współrzędną x możemy obliczyć wartość współrzędnej y rozwiązując równanie $y^2 \bmod p = (x^3 + 7) \bmod p$

p. Umożliwia to nam zapisywanie tylko wartości współrzędnej x w punkcie klucza publicznego z pominieniem wartości współrzędnej y oraz zredukowaniem wielkości klucza oraz miejsca niezbędnego do przechowywania klucza w postaci 256 bitowej. Niemal 50% redukcja rozmiaru każdej transakcji za każdym razem pozwala zapisać wiele danych!

Ponieważ nieskompresowane klucze publiczne mają prefix 04, skompresowany klucz publiczny zaczyna się od prefiksu 02 lub 03. Sprawdźmy, dlaczego mogą występować dla możliwe prefiksy: ponieważ lewa z lewej strony równania mamy wyrażenie y^2 , oznacza to, że rozwiązaniem dla y jest pierwiastek kwadratowy, który może mieć wartość dodatnią lub ujemną. Wizualnie oznacza to, że wynikowa wartość współrzędnej y może powyżej lub poniżej osi x . Jak widać na wykresie krzywej eliptycznej na Rys. 4-2, krzywa ta jest symetryczna, ponieważ stanowi lustrzane odbicie osi x . Chociaż możemy pominąć współrzędną y musimy zapisać znak y (dodatni lub ujemny), innymi słowy, musimy pamiętać, czy znajdował się nad czy pod osią x , ponieważ każda z tych opcji reprezentuje inny punkt i klucz publiczny. Obliczając krzywą eliptyczną w systemie obliczeń binarnych na skończonym polu w porządku głównym p , współrzędna y jest liczbą parzystą lub nieparzystą, która odpowiada znakowi dodatniemu/ujemnemu zgodnie z wcześniejszym omówieniem. Dlatego, aby rozróżnić dwie potencjalne wartości y , zapisujemy skompresowany klucz publiczny z prefiksem 02, w przypadku parzystej wartości y oraz 03 dla wartości nieparzystej umożliwiając oprogramowaniu poprawne wywnioskowanie wartości współrzędnej y na podstawie koordynaty x oraz zdekompresowanie klucza publicznego w oparciu o pełne wartości współrzędnych punktu. Kompresja klucza publicznego jest zilustrowana na Rys. 4-7.



Rys. 4-7. Kompresja klucza publicznego

Poniżej przedstawiamy ten sam klucz publiczny, który wygenerowaliśmy poprzednio jako skompresowany klucz publiczny przechowywany w postaci 264 bitowej (cyfry w system szesnastkowym 66) z prefiksem 03 wskazującym, że współrzędna y jest nieparzysta:

$K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A$

Ten skompresowany klucz publiczny odpowiada temu samemu kluczowi prywatnemu, co oznacza, że jest wygenerowany w oparciu o ten sam klucz prywatny. Nie mniej jednak różni się on od nieskompresowanego klucza publicznego. Co więcej, jeżeli skonwertujemy ten skompresowany klucz publiczny na adres bitcoin za pomocą funkcji podwójnego hasha (RIPEMD160(SHA256(K))) to wygenerujemy różne adresy bitcoin. Może być to nieco mylące, ponieważ oznacza to, że jeden prywatny klucz może służyć do wygenerowania klucza

publicznego wyrażonego w dwóch różnych formatach (skompresowanym i nieskompresowanym), które generują dwa różne adresy bitcoin. Nie mniej jednak, klucz prywatny jest identyczny dla obu adresów bitcoin.

Skompresowane klucze publiczne stopniowo stają się domyślne u wszystkich klientów bitcoin, co ma znaczący wpływ na zredukowanie wielkości treści i tym samym łańcuchów blokowych. Nie mniej jednak jeszcze nie wszyscy klienci wspierają skompresowane klucze publiczne. Nowsi klienci wspierający klucze skompresowane muszą także zawierać transakcje ze starszymi klientami, które nie wspierają skompresowanych kluczy publicznych. Jest to szczególnie istotne w sytuacji, gdy aplikacja portfelowa importuje prywatne klucze z innej aplikacji bitcoin, ponieważ nowy portfel wymaga zeskanowania łańcucha blokowego, aby znaleźć transakcje odpowiadające tym zaimportowanym kluczom. Jakich adresów bitcoinowych powinien szukać portfel bitcoin? Adresy bitcoin generowane przez nieskompresowane klucze publiczne lub adresy bitcoin wygenerowane przez skompresowane klucze publiczne? Obydwa są ważnymi adresami bitcoin i mogą być podpisane kluczem prywatnym, ale są to zupełnie różne adresy!

Aby rozwiązać ten problem, podczas eksportu kluczy prywatnych z portfela, Wallet Import Format wykorzystywany do ich reprezentacji jest implementowany w inny sposób w nowszych portfelach bitcoin wskazując, że te klucze prywatne zostały wykorzystane do wygenerowania *skompresowanych* kluczy publicznych i tym samym *skompresowanych* adresów bitcoin. To umożliwia portfelowi importującemu rozróżnienie pomiędzy kluczami prywatnymi ze starszych i nowszych portfeli oraz przeszukiwanie łańcucha blokowego w celu znalezienia transakcji z adresami bitcoin odpowiadającym nieskompresowanym lub skompresowanym kluczom publicznym, odpowiednio. Przyjrzyjmy się teraz jak to działa nieco bardziej szczegółowo w dalszej części rozdziału.

Skompresowane klucze prywatne

Jak na ironię, termin "skompresowany klucz prywatny" jest mylący, ponieważ kiedy klucz prywatny jest eksportowany w postaci skompresowanej w WIF jest on tak naprawdę o jeden bajt *dłuższy* niż klucz prywatny "nieskompresowany". Wynika to z faktu, że ma on dodany prefiks 01, który oznacza, że pochodzi on z nowego portfela i powinien być wykorzystywany w celu wygenerowania skompresowanych kluczy publicznych. Klucze prywatne nie są skompresowane i nie mogą być skompresowane. Termin "skompresowany klucz prywatny" w rzeczywistości oznacza "prywatny klucz, w oparciu o który powinny być generowane skompresowane klucze publiczne „podczas gdy „nieskompresowany klucz prywatny” w istocie oznacza „klucz prywatny w oparciu o który generowane są nieskompresowane klucze publiczne”. Należy odnosić się jedynie do formatu eksportu jako "skompresowanego w WIF" lub "WIF" a nie odnosić się do kluczy prywatnych jako "skompresowanych", aby uniknąć dalszego zamieszania.

Trzeba pamiętać, że formatów tych *nie* można stosować zamiennie. W nowszym portfelu, który implementuje skompresowane klucze publiczne, klucz prywatny będzie zawsze eksportowany jako skompresowany w WIF (z prefiksem K lub L). Jeżeli portfel jest starszy i nie zawiera skompresowanych kluczy publicznych, klucze prywatne będą eksportowane zawsze jako WIF (z prefiksem 5). Chodzi o to, aby zasygnalizować portfelowi importującemu te klucze prywatne, czy musi przeszukać łańcuch blokowy w celu znalezienia skompresowanych lub nieskompresowanych kluczy publicznych i adresów.

Jeżeli portfel bitcoin jest w stanie zaimplementować skompresowane klucze publiczne, to będzie je wykorzystywać we wszystkich transakcjach. Klucze prywatne w portfelu będą wykorzystywane w celu wygenerowania punktów kluczy publicznych na krzywej, które będą skompresowane. Skompresowane klucze publiczne będą następnie wykorzystane do utworzenia adresów bitcoin podawanych w transakcjach. Importując prywatne klucze z portfela, który

Implementuje skompresowane klucze publiczne format Wallet Import Format jest modyfikowany przez dodanie jednobajtowego sufiku 01 do klucza prywatnego. Wynikowy klucz prywatny kodowany w Base58Check jest nazywany skompresowanym w WIF i zaczyna się od litery

K lub L, a nie od “5”, jak to ma miejsce w przypadku kluczy kodowanych w WIF (nieskompresowanych) ze starszych portfeli.

Tabela 4-4 przedstawia ten sam klucz kodowany w WIF oraz w formatach skompresowanych w WIF.

Table 4-4. Przykład: ten sam klucz, różne formaty

Format	Klucz prywatny
Hex	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Skompresowany w hex-compressed	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD_01
Skompresowany w WIF	KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawrtJ



“Skompresowany klucz publiczny” to niewłaściwa nazwa! One nie są skompresowane; format skompresowany w WIF oznacza, że powinny one być wykorzystywane w celu wygenerowania skompresowanych kluczy publicznych oraz odpowiadających im adresów bitcoin. Jak na ironię, szyfrowany klucz prywatny skompresowany w WI jest o jeden bajt dłuższy, ponieważ ma dodany sufiks 01 w celu rozróżnienia go od postaci nieskompresowanej.

Wdrażanie kluczy i adresów w Python

Najbogatszą biblioteką bitcoin w Python jest [pybitcointools](#) opracowana przez Vitalika Buterina. W [Przykładzie 4-4](#), wykorzystujemy bibliotekę pybitcointools (zimportowaną jako “bitcoiny”), aby wygenerować i wyświetlić klucze i adresy w różnych formatach.

Przykład 4-4. Generowanie i formatowanie kluczy a adresów przy wykorzystaniu biblioteki pybitcointools

```
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.base10_multiply(bitcoin.G, decoded_private_key)
print "Public Key (x,y) coordinates is:", public_key

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

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

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

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

```

Przykład 4-5 dane wynikowe z efekcie uruchomienia kodu.

Przykład 4-5. Uruchomienie 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:

```

```
045c0de3b9c8ab18dd04e3511243ec2952002dbfad864b9628910169d9b9b00ec4
```

```

243bcefdd4347074d44bd7356d6a53c495737dd96295e2a9374bf5f02ebfc176
Compressed Public Key (hex) is:
025c0de3b9c8ab18dd04e3511243ec2952002dbfad864b9628910169d9b9b00ec
Bitcoin Address (b58check) is:
1thMirt546nngXqyPEz532S8fLwbozud8
Compressed Bitcoin Address (b58check) is:
14cxpo3MBCYYWCgF74SWTdcmxipnGUspw3

```

Przykład 4-6 to kolejny przykład wykorzystujący bibliotekę Python ECDSA w celu obliczeń w oparciu o krzywą eliptyczną oraz bez wykorzystywania wyspecjalizowanych bibliotek bitcoin.

Przykład 4-6. Skrypt przedstawiający obliczenia kluczy bitcoin w oparciu o krzywą eliptyczną

Przykład 4-7 przedstawia blok wynikowy wygenerowany przez ten skrypt.

Przykład 4-7. Instalacja biblioteki Python ECDSA i uruchomienie skryptu `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:
(7004885353186717948957750497606966272382583471322935454624595540007269312627,
105262206478686743191060800263479589329920209527285803935736021686045542353380)
BTC public key: 029ade3effb0a67d5c8609850d797366af428f4a0d5194cb221d807770a1522873
```

Portfele

Portfele to pojemniki dla kluczy prywatnych zwykle implementowane jako pliku ustrukturyzowane lub proste bazy danych. Inna metoda tworzenia kluczy to metoda *deterministycznego generowania kluczy*. W tym przykładzie generowany jest każdy nowy klucz prywatny za pomocą jednokierunkowej funkcji hashującej w oparciu o wcześniejszy klucz prywatny łączący je w sekwencję. Tak długo, jak można te sekwencje odtworzyć wystarczy znać tylko pierwszy klucz (klucz źródłowy lub *główny*), aby wygenerować wszystkie klucze. W tej części zapoznamy się z różnymi metodami generowania kluczy oraz strukturami portfeli, które są zbudowane wokół nich.



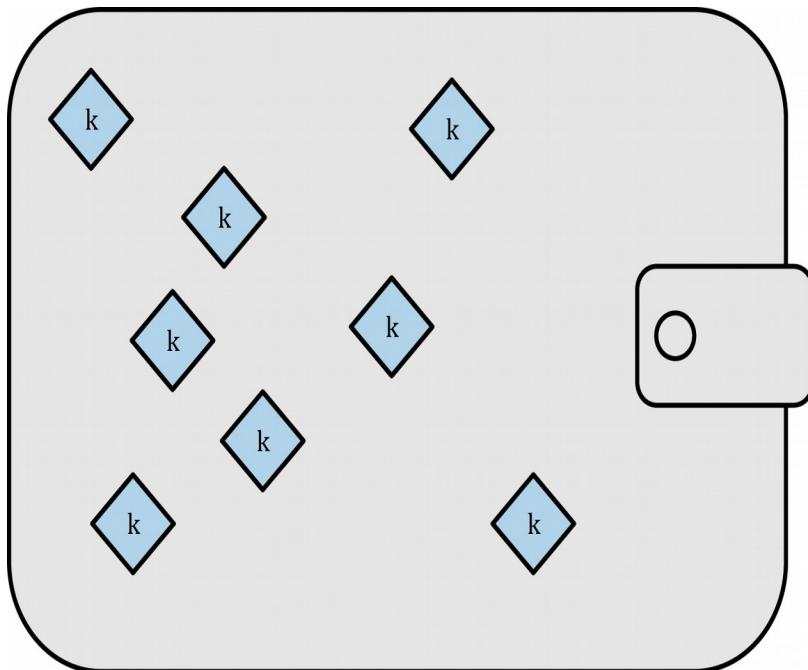
Portfele bitcoin zawierają klucze, a nie monety. Każdy użytkownik posiada portfel zawierający klucze. Portfele są tak naprawdę łańcuchami kluczy zawierające pary kluczy prywatnych/publicznych (patrz: „[Klucze prywatne i publiczne](#)”). Użytkownicy podpisują transakcję kluczami i przez to potwierdzają, że są posiadaczami bloków wyjściowych tej transakcji (monet). Monety są przechowywane w łańcuchach blokowych w postaci bloków wyjściowych transakcji (często zapisywanych jako vout lub txout).

Portfele niedeterministyczne (losowe)

W pierwszych klientach bitcoin, portfele stanowiły po prostu zbiór losowo wygenerowanych kluczy prywatnych. Ten rodzaj portfela jest nazywany *portfelem niedeterministycznym typu-0*. Na przykład, klient Bitcoin Core wstępnie generuje 100 losowych kluczy prywatnych podczas pierwszego uruchomienia, a następnie coraz więcej kluczy w miarę potrzeby wykorzystując każdy klucz tylko raz. Ten rodzaj portfela nazywany jest „Just a Bunch Of Keys” lub JBOK (piękiem kluczy) i tego typu portfele są zastępowane deterministycznymi portfelami, ponieważ są one niezbyt wygodne w zarządzaniu, backupowaniu oraz importowaniu. Wadą kluczy losowych jest to, że należy przechowywać kopie wszystkich wygenerowanych kluczy, co oznacza konieczność częstego wykonywania kopii zapasowych. Każdy klucz musi być backupowany, w przeciwnym razie fundusze, które kontroluje zostaną w sposób nieodwracalny utracone, jeżeli portfel stanie się niedostępny. To jest w bezpośrednim konflikcie z zasadą unikania ponownego wykorzystywania adresów wykorzystując każdy adres bitcoin w odniesieniu do tylko jednej transakcji. Ponowne wykorzystanie adresu powoduje zredukowanie prywatności poprzez skojarzenie wielu transakcji oraz adresów ze sobą. Portfel niedeterministyczny typu 0 to nienajlepszy wybór zwłaszcza, jeśli chcemy uniknąć ponownego wykorzystywania adresów, co oznacza, zarządzanie wieloma kluczami i konieczność częstego tworzenia kopii zapasowych. Chociaż klient Bitcoin Core zawiera portfel typu 0, programiści Bitcoin Core nie zachęcają do korzystania z niego. [Rys. 4-8](#) przedstawia portfel niedeterministyczny zawierający luźny zbiór kluczy losowych.

Portfele deterministyczne (źródłowe)

Portfele deterministyczne lub inaczej "źródłowe" to portfele zawierające klucze prywatne, które są generowane w oparciu o wspólne źródło wykorzystując jednokierunkową funkcję hash. Źródło jest to losowo generowana liczba, która jest łączona z innymi danymi takimi jak np. numer indeksu lub "kod łańcucha" (patrz „[Hierarchiczne portfele deterministyczne \(BIP0032/BIP0044\)](#)” w celu wygenerowania kluczy prywatnych. W portfelu deterministycznym, źródło jest wystarczającą przesłanką do odzyskania wszystkich wygenerowanych kluczy i dlatego pojedynczy backup w czasie utworzenia wystarczy. Źródło także wystarczy do importu i eksportu portfela, co umożliwia łatwą migrację wszystkich kluczy użytkownika pomiędzy różnymi implementacjami portfela.



Rys. 4-8. Portfele niedeterministyczny typu-0 (losowy): zbiór losowo wygenerowanych kluczy

Kody mnemotechniczne

Kody mnemotechniczne to sekwencje słów angielskich reprezentujących (zakodowane) liczby losowe wykorzystywane w celu wygenerowania portfela deterministycznego. Sekwencja słów jest wystarczająca, aby odtworzyć źródło i na tej podstawie odtworzyć portfel oraz wszystkie wygenerowane klucze. Aplikacja portfelowa, która wdraża portfele deterministyczne za pomocą kodów mnemotechnicznych przedstawi użytkownikowi sekwencję od 12 do 24 słów budując portfel po raz pierwszy. Ta sekwencja słów to jednocześnie kopia zapasowa, która może być wykorzystana do odzyskania i odtworzenia wszystkich kluczy w tej samej lub kompatybilnej aplikacji portfelowej. Słowa wykorzystywane w kodach mnemotechnicznych ułatwiają użytkownikom backupowanie portfeli, ponieważ są one łatwe do odczytania w porównaniu z losową sekwencją liczb oraz są poprawnie zapisane.

Kody mnemotechniczne są zdefiniowane w Bitcoin Improvement Proposal 39 (patrz: [\[bip0039\]](#)), obecnie w fazie wersji wstępnej. Należy podkreślić, że BIP0039 to wersja wstępna, a nie standard. Istnieje odmienny standard wykorzystujący inny zestaw słów wykorzystywany przez portfel Electrum, który jest wcześniejszy w

stosunku do BIP0039. BIP0039 jest wykorzystywany przez portfel Trezor oraz kilka innych portfeli, ale nie jest zgodny z oprogramowaniem Electrum.

BIP0039 definiuje tworzenie kodów mnemonicznych oraz źródeł zgodnie z poniższym:

1. Utwórz sekwencje losową (entropia) od 128 do 256 bitową.
2. Utwórz sumę kontrolną sekwencji losowej wykorzystując pierwszych kilka bitów wartości hash SHA256.
3. Dodaj sumę kontrolną na końcu sekwencji losowej.
4. Podziel sekwencję na części 11-bitowe, wykorzystując je w celu zindeksowania słownika 2048 predefiniowanych słów.
5. Wygeneruj 12 do 24 słów reprezentujących kod mnemoniczny.

Tabela 4-5 przedstawia relację pomiędzy rozmiarem danych losowych a długością kodów mnemonicznych w słowach.

Table 4-5. Kody mnemotechniczne: entropia i długość słów

Entropia (bity)	Suma kontrolna (bity)	Entropia+suma kontrolna	Długość słowa
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

Kod mnemotechniczny reprezentuje 128 do 256 bitów, które są wykorzystywane w celu wygenerowania dłuższych (512-bitowych) źródeł wykorzystując funkcję wydłużania kluczy PBKDF2. Uzyskane w ten sposób źródło jest następnie wykorzystywane w celu zbudowania portfela deterministycznego oraz wszystkich kluczy pochodnych.

Tabele **4-6** i **4-7** zawierają niektóre przykładowe kody mnemotechniczne oraz źródła, jakie generują.

Table 4-6. 128-bitowy kod entropii oraz źródło wygenerowane na jego podstawie

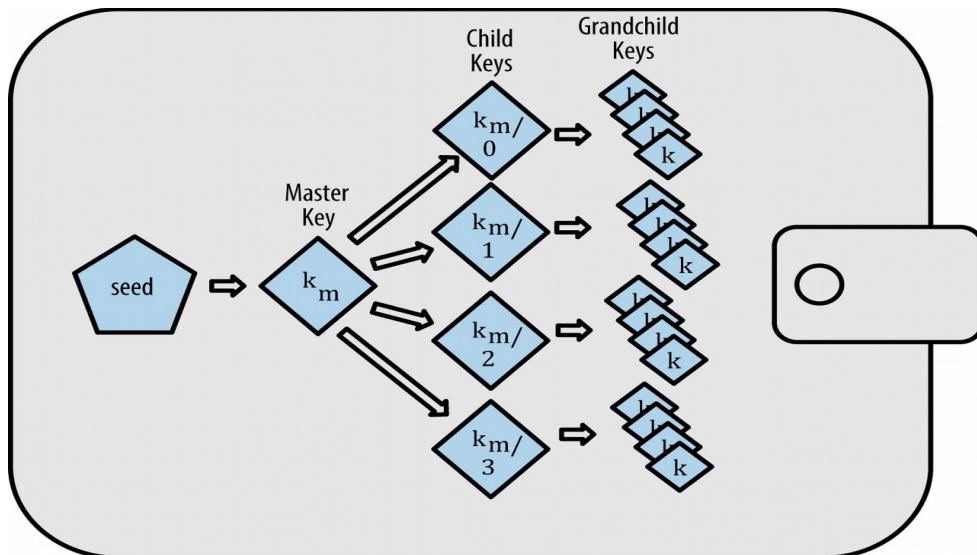
Dane wejściowe entropii (128 bitów)	0c1e24e5917779d297e14d45f14e1a1a
Mnemotechniczne (12 słów)	army van defense carry jealous true garbage claim echo media make crunch
Źródło (512 bitów)	3338a6d2ee71c7f28eb5b882159634cd46a898463e9d2d0980f8e80dfbba5b0fa0291e5fb88 8a599b44b93187be6ee3ab5fd3ead7dd646341b2cdb8d08d13bf7

Tabela 4-7. 256-bitowy kod entropii mnemotechnicznej or wynikowy kod źródłowy

Dane wejściowe entropii (256 bits)	2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c
Mnemotechniczne (24 słowa)	cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige
Źródło (512 bits)	3972e432e99040f75ebe13a660110c3e29d131a2c808c7ee5f1631d0a977fcf473bee22 fce540af281bf7cdeade0dd2c1c795bd02f1e4049e205a0158906c343

Portfele hierarchiczno-deterministyczne (BIP0032/BIP0044)

Deterministyczne portfele zostały opracowane w celu ułatwienia generowania wielu kluczy z pojedynczego źródła. Najbardziej zaawansowaną postacią portfeli deterministycznych jest *portfel hierarchiczno-deterministyczny* lub *portfel HD* zdefiniowany w standardzie BIP0032. Portfele hierarchiczno-deterministyczne zawierają klucze wygenerowane w strukturze drzewka w taki sposób, że klucz macierzysty może generować sekwencje dzieci – kluczy, z których każdy może generować sekwencje wnuków, itp. aż do nieskończoności. Ta struktura drzewka jest zilustrowana na Rys. 4-9.



Rys. 4-9. Portfel hierarchiczno-deterministyczny typu 2: drzewko kluczy wygenerowanych z klucza źródłowego



Implementując portfel bitcoin, należy go zbudować jako portfel HD przestrzegając standardów BIP0032 i BIP0044.

Portfele HD mają dwie ogromne zalety w porównaniu z kluczami losowymi (niedeterministycznymi). Po pierwsze, struktura drzewka może być wykorzystywana w celu nadania dodatkowego znaczenia organizacyjnego, tzn. określania gałęzi podkluczy w celu przyjęcia przychodzących płatności oraz innej w celu odbierania reszty po zrealizowaniu płatności wychodzących. Gałęzie kluczy mogą być także wykorzystywane w ustawieniach korporacyjnych w celu alokacji różnych gałęzi do departamentów, jednostek zależnych, specyficznych funkcji lub kategorii księgowych.

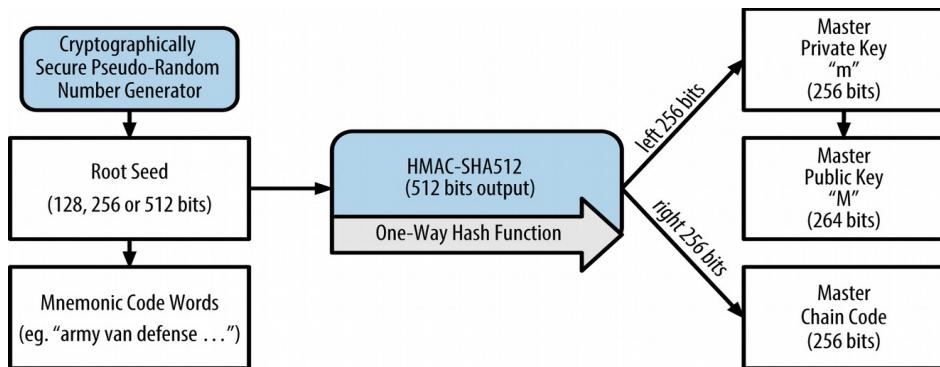
Drugą zaletą portfeli HD jest to, że użytkownicy mogą tworzyć sekwencje kluczy publicznych bez konieczności uzyskiwania dostępu do odpowiadających im kluczy prywatnych. To umożliwia portfelom HD wykorzystanie na niezabezpieczeniach serwerach lub tylko w charakterze odbiorcy i generowanie innego klucza publicznego dla każdej transakcji. Klucze publiczne nie muszą być wstępnie pobrane lub wygenerowane z góry; nie mniej jednak serwer nie ma kluczy prywatnych umożliwiających wydatkowanie środków na koncie.

Tworzenie portfela HD z kodu źródłowego

Portfele HD są tworzone z pojedynczego klucza źródłowego, który jest 128-, 256-, lub 512-bitową liczbą losową. Pozostałe elementy portfela HD są generowane w sposób deterministyczny w oparciu o ten klucz, co umożliwia odtworzenie całego portfela HD w oparciu o ten kod w każdym kompatybilnym portfelu HD. Ułatwia

to także backupowanie, odzyskiwanie, eksport oraz import portfeli HD zawierających tysiące, a nawet miliony kluczy przekazując jedynie klucz źródłowy. Kod źródłowy jest bardzo często reprezentowany przez mnemotechniczną sekwencję słów zgodnie z opisem w poprzedniej części „**Mnemotechniczne kody słowne**” w celu ułatwienia transkrypcji i zapisania.

Proces tworzenia kluczy głównych oraz głównego kodu łańcucha dla portfela HD jest przedstawiony na Rys. 4-10.



Rys. 4-10. Tworzenie kluczy głównych oraz kodów łańcucha w oparciu o klucz źródłowy

Klucz źródłowy jest wprowadzany do algorytmu HMAC-SHA512 a powstała w ten sposób wartość hash jest wykorzystywana do utworzenia **głównego klucza prywatnego** (*m*) oraz **głównego kodu łańcucha**. Główny prywatny klucz (*m*) następnie generuje odpowiadający mu klucz główny (*M*), wykorzystując normalny proces mnożenia krzywej eliptycznej $m * G$, z którym zapoznaliśmy się wcześniej w niniejszym rozdziale. Kod łańcucha jest wykorzystywany w celu wprowadzenia entropii w funkcji, która tworzy klucze – dzieci w oparciu o klucze rodzicielskie, o czym przekonamy się w kolejnej części.

Derywacja prywatnego klucza dziecka

Portfele hierarchiczno-deterministyczne wykorzystują funkcję *child key derivation* (CKD) w celu wygenerowania kluczy dzieci w oparciu o klucze rodzicielskie.

Funkcja derywacji kluczy pochodnych (child key derivation) są oparte o jednokierunkową funkcję hash, która łączy:

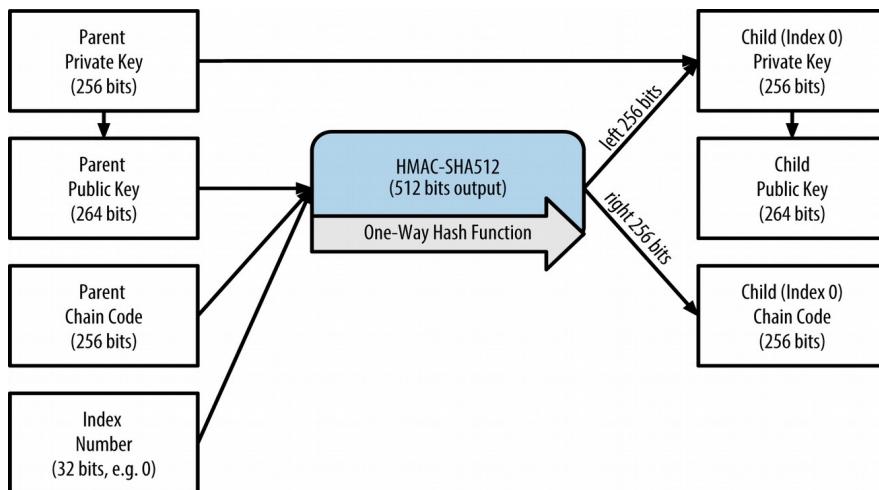
- Klucz prywatny lub publiczny (nieskompresowany klucz ECDSA)
- Klucz źródłowy nazywany kodem łańcucha (256 - bitowy)
- Numer indeksu (32-bitowy)

Kod łańcucha jest wykorzystywany w celu wprowadzenia pozornie losowych danych do procesu, w taki sposób, że indeks nie będzie wystarczający w celu wygenerowania innych kluczy – dzieci. Tym samym posiadanie klucza-dziecka nie umożliwia znalezienia rodzeństwa, chyba że posiadamy także kod łańcucha. Wstępny źródłowy kod łańcucha (u podstawy drzewa) jest generowany w oparciu o dane losowe, podczas, gdy kolejne kody łańcucha pochodzą z każdego kodu łańcucha rodzicielskiego.

Te trzy elementy są łączone i hashowane w celu wygenerowania kluczy dzieci zgodnie z poniższym.

Rodzicielski klucz publiczny, kod łańcucha oraz numer indeksu są łączone i hashowane za pomocą algorytmu HMAC-SHA512 w celu wygenerowania 512-bitowej wartości hash. Wynikowa wartość hash następnie dzielona na dwie równe części. Prawa część 256-bitowego bloku wynikowego hash staje się kodem łańcucha dziecka.

Lewa 256-bitowa strona wartości hash oraz numer indeksu są dodawane do prywatnego klucza rodzicielskiego generując klucz prywatny dziecka. Na [Rys. 4-11](#), widzimy ilustrację tego procesu z indeksem o wartości 0 generującym 0-we (pierwsze w indeksie) dziecko rodzica.



Rys. 4-11. Rozszerzanie rodzicielskiego klucza prywatnego w celu utworzenia prywatnych kluczy - dzieci

Zmiana indeksu umożliwia nam rozszerzenie klucza – rodzicielskiego i utworzenie innych dzieci w sekwencji, np. dziecko 0, dziecko 1, dziecko 2, etc. Każdy klucz rodzicielski może mieć 2 miliardy kluczy-dzieci.

Potworzenie procesu na niższym poziomie powoduje, że każde dziecko może stać się rodzicem i tworzyć swoje własne nieograniczone co do liczby pokolenia.

Wykorzystywanie pochodnych kluczy-dzieci

Prywatne klucze-dzieci są nieroróżnialne od niedeterministycznych (losowych) kluczy. Ponieważ funkcja derywacji jest funkcją jednokierunkową, klucz-dziecko nie może być wykorzystywany w celu znalezienia klucza - rodzice. Klucz-dziecko także nie może być wykorzystywany w celu znalezienia rodzeństwa. Jeżeli macie n dziecko, nie możecie znaleźć jego rodzeństwa n-1 lub n+1, ani żadnych innych dzieci, które są częścią sekwencji. Jedynie klucze rodzicielskie oraz kody łańcucha mogą generować wszystkie dzieci. Bez kodu łańcucha-dziecka, dziecko nie może być także wykorzystane w celu wygenerowania wnuka. Potrzebne są oba klucze prywatne – dziecko i rodzice, aby uruchomić nową gałąź i wygenerować wnuki.

Zatem do czego klucz prywatny dziecko może być wykorzystywany jako taki? Do generowania klucza publicznego oraz adresu bitcoin. Może być także wykorzystywany do podpisywania transakcji i wydatkowania środków zdeponowanych na danym koncie-adresie.



Klucz prywatny-dziecko, odpowiadający mu klucz publiczny oraz adres bitcoin można odróżnić od kluczy oraz adresów utworzonych losowo. Fakt, że są one częścią sekwencji nie jest widoczny poza funkcją portfela HD, która je utworzyła. Po utworzeniu, działają całkowicie jak "normalne" klucze.

Klucze rozszerzone

Jak przekonaliśmy się wcześniej funkcja derywacji kluczy może być wykorzystana w celu utworzenia dzieci na poziomie drzewka w oparciu od trzy typy strumieni wejściowych: klucz, kod łańcucha oraz indeks danego dziecka. Dwa podstawowe składniki to klucz oraz kod łańcucha, które łącznie nazywane są *kluczem rozszerzonym*. Termin “klucz rozszerzony” może być także uznany za “klucz rozszerzalny”, ponieważ klucz ten może być wykorzystywany do generowania pochodnych kluczy-dzieci.

Klucze rozszerzone są przechowywane i reprezentowane po prostu jako dodatek 256-bitowego klucza oraz 256-bitowego kodu łańcucha do sekwencji 512-bitowej. Istnieją dwa typy kluczy rozszerzonych. Rozszerzony klucz prywatny stanowi kombinację klucza prywatnego oraz kodu łańcucha, który może być wykorzystany w celu wygenerowania prywatnych kluczy-dzieci (a z nich następnie publicznych kluczy-dzieci). Rozszerzony klucz publiczny to klucz publiczny oraz kod łańcucha, który można wykorzystać w celu utworzenia publicznego klucza-dziecka zgodnie z opisem w części **“Generowanie klucza publicznego”**.

Wyobraźcie sobie klucz rozszerzony jako podstawę gałęzi w strukturze drzewa portfela HD. Mając tę podstawę, można wyderywować pozostałe elementy-gałęzie drzewa. Rozszerzony klucz może następnie tworzyć kompletne gałęzie, a rozszerzony klucz publiczny może tworzyć wyłącznie gałęzie kluczy publicznych.



Klucz rozszerzony składa się z klucza prywatnego lub publicznego oraz kodu łańcucha. Klucz rozszerzony może tworzyć dzieci i generować swoje własne gałęzie w strukturze drzewa. Klucz rozszerzony daje dostęp do całej gałęzi.

Klucze rozszerzone są szyfrowane w systemie Base58Check, aby ułatwić eksport oraz import pomiędzy różnymi portfelami kompatybilnymi z BIP0032. Szyfrowanie kluczy rozszerzonych w systemie Base58Check jest w oparciu o specjalny numer wersji, który stanowi prefiks “xprv” i “xpub” w przypadku szyfrowania znakami Base58, aby można było łatwo rozpoznać. Ponieważ klucz rozszerzony jest 512 lub 513 bitowy jest on także znacznie dłuższy niż pozostałe strumienie danych szyfrowane za pomocą Base58Check, które omówiliśmy wcześniej.

Poniżej mamy przykład rozszerzonego klucza prywatnego szyfrowanego w Base58Check:

```
xprv9tyUQV64JT5qs3RSTJkXCWKMMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6GoNMKUga5biW6H  
x4twS2six3b9c
```

W tym przykładzie widzimy rozszerzony klucz publiczny szyfrowany za pomocą Base58Check:

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2ttypeQbBs2UAR6KECeeMVKZBPLrtJunSDMstweylXhRgPxd  
p14sk9tJPW9
```

Derywacja kluczy publicznych

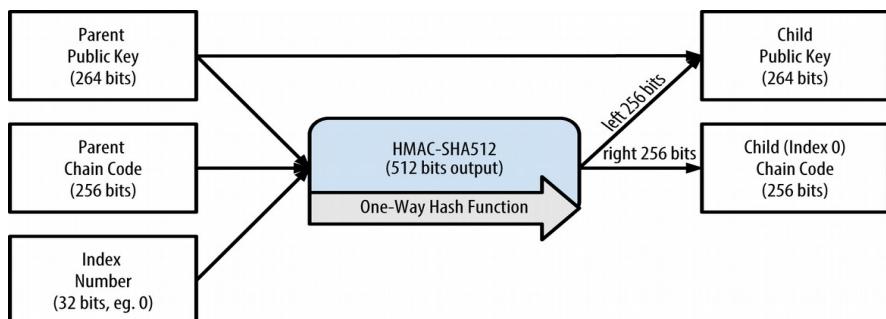
Jak już wspomnieliśmy, użyteczną cechą portfeli hierarchiczno-deterministycznych jest zdolność do derywacji kluczy publicznych-dzieci w oparciu o klucze-rodziców *bez konieczności* posiadania klucza prywatnego. To umożliwia nam derywację publicznych kluczy -dzieci na dwa sposoby: albo na podstawie prywatnego klucza-dziecka lub bezpośrednio z publicznego klucza-dziecka.

Rozszerzony klucz publiczny może być w związku z tym wykorzystany w celu wygenerowania wszystkich kluczy publicznych (i tylko kluczy publicznych) w danej gałęzi struktury portfela HD.

Ten skrót można wykorzystywać w celu utworzenia bardzo bezpiecznej struktury wyłącznie kluczy publicznych, gdzie na serwerze lub w aplikacji przechowywane są kopie rozszerzonych kluczy publicznych bez kluczy prywatnych. Tego typu rozwiązanie może wygenerować nieskończoną liczbę kluczy publicznych oraz adresów bitcoin, ale nie umożliwia wydatkowania środków zdeponowanych pod tymi adresami. Jednocześnie na innym serwerze z lepszym zabezpieczeniem, rozszerzony klucz prywatny może wygenerować wszystkie odpowiadające mu klucze prywatne niezbędne do podpisania transakcji oraz korzystania ze środków.

Jednym z zastosowań tego rozwiązania jest instalacja rozszerzonego klucza publicznego na serwerze webowym, który obsługuje aplikację typu ecommerce. Serwer ten może wykorzystywać funkcje derywacji klucza publicznego w celu utworzenia nowych adresów bitcoin dla każdej transakcji (np. dla wózka z zakupami klienta). Serwer webowy nie będzie posiadać żadnych kluczy prywatnych, które byłyby podatne na kradzież. Bez portfela HD, jedynym sposobem jest wygenerowanie tysięcy adresów bitcoin na oddzielnym zabezpieczonym serwerze na następne wstępne ich wprowadzenie na serwer ecommerce. Podejście to jest niewygodne i wymaga stałej opieki i kontroli, aby uniknąć sytuacji, w której zabraknie kluczy na serwerze ecommerce.

Kolejnym popularnym zastosowaniem tego rozwiązania jest cold-storage oraz portfele sprzętu. W tym scenariuszu, rozszerzone klucze prywatne można przechowywać w portfelu papierowym lub na urządzeniu (np. w portfelu sprzętowym Trezor), podczas gdy rozszerzony klucz publiczny może być przechowywany online. Użytkownik może tworzyć - "odbierać" -dowolne adresy, podczas gdy klucze prywatne są bezpiecznie przechowywane w trybie offline. Aby dysponować środkami, użytkownik może wykorzystywać klucze prywatne znajdujące się kliencie podpisów bitcoin w trybie offline lub podpisywać transakcje na urządzeniu portfelowym (np. Trezor). **Rys. 4-12** ilustruje ten mechanizm rozszerzania nadzędnych kluczy publicznych w celu wyderywowania publicznych kluczy pochodnych - dzieci.

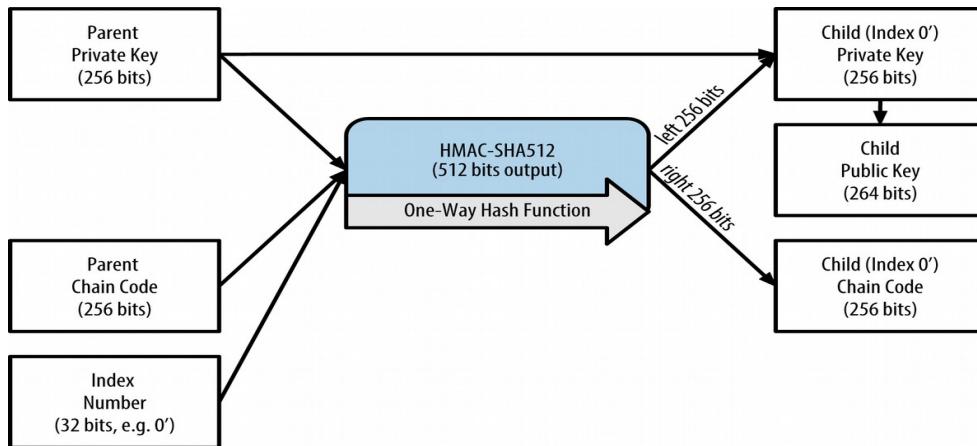


Rys. 4-12. Rozszerzenie klucza publicznego w celu utworzenia publicznego klucza pochodnego

Derywacja kluczy -dzieci z dodatkowym zabezpieczeniem

Zdolność do derywacji gałęzi kluczy publicznych z rozszerzonego klucza publicznego jest niezwykle użyteczna, ale wiąże się z potencjalnym ryzykiem. Dostęp do rozszerzonego klucza publicznego nie oznacza dostępu do prywatnych kluczy - dzieci. Nie mniej jednak, ponieważ rozszerzony klucz publiczny zawiera kod łańcucha znając prywatny klucz-dziecko lub wykorzystując wyciek danych, może on być wykorzystany wraz z kodem łańcucha w celu wyderywowania wszystkich pozostałych prywatnych kluczy dzieci. Pojedynczy wyciek klucza prywatnego wraz z kodem łańcucha nadzędnego (rodzica) ujawnia wszystkie pochodne klucze prywatne. Co gorsza pochodzący klucz prywatny wraz z nadzędnym (rodzicielskim) kodem łańcucha może być wykorzystywany w celu wywnioskowania rodzicielskiego klucza prywatnego.

Aby przeciwdziałać temu ryzyku, portfele HD wykorzystują alternatywne funkcje derywacji nazywane *derywacją z dodatkowym zabezpieczeniem*, które „zrywają” relacje pomiędzy nadzędnym kluczem publicznym, a kodem łańcucha kluczów pochodnych. Funkcja derywacji z zabezpieczeniem wykorzystuje nadzędny klucz prywatny w celu wyderywowania kodu łańcucha-dziecka zamiast publicznego klucza-dziecka. To rozwiązanie tworzy “firewall” w sekwencji rodzic/dziecko z kodem łańcucha, który nie może być wykorzystany w celu złamania klucza prywatnego nadzędniego lub siostrzanego. Funkcja derywacji z zabezpieczeniem wygląda niemal identycznie, normalna funkcja derywacji klucza prywatnego z tym, że prywatny klucz nadzędny jest wykorzystywany jako strumień wejściowy w funkcji hashującej zamiast nadzędnego klucza publicznego zgodnie z wykresem na Rys. 4-13.



Rys. 4-13. Derywacja klucza-dziecka z zabezpieczeniem; z pominięciem rodzicielskiego klucza publicznego

Wykorzystując funkcje derywacji z zabezpieczeniem, wynikowy prywatny klucz-dziecko oraz kod łańcucha są całkowicie odmienne od tego, co uzyskujemy w trakcie normalnej derywacji. Uzyskane w ten sposób „gałęzie” kluczy można wykorzystywać w celu wygenerowania rozszerzonych kluczy publicznych, które są odporne, ponieważ kod łańcucha, który zawierają nie może być wykorzystany w celu ujawnienia żadnego klucza prywatnego. Derywacja z zabezpieczeniem jest, dlatego wykorzystywane w celu utworzenia “luki” w drzewku powyżej poziomu, na którym wykorzystywane są rozszerzone klucze publiczne.

Mówiąc prościej, jeżeli chcemy wykorzystać wygodę oferowaną przez rozszerzony klucz publiczny, w celu wygenerowania gałęzi kluczy publicznych bez narażania się na ryzyko wycieku kodu łańcucha, należy go wyderywować w oparciu o zabezpieczonego rodzica, a nie zwykły klucz nadzędny. Najbezpieczniejszą praktyką jest generowaniem dzieci na poziomie 1 z kuczy głównych w procesie derywacji z zabezpieczeniem, aby uniemożliwić ich złamanie.

Liczba indeksu w derywacji normalnej i z zabezpieczeniem

Liczba indeksów wykorzystywanych w funkcji derywacji to 32-bitowa liczba całkowita. Aby ułatwić rozróżnianie pomiędzy kluczami wygenerowanymi przy użyciu normalnej funkcji derywacji a kluczami wygenerowanymi przy wykorzystaniu derywacji z zabezpieczeniem, liczba indeksu jest podzielona na dwa przedziały. Indeksy od 0 do $2^{31}-1$ (0x0 do 0xFFFFFFFF) są wykorzystywane wyłącznie w procesie normalnej derywacji. Liczby indeksu od 2^{31} do $2^{32}-1$ (0x80000000 do 0xFFFFFFFF) są wykorzystywane wyłącznie w procesie derywacji z zabezpieczeniem. Dlatego, jeżeli liczba indeksu jest mniejsza niż 2^{31} , oznacza to, że dziecko jest normalne; jeżeli liczba indeksu jest równa lub większa niż 2^{31} dziecko zostało wygenerowane z zabezpieczeniem.

Aby ułatwić odczyt i wyświetlanie, liczby indeksu dla kluczy-dzieci z zabezpieczeniem są wyświetlane od zera, ale z symbolem liczby pierwszej. Pierwsze normalne dziecko jest przedstawiane jako 0, a pierwsze dziecko z zabezpieczeniem (index 0x80000000) jako 0'. Zatem w sekwencji, drugi klucz z zabezpieczeniem będzie mieć indeks 0x80000001 i będzie wyświetlany jako 1', itd. Kiedy zobaczymy indeks portfela HD i', oznacza to wartości 2^{31+i} .

Identyfikator klucza portfela HD (ścieżka)

Klucze w portfelu są identyfikowane za pomocą konwencji wykorzystywanej do nazw "ścieżek"; każdy poziom drzewka jest oddzielany ukośnikiem (/) (patrz: **Tabela 4-8**). Klucze prywatne pochodzące od głównego klucza prywatnego zaczynają się literą "m". Klucze publiczne pochodzące od głównego klucza publicznego zaczynają się od "M". Dlatego pierwszy prywatny klucz-dziecko głównego klucza prywatnego oznaczone jest m/0. Pierwszy pochodny klucz publiczny oznaczone jest M/0. Drugi wnuk pierwszego dziecka to m/0/1, itd.

"Przodków" kluczy odczytujemy od prawej do lewej, aż do klucza głównego, od którego wszystkie pozostałe pochodzą. Na przykład: identyfikator m/x/y/z opisuje klucz, który jest z-tym dzieckiem klucza m/x/y, który jest y-tym dzieckiem klucza m/x, który jest x-tym dzieckiem m.

Tabela 4-8. Przykład ścieżki portfela HD

Ścieżka HD	Opis klucza
m/0	Pierwszy prywatny klucz-dziecko (0) wygenerowany z głównego klucza prywatnego(m)
m/0/0	Pierwszy prywatny klucz-wnuk pierwszego dziecka (m/0)
m/0'/0	Pierwszy normalny wnuk pierwszego dziecka wygenerowanego z zabezpieczeniem (m/0')
m/1/0	Pierwszy prywatny klucz wnuk drugiego dziecka (m/1)
M/23/17/0/0	Pierwszy pra-prawnuk klucza publicznego pierwszego prawnuka 18-tego wnuka 24-go dziecka

Poruszanie się po strukturze drzewka portfela HD

Struktura drzewka portfela HD oferuje ogromną elastyczność. Każdy rodzicielski klucz rozszerzony może mieć do 4 miliardów dzieci: 2 miliardy dzieci normalnych oraz 2 miliardy dzieci z zabezpieczeniem. Każde z tych dzieci może mieć kolejne 4 miliardy potomków itd. Drzewko może mieć dowolną głębokość i nieskończoną liczbę generacji. Elastyczność ta jednak sprawia, że poruszanie się po tym nieskończonym drzewku jest dość trudne. Szczególne problemy nastręcza transfer portfeli HD pomiędzy programami, ponieważ możliwości wewnętrznej organizacji i podziału na gałęzie lub podgałęzie są nieskończone.

Dwa dokumenty Bitcoin Improvement Proposals (BIP) zawierają rozwiązanie dla tego złożonego problemu poprzez opracowanie standardów dla struktur drzewek portfeli HD. BIP0043 proponuje wykorzystanie pierwszych indeksów kluczy-dzieci z zabezpieczeniem, jako specjalnych identyfikatorów zawierających informacje o "celu", utworzenia struktury drzewka. Na podstawie BIP0043, portfel HD powinien wykorzystywać wyłącznie jeden poziom 1 gałęzi drzewka z liczbą indeksu określającą strukturę oraz nazwy pozostałych części drzewka poprzez zdefiniowanie celu. Na przykład: portfel HD wykorzystujący jedynie gałąź m/i/ ma na celu oznaczenie określonego celu i ten jest oznaczony w indeksie liczbą „i”.

Rozszerzając tę specyfikację, BIP0044 proponuje utworzenie struktury wielu kont jako liczby "celu" 44' w ramach BIP0043. Wszystkie portfele stosujące strukturę BIP0044 są identyfikowane na podstawie faktu wykorzystania jednej gałęzi drzewa: m/44'/. BIP0044 strukturę jako składającą się z pięciu predefiniowanych poziomów:

m / purpose' / coin_type' / account' / change / address_index

“Cel” pierwszego poziomu jest zawsze ustawiony jako 44'. Drugi poziom to “coin_type”, który określa rodzaj monet wykorzystywanych w kryptowalutach umożliwiając tworzenie wielowalutowych portfeli HD, gdzie każda waluta ma swoje własne pod-drzewko na drugim poziomie. Obecnie zdefiniowano trzy waluty: Bitcoin to m/44'/0', Bitcoin Testnet to m/44'/1', a Litecoin to m/44'/2'.

Trzeci poziom drzewka to “konto,” które umożliwia użytkownikom podzielenie swoich portfeli na subkonta logiczne w celu księgowania i odpowiedniej organizacji. Na przykład, portfel HD może zawierać dwa “konta” bitcoinowe: m/44'/0'/0' oraz m/44'/0'/1'. Każde z nich stanowi podstawę dla pochodnych struktur drzewek.

Na czwartym poziomie, “reszty,” portfel HD posiada dwie struktury pochodne – jedną w celu tworzenia adresów odbiorców oraz drugą dla tworzenia adresów reszty. Należy podkreślić, że ponieważ wcześniejsze poziomy wykorzystywały derywacje z zabezpieczeniem, ten poziom wykorzystuje derywacje standardową. Rozwiążane to ma na celu umożliwienie eksportowania z tego poziomu rozszerzonych kluczy publicznych, które będą wykorzystywane w niezabezpieczonym środowisku. Adresy są generowane przez portfel HD jako dzieci czwartego poziomu i sprawiając, że piąty poziom drzewka będzie zawierać “address_index.” Na przykład: trzeci adres odbiorczy płatności bitcoin na głównym koncie będzie oznaczony jako M/44'/0'/0'/0/2. **Tabela 4-9** zawiera więcej przykładów.

Tabela 4-9. Przykładowa struktura portfela BIP0044 HD

Ścieżka HD	Opis klucza
M/44'/0'/0'/0/2	Trzeci klucz publiczny odbiorcy dla głównego konta bitcoin
M/44'/0'/3'/1/14	Piętnasty klucz publiczny adresu reszty dla czwartego konta bitcoin
m/44'/2'/0'/0/1	Drugi prywatny klucz na koncie głównym Litecoin do podpisywania transakcji

Eksperymenty na portfelach HD przy wykorzystaniu narzędzi sx

Wykorzystując wiersze poleceń narzędzi sx, omówionych w **Rozdziale 3**, można przeprowadzać eksperymenty z generowaniem i rozszerzaniem kluczy deterministycznych BIP0032 oraz wyświetlać je w różnych formatach:

```
$ sx hd-seed > m # create a new master private key from a seed and store in
file "m"
$ cat m # show the master extended private key
xprv9s21ZrQH143K38iQ9Y5p6qoB8C75TE71NfpvQPdfGvzghDt39DHPFpovvtWZaRgY5uPwV7RpEgHs7cvdgfiSjLjbuKGcjRyU7
RGSSS8Xa
$ cat m | sx hd-pub 0 # generate the M/0 extended public key
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtJ2ttypeQbBs2UAR6KEceeMVKZBPLrtJunSDMstweyLxhRgPxd
p14sk9tJPW9
$ cat m | sx hd-prv 0 # generate the m/0 extended private key
xprv9tyUQV64JT5qs3RSTJkXCWKMMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6GoNMKUga5biW6H
x4tw52six3b9c
$ cat m | sx hd-prv 0 | sx hd-to-wif # show the private key of m/0 as a WIF
L1pbvV86crAGoDzqmgY85xURkz3c435Z9nirMt52UbnGjYMzKBUN
$ cat m | sx hd-pub 0 | sx hd-to-address # show the bitcoin address of M/0
1CHCnCjgMNb6digimckNQ6TBVcTWBAmPHK
$ cat m | sx hd-prv 0 | sx hd-prv 12 --hard | sx hd-prv 4 # generate m/
0/12'/4
xprv9yL8ndfdPVeDWJenF18oiHguRUj8jHmVrqD97YQHeTcR3LCeh53q5PXPkLsy2kRaqgwoS6YZBLatRZRyUeAkRPe1kLR1P6Mn
7jUrxFquUt
```

Zaawansowane klucze i adresy

W kolejnych rozdziałach omówimy zaawansowane postaci kluczy i adresów, np. szyfrowane klucze prywatne, skrypty i adresy z wieloma podpisami, adresy vanity oraz portfele papierowe.

Szyfrowane klucze prywatne (BIP0038)

Prywatne klucze muszą pozostać kluczami sekretnymi. Obowiązek zachowania poufności kluczy prywatnych to truizm, który jest niezmiernie trudny do osiągnięcia w praktyce, ponieważ stoi on w sprzeczności z równie ważnym celem, jakim jest *dostępność*. Utrzymanie kluczy prywatnych jest o wiele bardziej trudne, kiedy trzeba zapisać kopie zapasowe, aby uniknąć utraty danych. Klucz prywatny przechowywany w portfelu jest szyfrowany za pomocą hasła i może być zabezpieczony, ale cały portfel należy backupować. Czasami użytkownicy muszą usuwać klucze z jednego portfela i przenosić do innego—aby podnieść wersję lub zmienić oprogramowanie portfela. Kopie zapasowe kluczy prywatnych można także przechowywać w postaci wydruku (patrz: "[Portfele papierowe](#)") lub na zewnętrznych nośnikach takich jak np. USB. Ale co się dzieje w przypadku kradzieży lub zagubienia kopii zapasowych? Te sprzeczne cele związane z zabezpieczeniami doprowadziły do wprowadzenia przenośnych i bardzo dogodnych standardów szyfrowania kluczy prywatnych w sposób, który będzie zrozumiały dla wielu różnych portfeli oraz klientów bitcoin ustandaryzowanych w oparciu o Bitcoin Improvement Proposal 38 lub BIP0038 (patrz: [\[bip0038\]](#)).

BIP0038 proponuje wspólny standard szyfrowania kluczy prywatnych oparty na haśle i kodowaniu za pomocą Base58Check, aby zapewnić ich bezpieczne przechowywanie na nośnikach kopii zapasowych, bezpieczną wymianę pomiędzy portfelami i przechowywanie w warunkach, w których poufność mogłaby zostać naruszona. Standardowe szyfrowanie odbywa się zgodnie z Advanced Encryption Standard (AES) – standardem opracowanym przez National Institute of Standards and Technology (NIST), który jest powszechnie stosowany w programach szyfrujących wykorzystywanych komercyjnie lub dla celów wojskowych.

System szyfrujący A BIP0038 wykorzystuje klucz prywatny bitcoin, który jest zwykle szyfrowany w formacie Wallet Import Format (WIF), jako strumień Base58Check z prefiksem "5". Dodatkowo, system szyfrujący BIP0038 wykorzystuje hasło—długi ciąg—zwykle składający się z kilkunastu słów lub złożonego strumienia znaków alfanumerycznych. W efekcie, powstaje klucz prywatny szyfrowany w Base58Check, który zaczyna się od prefiksu 6P. Każdy klucz zaczynający się od 6P jest szyfrowany i wymaga podania hasła w celu konwersji (deszyfrowania) na klucz prywatny w formacie WIF (prefiks 5), który następnie może być wykorzystywany w dowolnym portfelu. Wiele aplikacji portfelowych obecnie rozpoznaje szyfrowane w BIP0038 klucze prywatne i wysyła użytkownikowi komunikat o konieczności podania hasła w celu odszyfrowania lub zimportowania klucza. Aplikacje stron trzecich takie jak niewiarygodnie użyteczna przeglądarka [Bit Address](#) (zakładka szczegółów portfela) może być także wykorzystywane do deszyfrowania kluczy BIP0038.

Najbardziej powszechnym przypadkiem wykorzystania kluczy szyfrowanych w BIP0038 są portfele papierowe, które można wykorzystywać w celu tworzenia kopii zapasowych na wydruku (papierze). Jeżeli tylko użytkownik dobrze mocne hasło, portfel papierowy z kluczami prywatnymi szyfrowanymi w BIP0038 jest bardzo bezpieczny i stanowi doskonały sposób przechowywania danych bitcoin w trybie offline (znanych pod nazwą "cold storage").

Można przetestować szyfrowane klucze z [Tabeli 4-10](#) korzystając z [bitaddress.org](#), aby sprawdzić w jaki sposób odzyskiwać odszyfrowane klucze wpisując hasło.

Tabela 4-10. Przykład klucza prywatnego szyfrowanego w BIP0038

Klucz prywatny (WIF)	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Hasło	MyTestPassphrase
Klucz szyfrowany Key (BIP0038)	6PRTHL6mWa48xSopbU1cKrVjpKbBZxcLRRCDctLJ3z5yxE87MobKoXdTsJ

Funkcja hashująca Pay-to-Script (P2SH) i adresy Multi-Sig

Jak wiadomo, tradycyjne konwencjonalne adresy bitcoin zaczynają się od cyfry "1" i są generowane jako pochodne klucza publicznego, który jest derywatem klucza prywatnego. Chociaż każdy może wysłać bitcoin na adres "1", to te bitcoiny mogą być wydane jedynie po przedstawieniu odpowiedniego podpisu klucza prywatnego oraz funkcji hashującej klucza publicznego.

Adresy bitcoin zaczynające się od cyfry "3" to adresy funkcji hashujących pay-to-script (P2SH) niekiedy mylnie nazywane adresami multisignature lub adresami multi-sig. Określają one beneficenta transakcji bitcoin jako wartość hash skryptu zamiast właściciela klucza publicznego. Cecha ta została wprowadzona w styczniu 2012 roku w ramach Bitcoin Improvement Proposal 16 lub BIP0016 (patrz: [\[bip0016\]](#)) i jest obecnie powszechnie stosowana, ponieważ umożliwia dodawanie funkcjonalności do adresu. W odróżnieniu od transakcji, które wysyłają środki na tradycyjne adres bitcoin "1", znane także pod nazwą pay-topublic-key-hash (P2PKH), środki przesyłane na adres "3" wymagają czegoś więcej niż przedstawienia jednej funkcji hashującej klucza publicznego oraz podpisu kluczem prywatnym w celu poświadczania własności. Wymogi są określone w chwili tworzenia adresu w skrypcie i wszystkie bloki wejściowe docierające pod ten adres będą ograniczone tymi samymi wymogami.

Adres funkcji hashującej pay-to-script jest tworzony w oparciu o skrypt transakcji, który definiuje kto może dysponować blokiem wyjściowym transakcji (w celu uzyskania bliższych informacji, patrz: ["Pay-to-Script-Hash \(P2SH\)"](#)). Szyfrowanie adresów funkcją hashującą pay-to-script obejmuje także wykorzystanie tej samej podwójnej funkcji hashującej, która jest wykorzystywana do tworzenia adresów bitcoin, z ta różnicą, że jest stosowana w skrypcie zamiast w kluczu publicznym:

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

Utworzona w ten sposób "funkcja hashująca skryptu" jest szyfrowana za pomocą Base58Check w wersji z przedrostkiem 5, który pozwala stworzyć szyfrowany adres zaczynający się od cyfry 3. Przykładowy adres P2SH może wyglądać tak: 32M8ednmuyZ2zVbes4puqe44NZumgG92sM.



P2SH jest niekoniecznie taki sam jak standardowa transakcja typu multi-signature. Najczęściej, adres P2SH reprezentuje skrypt typu multi-signature, ale może także reprezentować skrypt szyfrujący inne typy transakcji.

Adresy multi-signature i P2SH

Obecnie najczęściej spotykana odmianą funkcji P2SH jest skrypt adresów multisignature. Jak sama nazwa wskazuje, skrypt bazowy wymaga więcej niż jednego podpisu w celu potwierdzenia własności i dysponowania środkami. Cecha multisignature system bitcoin została skonstruowana w taki sposób, aby wymagać M podpisów (znanych także jako "progi")

Z ogólnej liczby N kluczów, nazywane także M-z-N multi-sig, gdzie M jest równe lub mniejsze niż N. Na przykład, Bob – właściciel kawiarni z [Rozdziału 1](#), może wykorzystywać adres multi-signature wymagający podania 1-z-2 podpisów z klucza należącego do niego oraz klucza należącego do jego małżonki zapewniając im obojętną możliwość autoryzowania transakcji wydatkowania w bloku wyjściowym zablokowanym pod tym adresem. To może przypominać rachunek wspólny założony w tradycyjnej formie w banku, gdzie każde z małżonków może składać dyspozycje składając jeden podpis. Gopesh, projektant sieci, któremu Bob zapłacił za zbudowanie strony może posiadać 2-z-3 adresów multi-signature dla swojej działalności, które gwarantują, że żadne środki

nie zostaną wydatkowane, chyba, że co najmniej dwóch partnerów biznesowych autoryzuje (podpisze) transakcję.

Z metodami tworzenia transakcji, które powodują wysłanie środków z adresów P2SH (i multisignature) zapoznamy się w [Rozdziale 5](#).

Adresy vanity

Adresy vanity to ważne adresy bitcoin zawierające wiadomości, które mogą być odczytane przez użytkownika. Na przykład 1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33 to ważny adres zawierający litery, które tworzą słowo "Love" jako pierwsze cztery litery Base-58. Adresy vanity wymagają wygenerowania i przetestowania miliardów kluczy prywatnych – kandydatów do chwili znalezienia adresu bitcoin o wymaganym wzorze. Chociaż istnieje możliwość optymalizacji algorytmu generowania adresów vanity, zasadniczo, proces ten polega na wyszukiwaniu kluczy prywatnych losowo w oparciu o klucze publiczne wygenerowane na podstawie adresów bitcoin i sprawdzenie, czy pasują do wymaganego wzoru vanity – proces ten powtarza się miliardy razy aż do znalezienia pasującego adresu.

Po znalezieniu adresu vanity pasującego do wzorca, klucz prywatny z którego został wyderywowany może być wykorzystywany przez właściciela w celu dysponowania bitcoinami dokładnie w takim sam sposób jak w przypadku pozostałych adresów. Adresy vanity nie są ani mniej, ani bardziej bezpieczne niż inne adresy. Zależne są od szyfrowania w oparciu od tą samą krzywą eliptyczną (Elliptic Curve Cryptography - ECC) oraz algorytm zabezpieczające funkcji has (Secure Hash Algorithm - SHA) jak pozostałe adresy. Znalezienie klucza prywatnego adresu zaczynającego się od wzoru vanity wcale nie jest łatwiejsze niż w przypadku pozostałych typów adresów.

W [Rozdziale 1](#), przedstawiliśmy Eugenię, dyrektorkę organizacji charytatywnej na rzecz dzieci z Filipin. Założmy, że Eugenia organizuje zbiórkę funduszy bitcoin i chce wykorzystać adres vanity bitcoin w celu upowszechnienia informacji o tej akcji. Eugenia musi stworzyć adres vanity zaczynający się od "1Kids", aby przekazać informację o zbiórce funduszy na rzecz dzieci. Zobaczmy, jak ten adres vanity utworzyć oraz co on oznacza dla bezpieczeństwa organizacji charytatywnej prowadzonej przez Eugenię.

Generowanie adresów vanity

Należy sobie uświadomić, że adres bitcoin to po prostu liczba reprezentowana przez symbole alfabetu Base58. Wyszukiwanie wzoru "1Kids" można interpretować jako poszukiwanie adresu w przedziale od 1Kids11111111111111111111111111111111 do 1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzzz. Istnieje ok. 58^{29} (w przybliżeniu $1.4 \cdot 10^{51}$) adresów w tym przedziale zaczynających się od "1Kids". [Tabela 4-11](#) przedstawia przedział adresów z prefiksem 1Kids.

Tabela 4-11. Przedział adresów vanity zaczynających się od "1Kids"

Od	1Kids11111111111111111111111111111111
Do	1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzzz

Przyjrzymy się wzorowi "1Kids" jako liczbie i zobaczymy jak często możemy napotkać ten wzór w adresach bitcoin (patrz: [Tabela 4-12](#)). Średniej klasy komputer PC, bez specjalistycznego oprzyrządowania może wyszukać ok. 100 000 w ciągu sekundy.

Tabela 4-12. Częstotliwość powtarzania się wzoru vanity (1KidsCharity) oraz średni czas wyszukania (znalezienia) na komputerze PC

Długość	Wzór	Częstotliwość	Średni czas wyszukiwania
1	1K	1 na 58 kluczy	< 1 milisekunda
2	1Ki	1 na 3 364	50 milisekund
3	1Kid	1 na 195 000	< 2 sekundy
4	1Kids	1 na 11 milionów	1 minuta
5	1KidsC	1 na 656 milionów	1 h
6	1KidsCh	1 na 38 miliardów	2 dni
7	1KidsCha	1 na 2,2 tryliona	3-4 miesiące
8	1KidsChar	1 na 128 trylionów	13-18 lat
9	1KidsChari	1 na 7 kwadrylionów	800 lat
10	1KidsCharit	1 na 400 kwadrylionów	46 000 lat
11	1KidsCharity	1 na 23 kwintyliony	2,5 miliona lat

Jak widać Eugenia nie będzie tworzyć adresu vanity “1KidsCharity” wkrótce, nawet gdyby miała dostęp do kilkunastu tysięcy komputerów. Każdy tradycyjny znak podnosi stopień trudności o czynnik 58. Wzory i więcej niż siedmiu znakach są z reguły znajdowane przez specjalistyczne urządzenia takie jak komputery budowane na zamówienie z wieloma GPU. Są to bardzo często przebudowane “urządzenie” miningowe bitcoin, które już nie przynosi zysków z górnictwa bitcoin, ale mogą być wykorzystywane w celu znajdowania adresów vanity. Wyszukiwanie adresów vanity przez systemy GPU to obsługują więcej adresów i robią to szybciej w porównaniu z CPU ogólnego przeznaczenia.

Innym sposobem znajdowania adresów jest outsourcing usług z puli górników adresów vanity - **Vanity Pool**. Pula to usługa umożliwiająca górnikom posiadającym GPU zarabianie bitcoinów dzięki wyszukiwaniu adresów vanity dla innych użytkowników. Za niedużą opłatą (0,01

Bitcoina lub ok. \$5 w chwili opracowywania niniejszego materiału), Eugenia może wykupić usługę wyszukiwania siedmioznakowego wzoru adresu vanity i zdobyć wyniki w ciągu kilku godzin zamiast miesiącami wyszukiwać za pomocą CPU.

Generowanie adresów vanity to brutalne działanie: trzeba wypróbować klucz losowy, sprawdzić uzyskane adresy pod kątem ich zgodności z wymaganym wzorem i powtarzanie czynności aż do skutku. **Przykład 4-8** przedstawia “górnika vanity” - program stworzony w celu wyszukiwania adresów vanity napisanych w C++. Przykład wykorzystuje bibliotekę libbitcoin, którą przedstawiliśmy w części „[Alternatywni klienci, biblioteki i narzędzia](#)”.

Przykład 4-8. Miner adresów vanity

```
#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()
{
    std::random_device random;
    std::default_random_engine engine(random());
    // Loop continuously...
    while (true)
    {
        // Generate a random secret.
```

```

bc::ec_secret secret = random_secret(engine);
// Get the address.
std::string address = bitcoin_address(secret);
// Does it match our search string? (1kid)
if (match_found(address))
{
    // Success!
    std::cout << "Found vanity address! " << address << std::endl;
    std::cout << "Secret: " << bc::encode_hex(secret) << std::endl;
    return 0;
}
// Should never reach here!
return 0;
}
bc::ec_secret random_secret(std::default_random_engine& engine)
{
    // Create new secret...
    bc::ec_secret secret;
    // Iterate through every byte setting a random value...
    for (uint8_t& byte: secret)
        byte = engine() % std::numeric_limits<uint8_t>::max();
    // Return result.
    return secret;
}
std::string bitcoin_address(const bc::ec_secret& secret)
{
    // Convert secret to pubkey...
    bc::point pubkey = bc::secret_to_public_key(secret);
    // Finally create address.
    bc::payment_address payaddr;
    bc::set_public_key(payaddr, pubkey);
    // Return encoded form.
    return payaddr.encoded();
}
bool match_found(const std::string& address)
{
    auto addr_it = address.begin();
    // Loop through the search string comparing it to the lower case
    // character of the supplied address.
    for (auto it = search.begin(); it != search.end(); ++it, ++addr_it)
        if (*it != std::tolower(*addr_it))
            return false;
    // Reached end of search string, so address matches.
    return true;
}

```

Przykładowy kod musi być skompilowany w oparciu o kompilator C i połączony z biblioteką libbitcoin (która najpierw musi być zainstalowana w tym systemie). Aby uruchomić przykład, należy uruchomić plik wykonywalny vanity-miner++ bez parametrów (patrz: [Przykład 4-9](#)), który podejmie próbę znalezienia adresu vanity zaczynającego się od “1kid”.

Przykład 4-9. Przykład kompilowania i uruchamianie vanity-miner

```

$ # Compile the code with g++
$ g++ -o vanity-miner vanity-miner.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Run the Przykład
$ ./vanity-miner
Found vanity address! 1KiDzkG4MxmovZryZRj8tK81oQRhbZ46YT
Secret: 57cc268a05f83a23ac9d930bc8565bac4e277055f4794cbd1a39e5e71c038f3f
$ # Run it again for a different result
$ ./vanity-miner

```

```

Found vanity address! 1Kidxr3wsmMzzouwXibKfwTYs5Pau8TUFn
Secret: 7f65bbbb6d8caae74a0c6a0d2d7b5c6663d71b60337299a1a2cf34c04b2a623
# Use "time" to see how long it takes to find a result
$ time ./vanity-miner
Found vanity address! 1KidPWhKgGRQWD5PP5TAnGfDyfWp5yceXM
Secret: 2a802e7a53d8aa237cd059377b616d2bfca4b0140bc85fa008f2d3d4b225349

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

```

Przykładowy kod znajdzie szukany trzyliterowy wzór “kid” w ciągu kilku sekund; jak widać, wykorzystujemy polecenie unixowe time, aby zmierzyć czas realizacji. Wystarczy zmienić wzór przeszukania w kodzie źródłowym, aby przekonać się o ile dłużej będzie trwało wyszukiwanie wzorów cztero- lub pięcioznakowych!

Zabezpieczenia adresów vanity

Adresy vanity mogą być wykorzystywane w celu udoskonalenia oraz pokonania zabezpieczeń, ale jest to broń obosieczna. Może być wykorzystana w celu podniesienia bezpieczeństwa – pojedynczy adres utrudnia atakującym podstawienie swoich własnych adresów i oszukanie klientów, aby zapłacili im, zamiast Wam. Niestety, adresy vanity umożliwiają dowolnej osobie utworzenie adresu, który przypomina adres losowy, a nawet inny adres vanity i tym samym wprowadzenie waszych klientów w błąd.

Eugenia mogłaby reklamować losowo wygenerowany adres (np.1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy), na który chętni mogliby przelewać darowizny. Mogłaby też wygenerować adres vanity zaczynający się od 1Kids, aby go wyróżnić spośród innych.

W obu przypadkach, jedno z ryzyk związanych z korzystanie z jednego, stałego adresu (zamiast oddzielnego adresu dynamicznego dla każdego darczyńcy) jest to, że złodziej może być w stanie przeniknąć do Waszej strony i zastąpić Wasz adres swoim tym samym powodując przekierowanie środków na swoje konto. Jeżeli zareklamowaliście swój adres do wpłat w kilu różnych miejscach, Wasze użytkownicy mogą wizualne sprawdzić adres przed dokonaniem płatności, aby upewnić się, że jest on taki sam jak na stronie internetowej, w mailu lub na ulotce. W przypadku adresów losowych takich jak

1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy, przeciętny użytkownik może sprawdzić klikając pierwszych znaków “1J7mdg” i uzna, że adres jest dobry. Wykorzystując generator adresów vanity, osoba planująca kradzież poprzez podstawienie podobnie wyglądającego adresu może w bardzo prostu sposób wygenerować adresy o identycznych pierwszych znakach – patrz: **Tabela 4-13**.

Tabela 4-13. Generowanie adresów vanity pasujących do adresów losowych

Oryginalny adres losowy	1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
Vanity (4-znakowe dopasowanie)	1J7md1QqU4LpctBetHS2ZoyLV5d6dShhEy
Vanity (5-znakowe dopasowanie)	1J7mdgYqyNd4ya3UEcq31Q7sqRMXw2XZ6n
Vanity (6-znakowe dopasowanie)	1J7mdg5WxGENmwyJP9xuGhG5KRzu99BBCX

A zatem, czy adres vanity zwiększa bezpieczeństwo? Jeżeli Eugenia wygeneruje adres vanity:

1Kids33q44erFfpeXrmDSz7zEqG2FesZEN, użytkownicy będą się raczej przyglądać wzorom słów vanity i może kolejnym znakom, na przykład zauważając część “1Kids33” adresu. To zmusi atakującego do wygenerowania adresu vanity o dopasowanych sześciu znakach (dwa więcej) co stanowi pracę 3 364 razy (58×58) większą niż wysiłek Eugenii w zbudowanie jej czteroznakowego adresu vanity. Zasadniczo, wysiłek Eugenii (lub opłata na rzecz outsourcingowej puli vanity) “zmusza” atakującego do wygenerowania dłuższego wzoru vanity. Jeżeli Eugenia zapłaci puli za wygenerowanie 8-znakowego adresu vanity, atakujący będzie zmuszony do wygenerowania 10 zgodnych znaków, co jest nieopłacalne przy wykorzystaniu komputer klasy PC i kosztowne nawet przy wykorzystaniu zmodyfikowanego narzędzia górnictwego, czy puli vanity. Co opłaca się Eugenii staje

się nieopłacalne dla atakującego, w szczególności, jeśli potencjalna nagroda lub kwota wyłudzenia nie jest dostatecznie duża, aby pokryć koszty wygenerowania adresu vanity.

Portfele papierowe

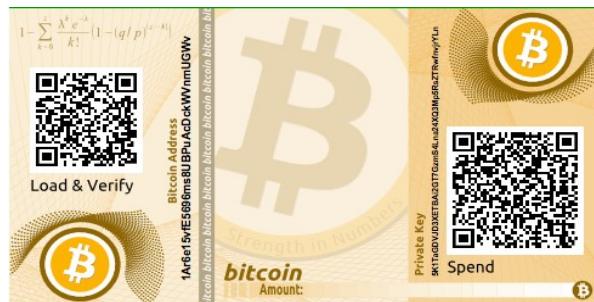
Portfele papierowe to prywatne klucze bitcoin wydrukowane na papierze. Często portfele papierowe zawierają także korespondujące adresy bitcoin dla wygody, ale nie jest to konieczne, ponieważ można je wygenerować w oparciu o klucz prywatny. Portfele papierowe to bardzo wydajny sposób tworzenia backupów lub przechowywania bitcoinów w trybie offline, nazywany także "cold storage." Jako mechanizm tworzenia kopii zapasowych, portfel papierowy może stanowić zabezpieczenie przed utratą klucza z uwagi na wypadki komputerowe w postaci na przykład uszkodzenie dysku twardego, kradzieży lub przypadkowego usunięcia. W przypadku mechanizmu "cold storage", jeżeli klucze portfela papierowego są generowane offline i nigdy nie zapisane w pamięci komputeraASA o wiele bardziej odporne na ataki hakerów, key-logerów oraz inne zagrożenia komputerowe.

Portfele papierowe mają wiele postaci, rozmiarów oraz konstrukcji, ale na poziomie podstawowym składają się z klucza o adresu wydrukowanego na papierze. **Tabela 4-14** przedstawia najprostszą postać portfela papierowego.

Tabela 4-14. Najprostszą postać portfela papierowego—wydruk adresów bitcoin i kluczy prywatnych.

Adres publiczny	Klucz prywatny (WIF)
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn

Portfele papierowe mogą być w prosty sposób generowane za pomocą narzędzi takich jak kliencki generator JavaScript na bitaddress.org. Strona ta zawiera wszystkie kody niezbędne do wygenerowania kluczy oraz portfeli papierowych, nawet przy całkowitym odłączeniu od Internetu. Aby móc z niego korzystać trzeba zapisać stronę HTML na lokalnym dysku lub na zewnętrznym nośniku USB. Odłączyć urządzenie od Internetu i otworzyć plik w przeglądarce. Można (co jest lepszym rozwiązaniem) zrestartować komputer wykorzystując systemie operacyjnym w czystej postaci, np. bootowalnym z CD-ROM-u Linuxie OS. Klucze wygenerowane za pomocą tego narzędzia w trybie offline mogą być wydrukowane na lokalnej drukarce połączonej kablem USB (nie bezprzewodowej) w ten sposób tworząc portfele papierowe, których klucze istnieją wyłącznie na papierze i które nie były nigdy wcześniej zapisane w żadnym systemie online. Warto zdeponować te papierowe portfele w ognioszczelnym sejfie i "wysłać" bitcoin na adres bitcoin, aby zaimplementować proste, ale bardzo wydajne rozwiązanie "cold storage". **Rys. 4-14** przedstawia portfel papierowy wygenerowany ze strony bitaddress.org.



Rys. 4-14. Przykładowy portfel papierowy ze strony bitaddress.org

Wadą prostego systemu portfeli papierowych jest to, że klucze wydrukowane można ukrąść. Złodziej, który może uzyskać dostęp do papieru, może je ukrąść lub zrobić zdjęcia i przejąć kontrolę nad bitcoinami chronionymi tymi kluczami. Bardziej zaawansowane systemy przechowywania portfeli papierowych

wykorzystując szyfrowanie kluczami prywatnymi BIP0038. Klucze wydrukowane w portfelu papierowym są chronione hasłem zapamiętywanym przez właściciela. Bez hasła, klucze te są bezużyteczne. Nie mniej jednak w dalszym ciągu są lepsze niż portfele chronione hasłem, ponieważ klucze te nigdy nie pojawiły się w sieci i musza być fizycznie wyjęte z sejfu lub z innego zabezpieczonego miejsca przechowywania. Rys. 4-15 przedstawia portfel papierowy z szyfrowanym prywatnym kluczem (BIP0038) utworzonym na stronie bitaddress.org.

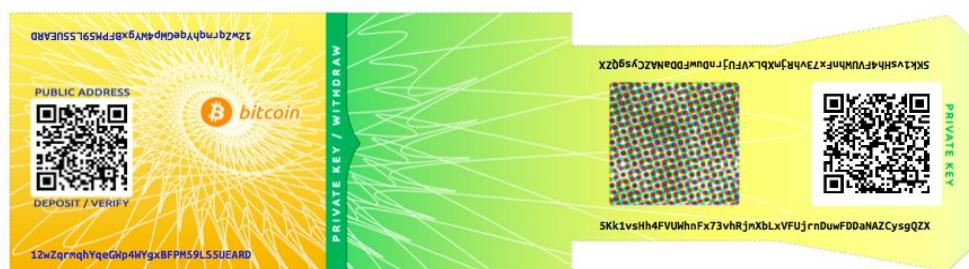


Rys. 4-15. Przykład szyfrowanego portfela papierowego ze strony bitaddress.org. Hasło to "test."



Chociaż istnieje możliwość deponowania funduszy kilkakrotnie w portfelu papierowym, można je pobrać tylko jeden raz i to całość. Jest tak dlatego, że proces deszyfrowania i wydawania środków powoduje ujawnienie klucza prywatnego, a niektóre portfele mogą generować adres dla reszty, jeżeli nie cała kwota środków jest wydawana. Jedną z opcji jest wybranie wszystkich dostępnych środków zdeponowanych w portfelu papierowym i przelanie pozostałych środków do nowego portfela papierowego.

Portfele papierowe mają różny wygląd i rozmiar oraz wiele elementów charakterystycznych. Niektóre z nich są wykorzystywane jako podarunku i zawierają wzory okazjonalne, np. Boże Narodzenie i Nowy Rok. Inne są przeznaczone do przechowywania w sejfie bankowym z prywatnym kluczem ukrytym albo pod nieprzejrzystymi zdrapkami lub złożone i zapieczętowane folią samoprzylepną. Rys. 4-16 do 4-18 przedstawiają różne przykłady portfeli papierowych z zabezpieczeniami i kopiami zapasowymi.



Rys. 4-16. Przykład portfela papierowego z bitcoinpaperwallet.com z prywatnym kluczem i zakładką.



Rys. 4-17. Portfel papierowy z ukrytym kluczem prywatnym.

Inne portfele mają dodatkowe kopie kluczy oraz adresów w postaci zdejmowanej nakładki lub przypominają bilet umożliwiając jednocześnie przechowywanie kilku kopii i ich ochronę przed ogniem, zalaniem lub innymi katastrofami naturalnymi.



Rys. 4-18. Przykład portfela papierowego z dodatkowymi kopiami kluczy na pasku backupu

Rozdział 5

Transakcje

Wstęp

Transakcje są najważniejszą częścią systemu Bitcoin. Wszystko inne w Bitcoin jest zaprojektowane po to, żeby wzmocnić pewność tworzenia transakcji, rozprzestrzeniania ich w sieci, zatwierdzania i wreszcie dodania do światowej księgi transakcji (łańcucha bloków). Transakcje są strukturami danych, które kodują transfer wartości między uczestnikami w systemie bitcoin. Każda transakcja jest wpisem publicznym w łańcuchu blokowym Bitcoin, globalnej księdze podwójnego zapisu.

W tym rozdziale zbadamy wszystkie możliwe formy transakcji, co zawierają, jak je tworzyć, w jaki sposób są weryfikowane i jak stają się częścią stałej ewidencji wszystkich operacji.

Cykl transakcji

Cykl życia transakcji rozpoczyna się od utworzenia tej transakcji, znanej również jako jej *zapoczątkowanie*. Transakcja zostaje podpisana jednym lub kilkoma podpisami nadającymi upoważnienie do wydania środków do których odwołują się transakcje. Transakcja zostaje przerzucona do szerokopasmowej sieci bitcoin, gdzie każdy węzeł sieci (uczestnik) zatwierdza i rozprzestrzenia transakcję aż do dotarcia do (prawie) każdego węzła w sieci. Wreszcie, transakcja jest weryfikowana przez węzeł eksploracyjny i zawarty w nim blok transakcyjny, który jest zapisywany w łańcuchu blokowym.

Po zapisie w łańcuchu blokowym i potwierdzeniu przez wystarczającą liczbę kolejnych bloków (potwierdzenia), transakcja staje się stałą częścią księgi bitcoin i jest akceptowana jako ważna przez wszystkich uczestników. Środki przypisywane nowemu właścielowi w wyniku tych czynności mogą być wydawane w nowej transakcji, przedłużając tym samym łańcuch własności i ponownie rozpoczynając cykl życia transakcji.

Tworzenie Transakcji

W pewnym sensie lepiej jest wyobrazić sobie transakcję w ten sam sposób jak czek tradycyjny, papierowy. Podobnie, jak czek transakcja jest instrumentem, który wyraża zamiar przekazania pieniędzy w sposób niewidoczny w systemie finansowym aż do momentu realizacji czeku. Jak w przypadku czeku, inicjator transakcji nie musi być tym samym, który podpisuje transakcję.

Transakcje mogą być tworzone online lub offline przez każdego, nawet jeśli dana osoba tworząca transakcje nie jest autoryzowanym sygnatariuszem konta. Na przykład: osoba odpowiedzialna za realizację zobowiązań może przetwarzać czeki, które podpisuje Dyrektor Generalny. Podobnie, osoba odpowiedzialna za realizację zobowiązań może tworzyć transakcje bitcoinowe i jeśli są autoryzowane cyfrowym podpisem Dyrektora Generalnego to czyni je ważnymi. Podczas, gdy czek odwołuje się do określonego konta jako źródła funduszy, transakcja Bitcoin odwołuje się do konkretnej poprzedniej transakcji jako źródła, nie zaś do rachunku.

Po utworzeniu, transakcja jest podpisana przez właściciela (lub właścicieli) środków źródłowych. Jeśli jest prawidłowo utworzona i podpisana staje ważna i zawiera wszystkie informacje potrzebne do wykonania przelewu środków. Wreszcie ważna transakcja musi dotrzeć do sieci Bitcoin tak, żeby mogła być rozprzestrzeniona aż dotrze do górnika w celu umieszczenia w księdze głównej (łańcuchu blokowym).

Rozprzestrzenianie transakcji po sieci Bitcoin

Po pierwsze transakcje muszą być dostarczone do sieci Bitcoin tak, by mogły być rozpropagowane i zawarte w łańcuchu blokowym. W istocie, transakcja Bitcoin to zaledwie 300 do 400 bajtów danych, które muszą dotrzeć do każdego z dziesiątek tysięcy węzłów bitcoinowych. Nadawcy nie muszą darzyć zaufaniem węzłów, których używają do transmisji transakcji tak długo jak używają więcej niż jednego węzła, by zapewnić rozprzestrzenianie się transakcji. Węzły nie muszą ufać nadawcy lub ustalać tożsamość nadawcy, ponieważ transakcja jest podpisana i nie zawiera informacji poufnych, prywatnych haseł czy pełnomocnictw, może być transmitowana używając jakiekolwiek dostępnej sieci transportowej. W odróżnieniu od transakcji dokonywanych np. kartą kredytową, które zawierają informacje poufne i mogą być przesyłane tylko drogą szyfrowaną, transakcje Bitcoin mogą być wysyłane przez dowolną sieć. Jeżeli transakcja może docierać do węzła bitcoin, będzie także rozsyłana po sieci. Nie ma natomiast znaczenia jej sposób przetransportowania do pierwszego węzła. Transakcje mogą być zatem przekazywane do sieci bitcoin na niezabezpieczonych sieciach, takich jak Wi-Fi, Bluetooth, NFC, Chirp, za pomocą kodów kreskowych lub przez skopiowanie i wklejenie w formie internetowej. W skrajnych przypadkach, transakcja Bitcoin może być transmitowana przez odbiornik radiowy, przekaźnik satelitarny lub na falach krótkich z wykorzystaniem transmisji rozrywającej, o widmie rozproszonym, lub przeskoku częstotliwości w celu uniknięcia wykrycia i zacinania. Transakcja Bitcoin może nawet być zakodowana jako uśmiezki (emotikony) i publikowana na forum publicznym lub wysyłana w postaci wiadomości tekstuowej lub wiadomości czatu na Skype. Bitcoin zwraca pieniądze w strukturze danych, dzięki czemu jest praktycznie niemożliwe, aby powstrzymać kogokolwiek od tworzenia i realizacji transakcji Bitcoin.

Rozprzestrzenianie transakcji w sieci bitcoin.

Gdy transakcja bitcoin jest wysyłana do dowolnego węzła połączonego z siecią bitcoin, dotarcie transakcji zostanie potwierdzone przez ten węzeł. Jeśli jest ważna, węzeł ten będzie rozprzestrzeniać transakcję do innych węzłów, do których jest podłączony, a komunikaty o skutecznej transakcji będą wracać do nadawcy synchronicznie. Jeśli transakcja jest nieważna, węzeł będzie ją odrzucać i synchronicznie wyśle komunikat o odrzuceniu do zleceniodawcy.

Bitcoin to sieć typu peer-to-peer (każdy z każdym), co oznacza, że każdy węzeł jest połączony z kilkoma innymi węzłami w sieci, które ujawnią się podczas uruchamiania przez protokół peer-to - peer. Cała sieć tworzy luźno związany układ siatki bez ustalonej topologii lub dowolnej struktury, dzięki czemu wszystkie węzły są równe innym. Wiadomości, w tym transakcje i bloki, są rozprzestrzeniane pomiędzy węzłami, do których są podłączone. Nowe, zatwierdzone transakcje wpięte w dowolny węzeł sieci będą wysłane do od trzech do czterech węzłów sąsiednich, przy czym każdy z nich przesyłać je do od trzech do czterech kolejnych węzłów i tak dalej. W ten sposób, w ciągu kilku sekund ważna transakcja będzie rozprzestrzeniać się w postępie wykładniczym, rozwijając się po całej sieci, aż zostanie doręczona do

wszystkich podłączonych węzłów. Sieć bitcoin jest przeznaczona do rozprzestrzeniania transakcji i bloków do wszystkich węzłów w efektywny i preżny sposób, który w dodatku jest odporny na ataki.

Aby uniknąć spamu, ataku denial-of service (atak typu odmowa usług) lub innych uciążliwych ataków przeciwko systemowi bitcoin, węzły niezależnie sprawdzają każdą transakcję przed przesłaniem jej dalej. Uszkodzona transakcja nie wydostanie się poza węzeł. Zasady zatwierdzania transakcji zostały opisane bardziej szczegółowo w części "Niezależna weryfikacja transakcji".

Struktura transakcji

Transakcja jest *strukturą danych*, która koduje transfer wartości ze źródła funduszy, zwanego blokiem wejściowym, do miejsca docelowego, zwanego blokiem wyjściowym. Bloki wejściowe i wyjściowe transakcji nie są związane z kontami lub tożsamościami. Zamiast tego, należy myśleć o nich jako wartościach bitcoinowych – fragmentach bitcoinów - które zostały zablokowane kluczem znanym jedynie dla właścicieli i osobom, którym ten klucz powierzył. Tylko osoby, które znają klucz mogą odblokować transakcję. Transakcja zawiera pewną liczbę pól – patrz **Tabela 5-1**.

Tabela 5-1. Struktura transakcji

Rozmiar	Obszar	Opis
4 bajty	version (wersja)	Określa zasady realizacji transakcji
1-9 bajtów (VarInt)	Licznik wejściowy	Ile wejść jest włączonych, uwzględnionych
zmienna	Bloki wejściowe	Jedno lub więcej wejść transakcyjnych
1-9 bajtów (VarInt)	Licznik wyjściowy	Ile wyjść jest włączonych, uwzględnionych
Zmienna	Bloki wyjściowe	Jedno lub więcej wyjść transakcyjnych
4 bajty	Locktime (czas blokady)	Datownik Unix albo numerów bloków

Czas blokady transakcji (Locktime)

Czas blokady określa najwcześniejszy czas, w którym transakcja może zostać dodana do łańcucha bloków. Jest on ustawiony na zero w większości transakcji, aby wskazać natychmiastowość wykonania. Jeśli czas blokady jest niezerowy i poniżej 500 milionów, jest interpretowany jako wysokość bloku, co oznacza, że transakcja nie jest wliczona w łańcuch bloków przed ustaloną wysokością bloku. Jeśli czas jest powyżej 500 milionów, jest interpretowany jako datownik Unix Epoch (sekundy od 1 stycznia 1970) oraz transakcja

nie jest wliczona w łańcuch bloków przed upływem określonego czasu. Zastosowanie czasu blokady jest równoważne do daty wysłania czeku tradycyjnego.

Bloki wyjściowe i wejściowe transakcji

Podstawowym budulcem transakcji bitcoin jest *niewydatkowany blok wynikowy transakcji* lub UTXO. UTXO są niepodzielnymi kawałkami waluty bitcoin zablokowanymi dla konkretnego właściciela, zamieszczonymi na łańcuchu blokowym i uznanyymi za jednostki walutowe przez całą sieć. Sieć Bitcoin śledzi wszystkie dostępne (niewykorzystane) UTXO liczące się obecnie w milionach. Zawsze, gdy użytkownik otrzymuje Bitcoiny, kwota ta jest rejestrowana w łańcuchach blokowych jako UTXO. Zatem Bitcoiny użytkownika mogą być rozproszone w UTXO spośród setek transakcji i bloków. W efekcie, nie ma czegoś takiego jak przechowywanie bilansu adresów bitcoinów lub kont; są to tylko rozproszone UTXO, zablokowane dla poszczególnych właścicieli. Pojęcie równowagi użytkownika bitcoin jest pochodną konstruktu utworzonego przez aplikację portfela. Portfel oblicza saldo użytkownika poprzez skanowanie łańcucha blokowego i sumując wszystkie UTXO należące do danego użytkownika.



Nie ma żadnych rachunków lub sald w sieci Bitcoin; są tylko *niewydatkowane bloki wyjściowe transakcji* (UTXO) rozproszone w łańcuchu blokowym.

UTXO może mieć dowolną wartość określaną jako wielokrotność satoshi. Podobnie jak dolary mogą być podzielone do dwóch miejsc po przecinku jako centy; bitcoiny można także podzielić do ośmiu miejsc po przecinku, jako satoshi. Chociaż UTXO mogą mieć dowolną wartość, są stworzone po to, by były niepodzielne, podobnie jak moneta nie może być przepołowiona. Jeżeli UTXO jest większa niż wymagana wartość transakcji, to wciąż musi być zużywana jako całość, a zmiany muszą być wygenerowane w transakcji. Innymi słowy: jeśli masz 20 Bitcoinów UTXO i chcesz zapłacić 1 Bitcoin, Twoja transakcja musi zużywać całą pulę 20 Bitcoin UTXO i to generuje dwa rozwiązania: pierwsze - płacisz 1 Bitcoin określonemu odbiorcy, a pozostałe 19 bitcoinów trafia z powrotem do portfela. W rezultacie, większość transakcji bitcoinowych generuje zmiany.

Wyobraźcie sobie klienta kupującego napój za 1,50 dolara, który sięgając do swojego portfela próbuje znaleźć kombinację monet i banknotów na pokrycie tej kwoty. Kupujący, o ile jest w posiadaniu różnych pieniędzy wybierze je (np. dolar i dwie ćwiartki), lub utworzy kombinację używając mniejszych nominałów (sześć ćwiartek), lub jeśli jest to konieczne, użyje większej jednostki, takiej jak np. banknot pięciodolarowy. Jeśli damy właścicielowi sklepu więcej pieniędzy za wspomniany napój, powiedzmy 5 dolarów, będziemy oczekiwali reszty w kwocie 3,50, która wróci do naszego portfela i będzie dostępna w celu realizacji przyszłych transakcji.

Podobnie jest w transakcjach bitcoinów, które muszą być tworzone z UTXO użytkownika w dowolnych nominałach będących do dyspozycji użytkownika. Użytkownicy nie mogą przeciąć UTXO na pół, tak jak nie mogą przeciąć banknotu na pół i używać połówek jako waluty. Aplikacja portfelowa użytkownika zazwyczaj wybiera spośród dostępnych UTXO różne jednostki i łączy je w kwotę większą lub równą żądanej kwocie transakcji.

Tak jak w prawdziwym życiu, aplikacja Bitcoin może wykorzystywać kilka strategii w celu pokrycia kwoty zakupu: łączenie kilku mniejszych jednostek, złożenie dokładnej kwoty z drobnych lub płatność za pomocą jednej jednostki większej od wartości transakcji i uzyskanie z niej reszty. Wszystkie te skomplikowane operacje z UTXO odbywają się poprzez portfel użytkownika automatycznie i są niewidoczne dla użytkowników. To ma zastosowanie wyłącznie jeśli programowo konstruujesz transakcje RAW z UTXO.

UTXO wykorzystywane w transakcji są nazywane blokami wejściowymi transakcji, a UTXO stworzone przez transakcję są nazywane blokami wyjściowymi transakcji. W ten sposób kawałki wartości bitcoin idą z właściciela na właściciela w łańcuchu transakcji zużywając i tworząc UTXO. Transakcje zużywają UTXO poprzez uwolnienie go z podpisem obecnego właściciela i stworzenie UTXO przez ulokowanie go na adresie bitcoinowym nowego właściciela.

Wyjątkiem dla łańcucha wyjściowego i wejściowego jest specjalny rodzaj transakcji zwany transakcją *coinbase*, która jest pierwszą transakcją w każdym bloku. Transakcja ta jest umieszczona tam przez "zwycięskiego" górnika i tworzy zupełnie nową markę - nowe Bitcoiny należne temu górnikowi jako nagroda za jego kopanie. W ten sposób podaż pieniądza Bitcoin jest tworzony w procesie wydobywczym, jak zobaczymy w [Rozdziale 8](#).



Co następuje wcześniej? Bloki wejściowe czy wyjściowe; kura czy jajko? Ścisłe mówiąc, bloki wyjściowe są pierwsze, bo transakcje coinbase, które generują nowe Bitcoiny, nie mają bloku wejściowego i tworzą bloki wyjściowe z niczego.

Bloki wyjściowe transakcji

Każda transakcja Bitcoin tworzy bloki wyjściowe, które są zapisywane w księdze Bitcoin. Prawie wszystkie z tych wyjść, z jednym wyjątkiem (patrz: "[Dane wyjściowe \(OP_RETURN\)](#)") tworzą fragmenty bitcoinów, które można wydać, zwane niewykorzystanymi wyjściami transakcji lub UTXO, które następnie są rozpoznawane przez całą sieć i dostępne dla właściciela, by mógł je wydać w przyszłych transakcjach. Wysyłanie komuś Bitcoinów jest tworzone jako niewykorzystany blok wyjściowy transakcji (UTXO) zarejestrowanu na jego adres i dostępny do wydawania.

UTXO są śledzone przez każdy pełny węzeł klienta bitcoin w bazie danych przechowywanej w pamięci, zwanej *zbiorem lub pulą UTXO*. Nowe transakcje zużywają (wydatkują) jedno lub więcej z tych bloków wyjściowych ze zbioru UTXO.

Bloki wyjściowe transakcji składają się z dwóch części:

- kwoty bitcoin, nominowanej w *satoshi*, najmniejszej jednostce Bitcoin
- Skryptu blokującego, znanego również jako "ograniczenie", które "blokuje" tę kwotę określając warunki, jakie muszą być spełnione, aby mogła wyjść.

Język skryptowy transakcji, używany w wspomnianym wcześniej skrypcie blokującym, jest szczegółowo omówiony w rozdziale "Skrypty transakcyjne i język skryptów" na stronie 121. Tabela 5-2 pokazuje strukturę bloków wyjściowych transakcji.

Tabela 5.2 – struktura bloku wyjściowego transakcji

Rozmiar	Obszar	Opis
8 bajtów	Kwota	Wartość bitcoinów wyrażona w satoshi
1-9 bajtów	Rozmiar blokującego skrypt	skrypt blokujący wyrażony w bitach podlegający obserwacji
Zmienny	Skrypt blokujący	skrypt definiujący warunki potrzebne do wyjścia

W [Przykładzie 5-1](#), używamy API blockchain.info aby znaleźć niewykorzystane bloki wyjściowe (UTXO) konkretnego adresu.

Przykład 5-1. Skrypt, który wywołuje API blockchain.info do znalezienia UTXO związanego z adresem.

```

# get unspent outputs from blockchain API

import json
import requests

# example address
address = '1Dorian4RoXcnBv9hnQ4Y2C1an6NJ4UrjX'

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

```

Przykład 5-2. Uruchomienie skryptu get.utxo.py

```

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

```

Warunki wydatkowania (ograniczenia)

Bloki wyjściowe transakcji skojarzone są z konkretną kwotą (podaną w satoshi) skojarzoną do konkretnego ograniczenia lub skryptu blokującego, który definiuje warunki, które muszą być spełnione, aby wydać tę kwotę. W większości przypadków skrypt ogranicza blok wyjściowy do określonego adresu Bitcoin, a tym samym przeniesienie własności tej kwoty na nowego właściciela. Kiedy Alicja zapłaciła w kawiarni Boba za kawę, jej transakcja utworzyła blok wyjściowy 0,015 w bitcoin z zablokowanym adresem konta Bitcoin kawiarni. To wyjście 0,015 Bitcoin zostało zapisane na łańcuchu blokowym i stało się częścią niewykorzystanych transakcji wyjściowych, co oznacza, że transakcja pokazała się w portfelu Boba w ramach dostępnego salda. Gdy Bob zdecyduje się wydać tę kwotę, jego transakcji wyda obciążanie, odblokowując blok wyjściowy poprzez dostarczenie odblokowującego skryptu zawierającego podpis prywatnego klucza Boba.

Bloki wejściowe transakcji

W prostych słowach, bloki wejściowe transakcji są wskaźnikami do UTXO. Zwracają się do konkretnego UTXO przez odniesienie do hasza transakcji i numeru sekwencji, gdzie UTXO jest zapisywane w łańcuchu blokowym. Aby móc wydać UTXO, wejście transakcji zawiera również odblokowane skrypty, które spełniają warunki wydawania określone przez UTXO. Skrypt odblokowane zwykle są podpisane, co udowadnia posiadanie adresu bitcoin, który jest zawarty w skrypcie blokującym.

Gdy użytkownicy dokonują płatności, ich portfel konstruuje transakcję wybierając z dostępnych UTXO. Na przykład, aby dokonać płatności 0,015 Bitcoin, Aplikacja portfelowa może wybrać 0,01 UTXO i 0.005 UTXO, użyć ich obu by uzyskać żądaną kwotę płatności.

W [Przykładzie 5-3](#), pokazujemy zastosowanie "zachłannego" algorytmu, aby wybrać spośród dostępnych UTXO w celu dokonania określonej płatności. Na przykład, dostępne UTXO są dostarczane w postaci stałego szyku, ale w rzeczywistości dostępne UTXO byłyby pobierane z wywołania RPC do Bitcoin Core lub do API strony trzeciej, jak pokazano w [Przykładzie 5-1](#).

Przykład 5-3. Skrypt do obliczania, ile będzie wydawanych całkowitych Bitcoinów.

Example 5-3. A script for calculating how much total bitcoin will be issued

```

# Selects outputs from a UTXO list using a greedy algorithm.

from sys import argv

class OutputInfo:

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

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

# Select optimal outputs for a send from unspent outputs list.
# Returns output list and remaining change to be sent to
# a change address.
def select_outputs_greedy(unspent, min_value):
    # Fail if empty.
    if not unspent:
        return None
    # Partition into 2 lists.
    lessers = [utxo for utxo in unspent if utxo.value < min_value]
    greater = [utxo for utxo in unspent if utxo.value >= min_value]
    key_func = lambda utxo: utxo.value
    if greater:
        # Not-empty. Find the smallest greater.
        min_greater = min(greater)
        change = min_greater.value - min_value
        return [min_greater], change
    # Not found in greater. Try several lessers instead.
    # Rearrange them from biggest to smallest. We want to use the least
    # amount of inputs as possible.
    lessers.sort(key=key_func, reverse=True)
    result = []
    accum = 0
    for utxo in lessers:
        result.append(utxo)

```

Przykład 5-4. Uruchamianie skryptu select-utxo.py

Example 5-4. Running the select-utxo.py script

```
$ python select-utxo.py 50000000
For transaction amount 50000000 Satoshi (0.500000 bitcoin) use:
```

```
{[<7dbc497969c7475e45d952c4a872e213fb15d45e5cd3473c386a71a1b0c136a1:0 with 25000000
Satoshis>, <7f42eda67921ee92eae5f79bd37c68c9cb859b899ce70dba68c48338857b7818:0 with
16100000 Satoshis>,
<6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ecf:0 with 16050000
Satoshis>], 'Change: 7150000 Satoshis')}
```

Po wybraniu UTXO, portfel produkuje odblokowane skrypty zawierające podpisy dla każdego UTXO, tym samym czyniąc je gotowym do wyjścia poprzez zaspokojenie warunków skryptów blokujących. Portfel dodaje odnośniki UTXO i odblokowane skrypty jako wejścia do transakcji. Tabela 5-3 pokazuje strukturę wprowadzania transakcji.

Tabela 5-3 Struktura wprowadzania transakcji

Rozmiar	Obszar	Opis
32 bajty	Hash transakcji	Wskaźnik do transakcji zawierający UTXO do wydania
4 bajty	Wskaźnik wyjścia	Numer indeksu UTXO do wydania; Pierwszym z nich jest 0
1-9 bajty (zmienna typu całkowitego)	Rozmiar odblokowania skryptu	Długość odblokowanego skryptu w bajtach
Zmienny	Odblokowany skrypt	Skrypt, który spełnia warunki skryptu blokującego UTXO.
4 bajty	Numer sekwencji	Obecnie wyłączona funkcja zastępcza Tx, ustawiona na 0xFFFFFFFF



Numer sekwencji służy do zastępowania transakcji przed upływem czasu blokowania transakcji, która ta funkcja jest obecnie wyłączona w Bitcoin. Większość transakcji ustawia tę wartość do maksymalnej liczby całkowitej (0xFFFFFFFF) i jest to ignorowane przez sieć Bitcoin. Jeśli transakcja ma niezerowy czas blokady, co najmniej jedno z jej wejść musi mieć liczbę sekwencji poniżej 0xFFFFFFFF w celu umożliwienia czasu blokady.

Opłaty transakcyjne

Większość transakcji zawiera opłaty transakcyjne, które są kompensatorem dla górników Bitcoin za zabezpieczanie sieci. Górnictwo, wypłaty i nagrody zebrane przez górników zostały omówione bardziej szczegółowo w rozdziale 8. W tej sekcji przedstawiono w jaki sposób opłaty transakcyjne są zawarte w typowej transakcji. Większość portfeli oblicza i obejmuje opłaty transakcyjne automatycznie. Jednakże, jeśli się konstruuje transakcję programowo lub za pomocą wiersza poleceń interfejsu, musisz ręcznie obliczyć kwotę tych opłat. Opłaty transakcyjne są zachętą do zawarcia (kopania) transakcji do następnego

bloku, a także jako czynnikiem zniechęcającym wobec transakcji "spam" lub jakiegokolwiek nadużycia systemu, przez nałożenie niewielkiej opłaty od każdej transakcji. Opłaty transakcyjne pobierane są przez górnika, który kopie w bloku, rejestrując transakcję na łańcuchu.

Opłaty transakcyjne są obliczane na podstawie wielkości transakcji w kilobajtach, a nie w wartościach Bitcoinów. Ogólnie rzecz biorąc, opłaty transakcyjne są ustalane na podstawie sił rynkowych w sieci bitcoin. Górnicy tworzą priorytety transakcji na podstawie wielu różnych kryteriów, w tym opłat, nawet mogą przetwarzać transakcje za darmo w pewnych okolicznościach. Opłata transakcyjna wpływa na priorytet przetwarzania, co oznacza, że transakcja z wystarczającymi opłatami może być uwzględniona w kolejnym, najbardziej zaminowanym bloku, podczas gdy transakcja z niewystarczającą lub brakiem opłatami może być opóźniona, przetwarzana na zasadzie best-effort po kilku blokach lub nieprzetworzona w ogóle. Opłaty transakcyjne nie są obowiązkowe, a transakcje bez opłat mogą być ewentualnie przetwarzane; jednakże załączenie opłaty transakcyjnej zachęca do priorytetowego przetwarzania.

Z biegiem czasu, metoda liczenia opłat transakcyjnych oraz ich wpływ na ustalanie priorytetów transakcji ewoluje. Początkowo opłaty transakcyjne zostały ustalone jako stałe w całej sieci. Stopniowo struktury opłat zostały złagodzone tak, że są podległe siłom rynkowym, w zależności od pojemności sieci i wolumenu transakcji. Obecna minimalna opłata transakcyjna jest ustalona na 0,0001 bitcoin lub dziesiątą milli-bitcoina na kilobajt, niedawno spadła z jednego milli-bitcoina. Większość transakcji jest mniejsza niż jeden kilobajt; jednak dla tych z wieloma wejściami i wyjściami mogą być większe. W przyszłych wersjach protokołu bitcoin, oczekuje się, że aplikacje portfela będą wykorzystywać analizę statystyczną, aby obliczyć najbardziej odpowiednią opłatę na dołączenie do transakcji opartej na średnich opłatach ostatnich transakcji.

Obecny algorytm używany przez górników do priorytetowego traktowania transakcji do włączenia do bloku w oparciu o ich opłaty jest badany szczegółowo w [Rozdziale 8](#).

Dodawanie opłat do transakcji

Struktura danych transakcji nie ma pola dla opłat. Zamiast tego opłaty są regulowane jako stosunek różnicy sumy wejść do sumy wyjść. Każda nadwyżka, która pozostaje po wszystkich wyjściach zostaje odjęta od wszystkich wejść jest opłatą pobieraną przez górników.

Opłaty transakcyjne są regulowane, jako nadwyżka wejść minus wyjścia:

$$\text{Fees} = \text{Suma (bloki wejściowe)} - \text{Suma (bloki wyjściowe)}$$

To jest trochę mylącym elementem transakcji i ważnym punktem do zrozumienia, bo jeśli chcesz tworzyć własne transakcje musisz się upewnić, że nie zawierasz przypadkowo za dużych opłat za niewykorzystane wejścia. Oznacza to, że "trzeba uwzględnić sumę dla wszystkich wejść, jeśli to konieczne przez stworzenie reszty lub skończysz dając górnikom bardzo duży napiwek!"

Na przykład, wykorzystując 20 bitcoinowe UTXO do dokonania płatności 1 Bitcoina, należy dołączyć blok wyjściowy z 19-Bitcoinową resztą, która wróci z powrotem do Twojego portfela. W przeciwnym wypadku, 19-Bitcoinowe "resztki" będą liczone jako opłaty transakcyjne i zostaną zebrane przez górnika, który wykonuje kopanie w Twoim bloku. Chociaż otrzymasz w zamian priorytet przetwarzania i zapewne uczynisz górnika szczęśliwym, to nie będzie chyba to tym, co miałeś zamiar zrobić.



Jeśli zapomnisz dodać reszyt do blok wyjściowy w ręcznie skonstruowanej transakcji, będziesz płacić resztę jako opłatę transakcyjną. "Zachowaj resztę" nie może być tym, co zamierzałeś.

Zobaczmy, jak to działa w praktyce, patrząc ponownie na kawę Alicji. Alicja chce wydać 0,015 Bitcoin aby zapłacić za kawę. By upewnić się, że ta transakcja będzie szybko przetwarzana ona będzie chciała dołączyć opłatę transakcyjną, powiedzmy 0,001. To będzie oznaczać, że całkowity koszt transakcji wyniesie 0,016. Jej portfel musi zawierać zestaw UTXO który dodaje do 0,016 bitcoin lub więcej, a jeśli to konieczne, utworzy resztę. Powiedzmy, że jej portfel ma dostępne 0,2-Bitcoin UTXO. Będzie zatem UTXO potrzebny do spożycia, tworząc jedno blok wyjściowy do kawiarni Boba na 0,015, a drugie blok wyjściowy z 0,184 Bitcoin jako reszta, która trafia z powrotem do portfela Alicji, pozostawiając 0,001 Bitcoin nieprzydzielone, jako domniemana opłata za transakcję.

Teraz spójrzmy na inny scenariusz. Eugenia, dyrektor fundacji na rzecz dzieci na Filipinach, zakończyła zbiórkę pieniędzy na zakup podręczników szkolnych dla dzieci. Otrzymała kilka tysięcy drobnych datków od ludzi na całym świecie, w sumie 50 Bitcoin, więc jej portfel jest pełen bardzo małych kwot (UTXO). Teraz chce kupić setki książek szkolnych od lokalnego wydawcy, płacąc Bitcoinami.

Kiedy Aplikacja portfelowa Eugenii próbuje skonstruować pojedynczy większy model transakcji płatniczej, który musi pozyskać źródło z dostępnego zestawu UTXO, który składa się z wielu mniejszych ilości. Oznacza to, że uzyskany wynik transakcji będzie zaopatrywany z ponad stu wartości UTXO jako bloków wejściowych i mając przy tym tylko jedno blok wyjściowy, płacąc wydawcy za książki. Transakcja z wieloma wejściami będzie większa niż jeden kilobajt, być może będzie w rozmiarze od 2 do 3 kilobajtów. W rezultacie konieczne będą opłaty wyższe niż minimalne za użytkowanie sieci.

Aplikacja portfelowa Eugenii wyliczy odpowiednią opłatę poprzez pomiar wielkości transakcji i mnożąc rozmiar przez opłaty za kilobajt. Wiele portfeli będzie przepłacać za opłaty dla większych transakcji, aby zapewnić sobie szybkość przetwarzania transakcji. W tym przypadku wyższa opłata nie jest spowodowana tym, że Eugenia wydała więcej pieniędzy, ale tym, że jej transakcja jest bardziej złożona i większa, więc opłata jest niezależna od wartości bitcoin tej transakcji.

Transakcja łańcuchowe i sieroce

Jak widzieliśmy, transakcje tworzą łańcuch, przy czym jedna transakcja wychodzi blok wyjściowy m poprzedniej transakcji (zwana jednostką macierzystą) i tworzy wyjścia dla kolejnej transakcji (znanej jako dziecko). Czasami cały łańcuch transakcji jest od siebie zależny, powiedzmy jak rodzice, dziecko i wnuki transakcji - są tworzone w tym samym czasie, aby zrealizować złożony obieg transakcyjny, która wymaga prawidłowego podpisu dzieci przed podpisem rodziców. Na przykład, jest to technika stosowana w transakcjach CoinJoin, w którym wiele stron transakcji łączy się by wspólnie chronić swoją prywatność.

Kiedy łańcuch transakcji jest przesyłany przez sieć, nie zawsze jego treść dociera do miejsca w tej samej kolejności. Zdarza się, że dziecko może dotrzeć przed rodzicem. W takim przypadku, węzły, które widzą dziecko jako pierwsze, mogą zobaczyć odwołanie do transakcji rodzica, który nie jest jeszcze znany. Zamiast odrzucić dziecko, lepiej jest umieścić go tymczasowym puli aż do przybycia jego rodzica i

przejścia do kolejnego węzła. Zbiornik transakcji bez rodziców jest znany jako zbiornik transakcji sierocich. Gdy rodzic przybywa wszelkie sieroty odwołujące się do UTXO utworzonego przez rodzica są uwalniane ze zbiornika, doprowadzając do przywrócenia swojej ważności rekurencyjnie, a następnie cały łańcuch transakcji może być zawarty w puli transakcji, już gotowy do eksploatawania w bloku. łańcuchy transakcji mogą być dowolnie długie, z dowolną liczbą pokoleń transmitowanych jednocześnie. Mechanizm umieszczenia sierot w puli sierocym gwarantuje, że ważne transakcje nie zostaną odrzucone tylko dlatego, że dotarcie rodzica zostało opóźnione i że w łańcuchu do którego należą zostanie zrekonstruowany we właściwej kolejności, niezależnie od kolejności przybycia.

Jest pewien limit liczby transakcji sierocich przechowywanych w pamięci, aby zapobiec atakowi denial-of-service przeciwko węzłom Bitcoin. Granic jest definiowana jako MAX_ORPHAN_TRANSACON w kodzie źródłowym klienta referencyjnego Bitcoin. Jeśli liczba transakcji sierocich w basenie przekracza MAX_ORPHAN_TRANSACTIONS, jedna lub więcej losowo wybranych transakcji sierocich jest wyrzucana ze zbiornika, dopóki rozmiar zbiornika nie wróci do dopuszczalnych granic.

Skrypty transakcji oraz język skryptów

Klienci bitcoiny zatwierdzają transakcję przez wykonanie skryptu napisanego w Forth - jako język skryptu. Zarówno skrypt blokujący (obciążenie) umieszczony na UTXO i skrypt odblokowujący, który zazwyczaj zawiera podpis są napisane w tym języku skryptowym. Kiedy transakcja zostanie zatwierdzona, skrypt odblokowania w każdym wejściu jest wykonywany obok odpowiedniego skryptu blokującego, aby sprawdzić, czy spełnia warunek wydatków.

Obecnie większość transakcji przetwarzanych przez sieć Bitcoin ma postać "Alicja płaci Bobowi" i są oparte na tym samym skrypcie zwanym skryptem Pay-to-Public-Key-Hash. Jednak wykorzystanie skryptów, aby zablokować wyjścia i odblokować wejścia oznacza, że dzięki zastosowaniu języka programowania, transakcje mogą zawierać nieskończoną liczbę warunków. Transakcje Bitcoin nie ograniczają się do formy "Alicja płaci Bobowi".

To tylko wierzchołek góry lodowej możliwości, które mogą być wyrażone z tym językiem skryptowym. W tej części pokażemy elementy języka skryptowego transakcji Bitcoin i to, jak może on być stosowany do wyrażania złożonych warunków wydatków i jak te warunki mogą być spełnione przez uwolnienie skryptów.



Walidację transakcji Bitcoin nie opiera się na statycznym wzorze, lecz uzyskuje się poprzez wykonanie języka skryptowego. Język ten pozwala na wyrażenie niemal nieskończonej różnorodność warunków. To jak Bitcoin pobiera moc z "programowalnych pieniędzy".

Budowa Skryptu (blokowanie i odblokowanie)

Silnik walidacji transakcji Bitcoin opiera się na dwóch rodzajach skryptów do zatwierdzania transakcji: skrypt blokujący i skrypt odblokowujący. Skrypt blokujący jest obciążaniem umieszczonym na wyjściu i określa warunki, które muszą być spełnione, aby opuścić blok wyjściowy w przyszłości. Historycznie, skrypt blokujący nazwano scriptPubKey, ponieważ zwykle zawierał publiczny klucz lub adres Bitcoin. W tej książce mówimy o nim jako o „skrypcie blokującym” w celu potwierdzenia znacznie szerszego wachlarza możliwości tej technologii skryptów. W większości w aplikacjach Bitcoin, to co określamy jako skrypt blokujący pojawią się w kodzie źródłowym jak scriptPubKey.

Skrypt odblokowujący jest skryptem, który "rozwija" lub wspiera warunki umieszczone na wyjściu przez skrypt blokujący i umożliwia blok wyjściowy być wydane. Uwalniające skrypty są częścią każdego bloku wejściowego transakcji, a większość czasu zawierają podpis cyfrowy wyprodukowany przez portfel użytkownika z jego klucza prywatnego. Historycznie rzecz biorąc, skrypt odblokowujący nazywa się scriptSig, ponieważ zwykle zawiera podpis cyfrowy. W większości zastosowań Bitcoin, kod źródłowy odnosi się do skryptu odblokowującego jako scriptSig. W tej książce, mówimy o nim jako "scenariuszu odblokowania", aby podkreślić szersze znacznie rangi wymagań blokowania skryptów, ponieważ nie wszystkie skrypty odblokowywane muszą zawierać podpisy.

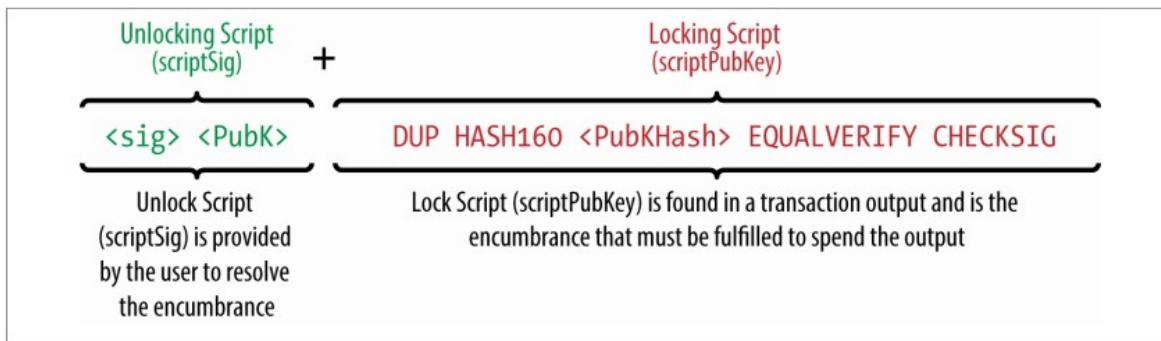
Każdy klient Bitcoin potwierdzi transakcję przez wykonanie funkcji blokowania i skryptów jednocześnie. Dla każdego wejścia do transakcji, oprogramowanie walidacji najpierw pobiera UTXO odwołanego do wejścia. To UTXO zawiera skrypt blokujący określający warunki wymagane, aby go wydać. Oprogramowanie walidacji podejmie wówczas skrypt odblokowujący zawarty w wyjściu, który próbuje wydać ten UTXO i wykonuje dwa skrypty.

W oryginalnym kliencie bitcoin, odblokowanie i blokowane skrypty były łączone i wykonywane po kolej. Ze względów bezpieczeństwa ta droga została zmieniona w 2010 roku, ze względu na lukę, która pozwalała uszkodzić skrypt odblokowujący i wcisnąć dane na stos i uszkodzić skrypt blokujący. W obecnej implementacji, skrypty są wykonywane oddziennie ze stosem przesyłanym pomiędzy dwoma egzekucjami, jak opisano dalej.

Po pierwsze, skrypt odblokowujący jest wykonywany przy użyciu silnika wykonania stosu. Jeżeli odblokowywany skrypt będzie wykonywany bez błędów (na przykład, że nie ma "wiszących" operacji), główny stos (nie stos alternatywny) jest kopowany i skrypt blokujący jest wykonany. Jeżeli wynik wykonania skryptu blokującego z danymi stosu skopiowany ze skryptu odblokowującego jest "prawdą", skryptowi odblokowania udało się spełnić warunki nałożone przez skrypty blokujące, a więc wejście jest ważną autoryzacją do wydania UTXO. Jeśli jednak wynik jest inny niż "prawdziwy" pozostaje ujawnione po wykonaniu skryptu kombinowanego, wejście jest nieważne, ponieważ nie udało się spełnić warunków wydatków umieszczonych na UTXO. Zauważ, że UTXO jest na stałe zapisane w łańcuchu blokowym, a więc jest zmienną wewnętrzną i jest niezależne od nieudanych prób wydania jego przez odniesienie w nowej transakcji. Tylko ważna transakcja, która poprawnie spełnia warunki wyników UTXO w UTXO oznaczona jako "wydana" i usunięta ze zbioru dostępnych (niewykorzystanych) UTXO.

Rysunek 5-1 jest przykładem odblokowywania i zablokowania skryptu dla najczęstszych rodzajów transakcji bitcoin (płatność do hash klucza publicznego), pokazujący skrypt kombinowany wynikający z konkatenacji do otwierania i zamykania skryptów przed walidacją skryptu.

Rysunek 5-1. Łączenie scriptSig i scriptPubKey do ewaluacji skryptu transakcji



Język skryptu

Język skryptu transakcji Bitcoin, zwany Script, jest podobny do języka Forth jako odwrotność polskiego zapisu na stosie opartej na realizacji języka. Jeśli to brzmi jak bełkot, to prawdopodobnie nie badaliście języków programowania lat 60-tych. Skrypt jest bardzo prostym językiem, który został zaprojektowany, aby być ograniczyć zakres i wykonywalny w zakresie hardware, chyba tak prostego jak we wbudowanym urządzeniu takim jak przenośny kalkulator. To wymaga minimalnego przetwarzania i nie może dokonać wielu fantazyjnych rzeczy, które nowoczesne języki programowania mogą zrobić. W przypadku programowalnych pieniędzy, to jest to funkcja zabezpieczeń celowych.

Język skryptowy Bitcoin nazywa się językiem na stosie, ponieważ wykorzystuje strukturę danych zwaną stosem. Stos jest bardzo prostą strukturą danych, która może być zwizualizowana jako stos kart. Stos umożliwia dwie operacje push i pop. Push dodaje element na wierzchu stosu. Pop usuwa górnego element ze stosu.

Język skryptowy wykonuje skrypt poprzez przetwarzanie każdego elementu, od lewej do prawej. Liczby (stałe dane) są wypychane na stos. Operatorzy push lub pop co najmniej jednego parametru ze stosu, oddziałują na nie i mogą oddziaływać na wynik na stosie.

Na przykład, OP_ADD zepchnie dwa elementy ze stosu, dodając je i pchnie wynikową sumę na stos.

Warunkowi operatorzy ocenią warunki, an, tworząc logiczny wynik PRAWDA lub FAŁSZ. Na przykład, OP_EQUAL popycha dwa elementy ze stosu i umieszcza PRAWDĘ (True-prawda- jest reprezentowana przez numer 1), jeżeli są one takie same lub FAŁSZ (reprezentowany przez zero), jeśli nie są one równe. Bitcoinowe skrypty transakcji zazwyczaj zawierają operatora warunkowego, dzięki czemu mogą one produkować prawdziwy wynik oznaczający, że transakcja jest ważna.

Na [Rysunku 5-2](#), skrypt 2 3 OP_ADD 5 OP_EQUAL demonstruje arytmetycznego operatora dodawania OP_ADD, dodając dwie liczby i oddając wynik na stos, a śledzonego przez warunkowego operatora OP_EQUAL, który sprawdza, że uzyskana suma jest równa 5. Dla zwięzłości, prefiks OP_ pominięto w przykładzie krok po kroku.

Poniżej znajduje się nieco bardziej skomplikowany scenariusz, który oblicza $2 + 7 - 3 + 1$. Zauważ, że gdy skrypt zawiera kilka operatorów w jednym rzędzie, stos pozwala na wyniki od jednego operatora które mają zostać rozpatrzone przez kolejnego operatora:

2 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL

Spróbuj samodzielnej walidacji poprzedniego skryptu za pomocą ołówka i papieru. Po zakończeniu wykonywania skryptu, powinien być pozostawiony z wartością TRUE na stosie.

Chociaż większość skryptów blokujące patrzy na adres bitcoin lub klucz publiczny i tym samym wymaga dowodu własności wydawanych funduszy, skrypt nie musi być aż tak skomplikowany. Dowolna kombinacja blokowania i odblokowywania skryptu, która prowadzi do wartości TRUE = PRAWDA jest ważna. Prostej arytmetycznej użyto jako przykład języka skryptowego ważnego jako skrypt blokujący, który może być stosowany do blokowania bloku wyjściowego transakcji. Użyj części arytmetycznej przykładowego skryptu jako skryptu blokującego:

3 OP_ADD 5 OP_EQUAL

który może być zaspokojony przez transakcję zawierającą wejście ze odblokowującym skryptem:

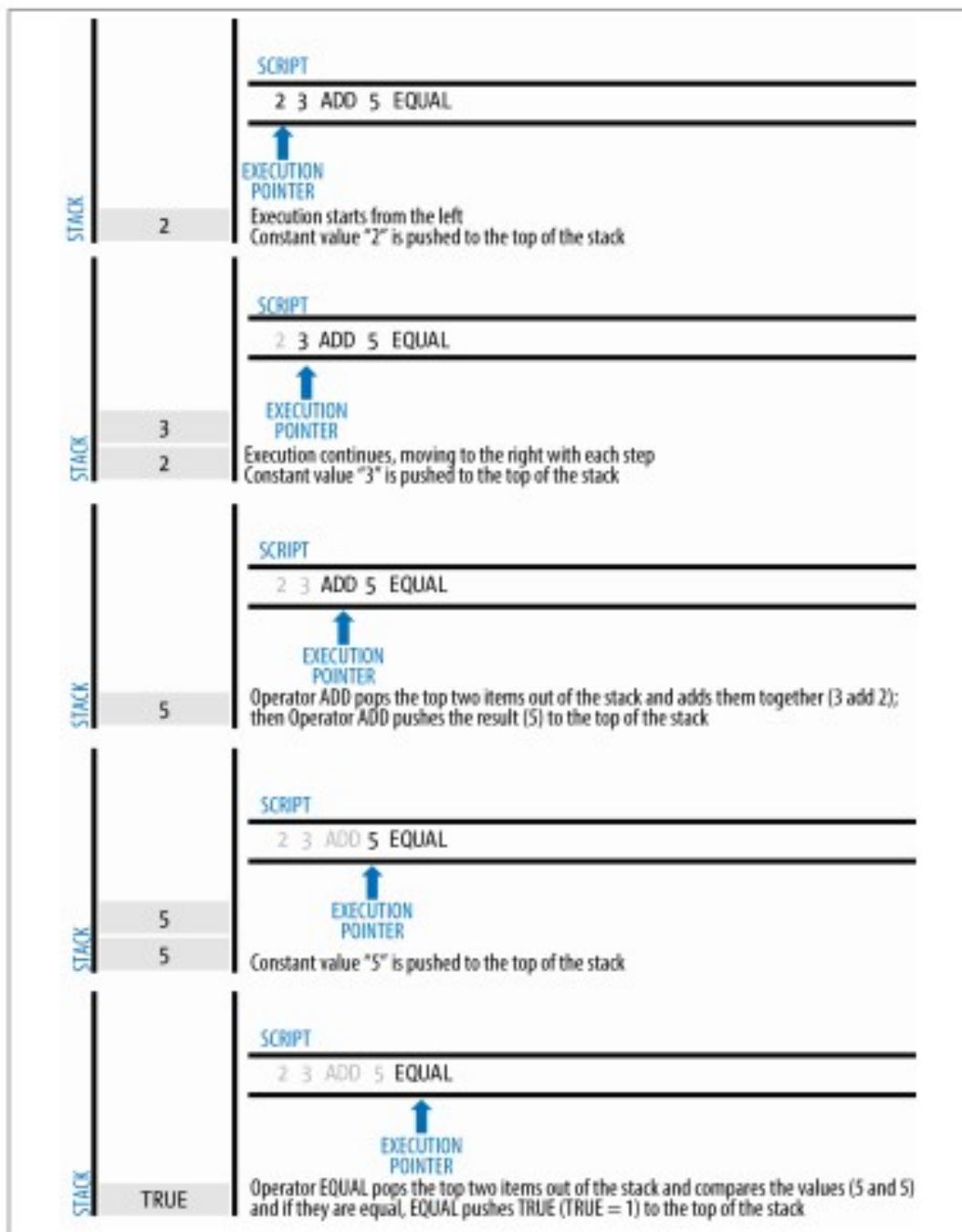
2

Oprogramowanie walidacji łączy blokowanie i uwolnienie skryptu i otrzymanym skryptem jest:

2 3 OP_ADD 5 OP_EQUAL

Jak widzieliśmy krok po kroku na przykładzie z rysunku 5 -2, gdy skrypt jest wykonywany, wynikiem jest OP_TRUE, dzięki czemu transakcja ważna. Nie tylko jest to poprawny skrypt blokujący blok wyjściowy transakcji, ale wynikające UTXO mogłyby zostać wykorzystane przez każdego z umiejętnościami arytmetycznymi, który wiedziałby, że liczba 2 spełnia skrypt.

Rysunek 5-2. walidacja skryptu Bitcoin na prostej matematyce:



Transakcje są ważne, jeżeli górny wynik na stosie jest PRAWDZIWY (TRUE) (oznaczony jako {0x01}), jakakolwiek inna niezerowa wartość lub to, że stos jest pusty po wykonaniu skryptu. Transakcje są nieważne, jeżeli górną wartość na stosie jest fałszem (zero-length pusta wartość, oznaczone jako {}) lub

jeśli wykonanie skryptu jest zatrzymane bezpośrednio przez operatora, tak jak OP_VERIFY, OP_RETURN lub warunkowego terminatora takiego jak OP_ENDIF. Zobacz [Załącznik A](#), aby poznać szczegóły.

Niekompletność Turinga

Język skryptowy Bitcoinowych transakcji zawiera wiele operatorów, ale jest celowo ograniczony jednym z ważnych sposobów, mianowicie - nie ma pętli lub kontroli przepływu innego niż warunkowego sterowania przepływem. Gwarantuje to, że język nie jest kompletnością Turinga, co oznacza, że skrypty mają ograniczoną złożoność i przewidywalny czas wykonania. Scenariusz nie jest językiem ogólnego przeznaczenia. Ograniczenia te zapewniają, że język nie może być użyty do utworzenia nieskończonej pętli lub innej formy "logicznej bomby", która może być osadzona w transakcji w sposób, który powoduje atak DoS przeciwko sieci bitcoin. Pamiętaj, że każda transakcja jest potwierdzona przez każdy pełny węzeł w sieci bitcoin. Ograniczenie języka zapobiega mechanizmowi walidacji transakcji być stosowanym jako luka w zabezpieczeniach.

Bezstatusowa weryfikacja

Język skryptowy transakcji Bitcoin jest bezstatusowcem to znaczy, że nie ma statusu przed wykonaniem skryptu, albo statusu zapisania po wykonaniu skryptu. W związku z tym, wszystkie informacje potrzebne do wykonania skryptu są zawarte w skrypcie. Skrypt będzie przewidującą wykonywany w ten sam sposób w każdym systemie. Jeśli system sprawdza skrypt, można mieć pewność, że każdy inny system w sieci bitcoin będzie również sprawdzać skrypt, co oznacza, że ważna transakcja jest ważna dla każdego i każdy o tym wie. Ta przewidywalność wyników jest zasadniczą zaletą systemu bitcoin.

Standardowe transakcje.

W pierwszych kilku latach rozwoju Bitcoin, twórcy wprowadzili pewne ograniczenia w rodzaju skryptów, które mogą być przetwarzane przez odpowiedniego klienta. Te ograniczenia są zakodowane w funkcji o nazwie `isStandard()`, która definiuje pięć typów "standardowych" operacji. Ograniczenia te mają charakter tymczasowy i mogą zostać zniesione do czasu w którym to czytasz. Do tego czasu, pięć standardowych typów skryptów transakcyjnych są jedynymi, które zostaną zaakceptowane przez odpowiedniego klienta i większość górników, którzy prowadzą kopanie dla odpowiedniego klienta. Chociaż jest możliwe utworzenie niestandardowej transakcji zawierającej skrypt, który nie jest jednym ze standardowych typów, musisz wtedy znaleźć górnika, który nie śledzi tych ograniczeń wydobywając tę transakcję w bloku.

Sprawdź kod źródłowy rdzennego klienta Bitcoin (implementacja referencyjna), aby zobaczyć co jest aktualnie dozwolone jako ważny skrypt transakcji. 5 standardowymi typami skryptów transakcyjnych są `pay-to-public-hash` (P2PKH), klucz publiczny, multi-podpis (ograniczony do 15 klawiszy), `pay-to-script-hash` (P2SH) oraz dane wyjściowe (`OP_RETURN`), które są opisane bardziej szczegółowo w dalszych częściach.

Pay-to-public-key-Hash (P2PKH)

Zdecydowana większość transakcji przetwarzanych w sieci bitcoin to działania P2PKH. Zawierają one skrypt blokujący, który obciąża blok wyjściowy z hash klucza publicznego, bardziej znanego jako adres bitcoin. Transakcje, które płacą adresy bitcoin zawierają skrypty P2PKH. Blok wyjściowy zablokowane

przez skrypt P2PKH może zostać odblokowane (zużyte) przedstawiając klucz publiczny i podpis cyfrowy utworzony przez odpowiedni klucz prywatny.

Na przykład, spójrzmy ponownie na wpłatę Alicji do Kawiarni Boba. Alicja dokonała zapłaty 0,015 Bitcoinów na adres bitcoin kawiarni. To blok wyjściowy transakcji miałyby skrypt blokujący w postaci:

OP_DUP OP_HASH160 <Cafe Public Key Hash> OP_EQUAL OP_CHECKSIG

Public Key Hash kawiarni jest równoważny z adresem bitcoin kawiarni, bez kodowania Base58Check. Większość aplikacji pokaże klawisz krzyżka publicznego w kodowaniu hexadec- IMAL i nie zna formatu adresu Bitcoin Base58Check, poza tym, że zaczyna się od "1". Powyższy skrypt blokujący może być zaspokojony przez skrypt odblokowania w postaci:

<Cafe Signature> <Cafe Public Key>

Oba skrypty razem będą tworzyć następujące połączeniu walidacji skryptu:

<Cafe Signature> <Cafe Public Key> OP_DUP OP_HASH160 <Cafe Public Key Hash> OP_EQUAL OP_CHECKSIG

Po wykonaniu ten kombinowany skrypt ewaluuje do PRAWDY- TRUE, wtedy i tylko wtedy, gdy skrypt odblokowania spełnia warunki określone przez skrypt blokujący. Innymi słowy, wynik będzie TRUE jeśli skrypt odblokowujący ma prawidłowy podpis z kluczem prywatnym w kawiarni, której odpowiada klawisz krzyżka publicznej ustawiony jako obciążenie.

Rys. 5-3 i 5-4 pokazują (w 2 częściach) krok po kroku wykonanie kombinowanego skryptu które udowodni, że to jest prawidłowa transakcja.

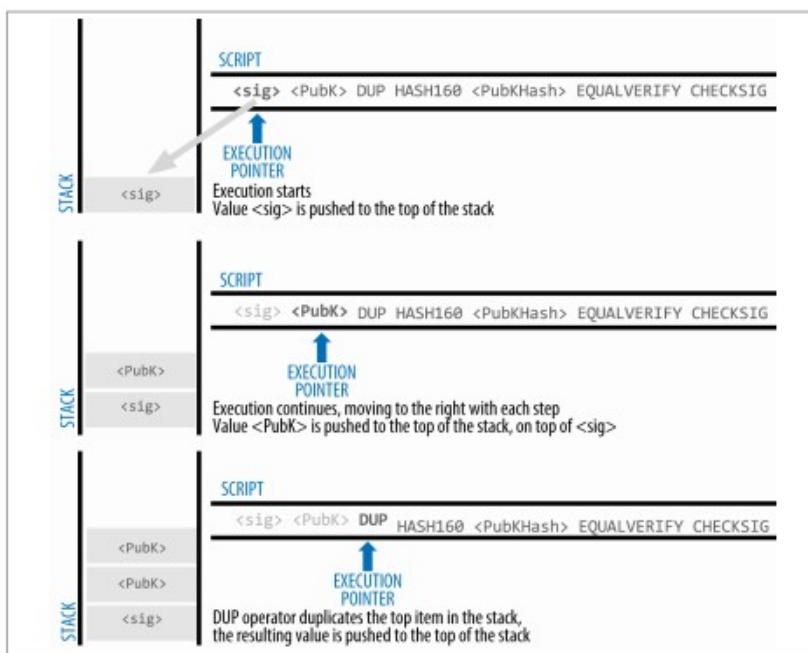


Figure 5-3. Evaluating a script for a P2PKH transaction (Part 1 of 2)

Pay-to-public-key

Pay-to-public-key jest prostszą formą płatności bitcoin niż pay-to-public-key-hash. Dzięki tej postaci skryptu sam klucz publiczny przechowuje się w scenariuszu blokującym, inaczej niż public-key-hash tak jak z P2PKH wcześniej, który jest znacznie krótszy. Pay-to-public-key-hash został wynaleziony przez Satoshi, aby uczynić adresy bitcoinowe krótszymi dla ułatwienia użytkowania. Pay-to-public-key-hash jest obecnie najczęściej spotykane w transakcjach coinbase, generowanych przez starsze oprogramowania górnictwa, które nie zostały zaktualizowane do korzystania z P2PKH.

Skrypt blokujący pay-to-public-key wygląda następująco:

<Public Key A> OP_CHECKSIG

Odpowiedni skrypt odblokowujący, który musi być podany, aby odblokować ten typ wyjścia jest prostym podpisem, jak coś takiego:

<podpis z kluczem prywatnym A>

Kombinowany skrypt, który jest zatwierdzony przez oprogramowanie zatwierdzania transakcji, to:

<podpis z kluczem prywatnym A> <klucz publiczny A> OP_CHECKSIG

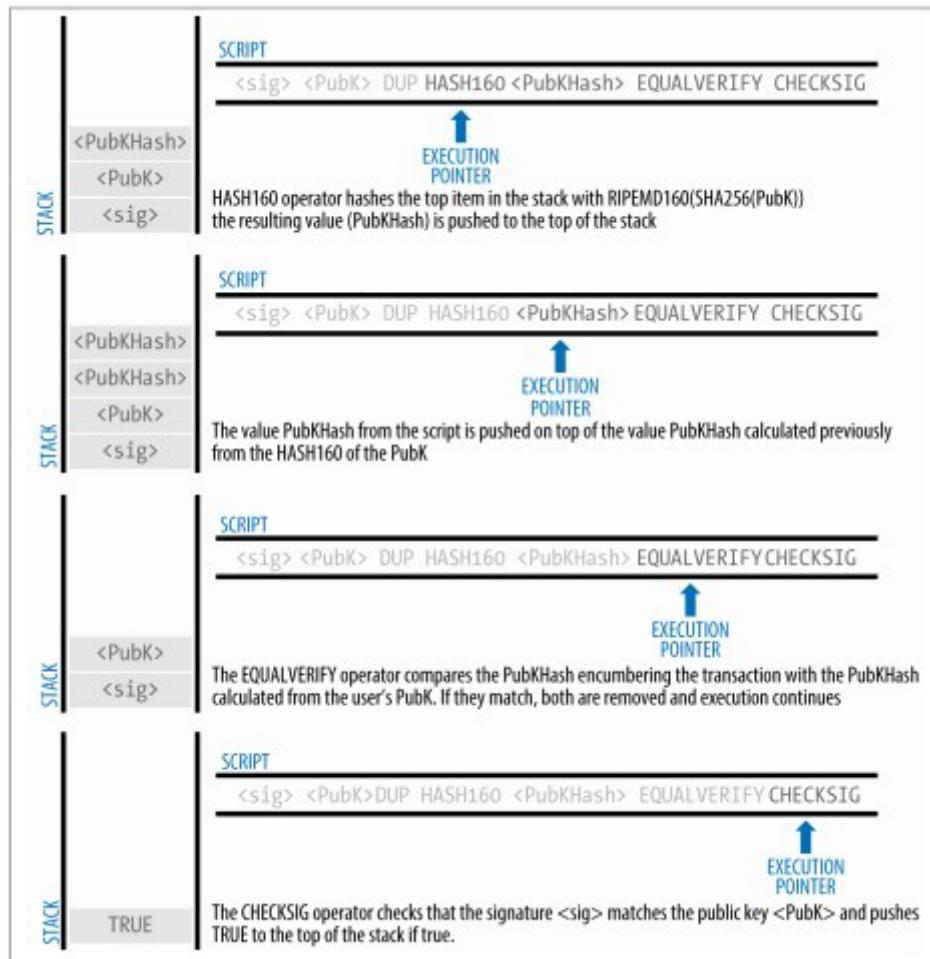


Figure 5-4. Evaluating a script for a P2PKH transaction (Part 2 of 2)

Ten skrypt jest prostym wywołaniem operatora CHECKSIG, który potwierdza podpis jako należący do prawidłowego klucza i zwraca prawdę - TRUE na stos.

Skrypty multi-signature

Skrypty multisignature odpowiadają warunkom, gdzie klucz publiczny N jest nagrany w skrypcie i przynajmniej M z nich musi dostarczyć podpisy do uwolnienia ograniczenia. Jest to też znane jako schemat M –z- N, gdzie N jest całkowitą liczbą przycisków, a M oznacza próg podpisów wymaganych do walidacji. Na przykład, 2-o-3. Multi-podpis jest wtedy, gdy trzy klucze publiczne są wymienione jako potencjalni sygnatariusze i co najmniej dwa z nich muszą być wykorzystane do tworzenie podpisów dla ważnej transakcji by wydać fundusze. W tym czasie, standardowe skrypty multipodpisowe są ograniczone do co najwyżej 15 wymienionych kluczy publicznych, dzięki czemu można zrobić cokolwiek od 1-w-1 do 15-of-15 multipodpisów lub dowolnej kombinacji w tym przedziale. Ograniczenie do 15 wymienionych klawiszy może być podniesione do czasu, gdy ta książka zostanie opublikowana, więc sprawdź funkcję isStandard (), aby zobaczyć, co jest obecnie akceptowane przez sieć.

Ogólna postać skryptu blokującego zachodzącego jako multipodpis w M-z-N spełnia warunek:

M <Public Key 1> <Public Key 2> ... <Klucza Publicznego N> N OP_CHECKMULTISIG

gdzie N jest całkowitą liczbą wymienionych publicznych kluczy i M jest progiem wymaganych podpisów aby opuścić blok wyjściowy. Skrypt blokujący ustawienia multi-podpisu z warunków 2-of-3 wygląda następująco:

2 <Klucz publiczny A> <Public Key B> <Klucza Publicznego C> 3 OP_CHECKMULTISIG

Poprzedni skrypt blokujący może być zadowolony ze skryptu odblokowującego zawierającego pary podpisów i klucze publiczne:

OP_0 <Podpis B> <Podpis C>

lub dowolną kombinację dwóch podpisów pochodzących od prywatnych kluczy odpowiadających trzem wymienionym kluczom publicznym



Przedrostek OP_0 jest wymagany ze względu na błąd we wdrożeniu CHECKMULTISIG, gdzie jeden z nich jest zdejmowany ze stosu zbyt wcześnie. Jest on ignorowany przez CHECKMULTISIG i jest prosto zastępowany.

Obydwa skrypty razem będą tworzyć skrypt walidacji kombinowanej:

OP_0 <Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C>
3 OP_CHECKMULTISIG

Po wykonaniu ten kombinowany skrypt będzie ewaluował od PRAWDY – TRUE wtedy i tylko wtedy gdy skrypt odblokowujący spełnia warunki określone przez skrypt blokujący. W tym przypadku, warunkiem jest czy skrypt odblokowujący ma prawidłowy podpis z dwóch kluczy prywatnych, który odpowiada dwóm z trzech kluczy publicznych określonych jako ograniczenia.

Blok danych wyjściowych (OP_RETURN)

Blok wyjściowy danych rozproszonych i czas utworzenia księgi głównej (OP_RETURN) Bitcoin, łańcuch bloków ma potencjalne użycie daleko poza płatnościami. Wielu programistów próbowało używać języka skryptowego transakcji w celu skorzystania z bezpieczeństwa i odporności systemu dla aplikacji, takich jak cyfrowe usługi notarialne, świadectwa udziałowe oraz inteligentne umowy. Wcześniej próby wykorzystania języka skryptowego Bitcoin dla celów związanych z tworzeniem wyjść transakcyjnych, które tworzą zapis danych na łańcuchu blokowym; na przykład aby zapisać cyfrowy odcisk palca z pliku w taki sposób, żeby każdy mógł ustalić dowód funkcjonowania tego pliku na podstawie specyficznych danych odnoszących się do tej transakcji.

Użycie bloku łańcuchowego bitcoinów do przechowywania danych niepowiązanych z płatnościami jest przedmiotem kontrowersji. Wielu programistów uważa to za niedozwolone i chce to znieść. Inni postrzegają to jako pokaz potężnych możliwości technologii łańcucha blokowego i chcą, aby wspierać takie rozwiązania. Ci, którzy sprzeciwiają się włączeniu danych niepłatniczych twierdzą, że powoduje ona "uwiązanie łańcucha" obciążając tych, którzy wykorzystują w pełni węzły bitcoin troszcząc się o koszt

pamięci dyskowej dla danych, o które łańcuch blokowy nie jest w stanie się troszczyć. Co więcej, takie transakcje tworzą UTXO, które nie mogą być wykorzystane z adresu docelowego Bitcoin jako wolne formy pola 20-bajtowego. Ponieważ adres jest używany do danych, które nie odpowiadają kluczowi prywatnemu, a otrzymane UTXO nigdy nie może być wydane; następuje fałszywa płatność.

Taka praktyka powoduje, że wielkość pamięci UTXO wzrasta i te transakcje, które nie mogą być wykorzystane w związku z tym nigdy nie są usuwane, zmuszając węzły Bitcoin do zatrzymania ich na zawsze w pamięci RAM, co jest znacznie bardziej kosztowne.

W wersji 0.9 klienta Bitcoin Core osiągnięto kompromis w sprawie wprowadzenia operatora OP_RETURN. OP_RETURN umożliwia programistom dodanie 40 bajtów danych niepłatniczych do bloku wyjściowego transakcji. Jednakże, w przeciwieństwie do stosowania "fałszywego" UTXO operator OP_RETURN tworzy jawnie potwierdza niewydawalność wyjścia, które nie musi być przechowywane w zestawie UTXO. Wyjścia OP_RETURN są zapisywane na łańcuchach blokowych, ale zużywają one miejsce na dysku i przyczyniają się do wzrostu wielkości łańcucha, ale nie są one przechowywane w zbiorze UTXO i dlatego nie uwiązuje zbiornika pamięci UTXO i nie obciążają pełnych węzłów bardziej kosztownej pamięci RAM.

Skrypty OP_RETURN wyglądają następująco:

OP_RETURN <data>

Część danych jest ograniczona do 40 bajtów i najczęściej reprezentuje hash, tak jak blok wyjściowy z algorytmu SHA256 (32 bajtów). Wiele aplikacji umieszcza prefiks przed danymi, aby pomóc w identyfikacji aplikacji. Na przykład, dowód istnienia notarialnej usługi cyfrowej wykorzystuje prefiks 8-bajtowy "DOCPROOF", który jest zakodowany jako ASCII 44f4350524f4f46 w systemie szesnastkowym.

Należy pamiętać, że nie ma "odblokowanego skryptu", który odpowiada OP_RETURN, które mogłyby być wykorzystane do "wyjścia" przez blok wyjściowy OP_RETURN. Sensem OP_RE z kolei jest, że nie można wydać pieniędzy zablokowanych w tym wyjściu, a zatem nie musi być zatrzymany w UTXO jako potencjalnie wydawalne -OP_RETURN. jeśli jest oczywiście niewydawalne. OP_RETURN jest zwykle jest blok wyjściowy m z wartością zerową, ponieważ każdy bitcoin dodany do każdego takiego wyjścia jest zazwyczaj skazany na stratę. Jeśli OP_RETURN jest napotkiwane przez oprogramowanie skryptu zatwierdzającego, jego natychmiastowym rezultatem jest zatrzymanie się, wykonanie walidacji skryptu i oznaczenie natychmiastwo transakcji jako nieważnej.

Standardowa transakcja (zgodna z weryfikacjami isStandard()) może mieć tylko jeden blok wyjściowych OP_RETURN. Tak więc, jeśli przypadkowo zaznaczyłeś blok wyjściowy OP_RETURN jako wejście w transakcji, ta transakcja jest nieważna.

Standardowa transakcji (taka, która jest zgodna z isStandard () checks) może mieć tylko jedno blok wyjściowy OP_RETURN. Jednak pojedyncze blok wyjściowy OP_RETURN mogą być łączone w transakcje z blokiem wyjściowym innego typu.

Pay-by-Script-Hash (P2SH)

Pay-to-script-hash (P2SH) zostało wprowadzone w 2012 roku jako potężny nowy typ transakcji, która znacznie upraszcza wykorzystanie skryptów transakcji złożonych. Aby wyjaśnić istotę P2SH, spójrzmy na praktyczny przykład.

W Rozdziale 1 wprowadziliśmy Mahometa, importera elektroniki z siedzibą w Dubaju. Firma Mohammeda wykorzystuje funkcję multi-podpis wyposażoną w znacznym stopniu ze swoich kont firmowych. Skrypty Multi-podpisu są jednym z najczęstszych zastosowań zaawansowanych funkcji skryptowych Bitcoin i są bardzo potężnym narzędziem. Firma Mohammeda wykorzystuje skrypt multi-podpisu dla płatności wszystkich klientów, znanych w ujęciu księgowym jako "należności", lub AR. Wraz z systemem multi-podpisu, wszelkie płatności dokonywane przez klientów są zablokowane w taki sposób, że wymagają one co najmniej dwóch podpisów, aby wypuścić transakcję - od Mahometa i jednego z jego wspólników lub od prawnika, który ma klucz zapasowy. Schemat multi-podpisu oferuje kontrole w zakresie ładu korporacyjnego i chroni przed kradzieżą, defraudacją lub stratami.

Powstały skrypt jest dość długi i wygląda następująco:

```
2 <Mahomet's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3 Public Key> < Attorney  
Public Key> 5 OP_CHECKMULTISIG
```

Chociaż skrypty multi-podpisu są potężnym narzędziem, ale są kłopotliwe w użyciu. Biorąc pod uwagę powyższy skrypt, Mohammed będzie musiał porozumieć się z tego skryptu z każdym klientem przed dokonaniem płatności. Każdy klient będzie musiał użyć specjalnego oprogramowania Bitcoin Portfel z możliwością tworzenia skryptów transakcji niestandardowych i będzie musiał zrozumieć, w jaki sposób utworzyć transakcję przy użyciu skryptów. Ponadto, uzyskane transakcje będą około pięć razy dłuższe niż zwykłe transakcje płatnicze, ponieważ ten skrypt zawiera bardzo długie klucze publiczne. Ciężar tej bardzo dużej transakcji byłby ponoszone przez klienta w formie opłat. Wreszcie, skrypt tak dużej transakcji jak ta powinien być przeprowadzony w UTXO i ustawiony w pamięci RAM w każdym pełnym węźle, dopóki nie zostanie wyczerpany. Wszystkie te kwestie sprawiają, że używanie skomplikowanych skryptów wyjściowych jest trudne w praktyce.

Pay-to-script-hash (P2SH) został opracowany w celu rozwiązania tych trudności praktycznych oraz do wykorzystania złożonych skryptów tak, by były proste jak płatności na adres bitcoin. Przy płatności P2SH kompleks blokujący skrypt jest zastąpiony cyfrowym odciskiem palca, kombinacji kryptograficznej. Gdy transakcja próbuje być dokonana UTXO jest przedstawiane później i musi zawierać skrypt, który pasuje do kombinacji, w odniesieniu do scenariusza odblokowującego. W prostych słowach, P2SH oznacza "Zapłacić do skryptu pasujące do tego kombinacji, skrypt, zostanie przedstawiony później, kiedy to blok wyjściowy będzie opuszczony".

W transakcjach P2SH skrypt blokujący, który jest zastąpiony przez hash jest opisywany jako skrypt wykupujący, ponieważ jest przedstawiony w systemie w czasie wykupu, a nie jako skrypt blokujący. Tabela 5-4 przedstawia skrypt bez P2SH a Tabela 5-5 przedstawia ten sam skrypt zakodowany z P2SH.

Tabela 5-4 - skrypt bez P2SH

Skrypt blokujący	2	PubKey1	PubKey2	PubKey3	PubKey4	PubKey5	5
Skrypt odblokowujący		OP_CHECKMULTISIG					
Sig1		Sig2					

Tabela 5-5 – skrypt z P2SH

Skrypt wykorzystany	PubKey1	PubKey2	PubKey3	PubKey4	PubKey5	5
	OP_CHECKMULTISIG					
Skrypt blokujący	OP_HASH160 <20-byte hash of redeem script>	OP_EQUIV				
Skrypt odblokowujący	Sig1	Sig2	redeem script			

Jak widać w tabelach, z P2SH skrypt jest kompleksowy tak że wyszczególnia warunki do opuszczenia bloku wyjściowego (umarzany skrypt) nie jest przedstawiony w skrypcie blokującym. Zamiast tego, z nich tylko hash jest w skrypcie blokującym, a wykorzystany skrypt sam jest przedstawiony później jako część skryptu odblokowania, gdy blok wyjściowy jest opuszczany. To przesuwa ciężar opłat i kompleksowość z nadawcy na odbiorcę (donatora) transakcji.

Spójrzmy na spółkę Mahometa, złożony skrypt multi-sig i wyniku skryptu P2SH. Najpierw na skrypt multi-podpisu, którego używa firma Mahometa dla wszystkich przychodzących płatności od klientów: 2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3 Public Key> <Attorney Public Key> 5 OP_CHECKMULTISIG

Jeśli zamienne są zastępowane przez rzeczywiste klucze publiczne (tutaj pokazane jako liczby 520-bitowe, poczynając od numeru 04) możesz dostrzec, że ten skrypt staje się bardzo długi:

2

```
04C16B8698A9ABF84250A7C3EA7EE-
DEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395D7E6C6984D83F1F50C900A24DD47F
569FD4193AF5DE762C58704A2192968D8655D6A935BEAF2CA23E3FB87A3495E7AF308EDF08DAC3C1
FCBFC2C75B4B0F4D0B1B70CD2423657738C0C2B1D5CE65C97D78D0E34224858008E8B49047E63248
B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812F393DA7D4420D7E1A9162F0
279CFC10F1E8E8F3020DECDBC3C0DD389D99779650421D65CBD7149B255382ED7F78E946580657EE
6FDA162A187543A9D85BAAA93A4AB3A8F044DA-
DA618D087227440645ABE8A35DA8C5B73997AD343BE5C2AFD94A5043752580AFA1EC-
ED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF518C022DD618DA7
74D207D137AAB59E0B000EB7ED238F4D800 5 OP_CHECKMULTISIG
```

Cały ten skrypt może być natomiast reprezentowany przez 20-bajtowy skrót kryptograficzny, najpierw zastosowując algorytm hashujący SHA256, a następnie zastosowując na wyniku algorytm RIPEMD160. Hash 20-bajtowy z poprzedniego skryptu to:

54c557e07dde5bb6cb791c7a540e0a4796f5e97e

Transakcja P2SH blokuje blok wyjściowy do tego hash w przeciwieństwie do dłuższego skryptu, za pomocą skryptu blokującego

`OP_HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e OP_EQUAL`

który, jak widać, jest o wiele krótszy. Zamiast "zapłacić tym pięciokluczowym skryptem multi-podpisu," transakcja P2SH "płaci do skryptu z tego hasha (skrótu)." Klient dokonując płatności na rzecz firmy Mahometa musi dzięki temu zatrzymać tylko ten znacznie krótszy skrypt blokujący w jego płatności. Kiedy Mohammed chce wydać ten UTXO, musi przedstawić oryginalny wydany skrypt (ten, którego skrót-hash-jest blokadą dla UTXO) oraz podpisy niezbędne, aby go odblokować, na przykład:

`<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG>`

2 skrypty składają się z dwóch etapów. Po pierwsze, odkupiony skrypt jest sprawdzany przez skrypt blokujący, aby upewnić się, że hash pasuje.

`<2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG> OP_HASH160 <redeem scriptHash> OP_EQUAL`

Jeśli hashe odkupionego skryptu pasują, skrypt odblokowania jest wykonywany we własnym zakresie, aby odblokować skrypt:

`<Sig1> <Sig2> 2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG`

[Adresy Pay-to-script-hash](#)

Kolejnym ważnym elementem funkcji P2SH jest możliwość zakodowania hasha –skrótu- skryptu jako adres, zgodnie z definicją w BIP0013. Adresy P2SH są kodowane Base58Check przez 20- bajtowy hash skryptu, podobnie jak adresy Bitcoin są kodowane przez Base58Check i odkodowywane hashem 20- bajtowego klucza publicznego. Adresy P2SH używają wersji prefiksu "5", którego rezultatem w odkodowaniu adresu przez Base58Check jest rozpoczęwanie od "3". Na przykład, skrypt Mahometa, hashowany i Base58Check odkodowany jako adres P2SH staje się : 39RF6JqA BiHdYHkfChV6USGMe6Nsr66Gzw. Teraz, Mohammed może dać ten "adres" do jego klientów, a oni mogą używać niemal każdego portfela Bitcoin, aby dokonać tej prostej płatności, tak jakby był to adres Bitcoin. Prefix 3 daje im wskazówkę, że jest to specjalny rodzaj adresu, odpowiadający skryptowi zamiast kluczowi publicznemu, ale poza tym to działa dokładnie w taki sam sposób jak płatności na adres bitcoin.

Adresy P2SH ukrywają wszystkie złożoności, tak aby osoba dokonująca płatności nie widziała skryptu.

[Korzyści z pay-to-script-hash](#)

Funkcja Pay-to-script-hash oferuje następujące korzyści w porównaniu do bezpośredniego użycia skomplikowanych skryptów w wyjściach blokujących:

Złożone skrypty zostają zastąpione krótszymi w postaci odcisków palców w wyjściu transakcji, co czyni transakcję mniejszą.

- Skrypty mogą być kodowane jako adres, więc nadawca i portfel nadawcy nie wymagają skomplikowanej inżynierii, by wdrożyć P2SH. P2SH przesuwa ciężar konstruowania skryptu na odbiorcę, a nie nadawcę.
- P2SH przenosi obciążenie w postaci pamięci danych długiego skryptu z wyjścia (który jest zawarty w UXTO, a tym samym na jego pamięć) do wejścia (przechowywany tylko w łańcuchu blokowym).
- P2SH przesuwa ciężar w przechowywaniu danych długiego skryptu z obecnego czasu (płatności) do czasu przyszłego (gdy jest wydany).
- P2SH przesuwa koszt opłaty transakcyjnej za długi skrypt od nadawcy do odbiorcy, który musi posiadać skrypt odkupujący, aby go wydać.

Realizacja (umarzanie) skryptu i walidacja isStandard

Przed wersją 0.9.2 klienta Bitcoin Core pay-to-script-hash zostało ograniczone do standardowych typów skryptów Bitcoinowych transakcji, przez funkcję `isStandard()`. Oznacza to, że odkupowany skrypt przedstawiony w transakcji wydatków może być tylko jednym ze standardowych typów: P2PK, P2PKH, lub wielo-podpisowym, z wyłączeniem `OP_RETURN` i samego P2SH.

Począwszy od wersji 0.9.2 klienta Bitcoin Core transakcje P2SH mogą zawierać dowolny poprawny skrypt, dzięki czemu czynią P2SH znacznie bardziej elastycznym i pozwalają na eksperymentowanie z wieloma nowatorskimi i złożonymi rodzajami transakcji.

Zauważ, że nie jesteś w stanie umieścić P2SH wewnątrz skryptu odkupionego P2SH, ponieważ specyfikacja P2SH nie jest rekurencyjna. Nadal nie możesz korzystać z `OP_RETURN` w skrypcie odkupionym, bo `OP_RETURN` nie mogą być z definicji umarzane.

Należy zauważyć, że skrypt wykupu nie jest prezentowany w sieci, dopóki nie ppodejmowana jest próba wyjścia P2SH; jeśli zablokowałeś blok wyjściowy z hashem z nieprawidłowej transakcji to będzie przetwarzany niezależnie. Jednakże nie będzie mógł ich wydać, ponieważ wydatki transakcyjne, które skrypt wykupu zawiera, nie zostaną zaakceptowane, ponieważ skrypt jest nieprawidłowy. To kreuje ryzyko, bo możesz zablokować Bitcoin w P2SH, żeby nie mógł być wydany później. Sieć będzie akceptować obciążanie P2SH nawet jeśli odpowiada ona na nieprawidłowy skrypt wykupu, ponieważ hash skryptu nie daje żadnych wskazówek o skrypcie jakim reprezentuje.



Skrypty blokujące P2SH zawierają hash skryptu odkupienia, który nie daje żadnych wskazówek co do treści samego skryptu. Transakcje P2SH zostaną uznane za ważne i akceptowane, nawet jeśli uznają skrypt odkupienia za nieprawidłowy. Może to spowodować przypadkowe zablokowanie Bitcoin w taki sposób, w taki sposób, że nie będzie można go później wykorzystać.

Rozdział 6

Sieć Bitcoin

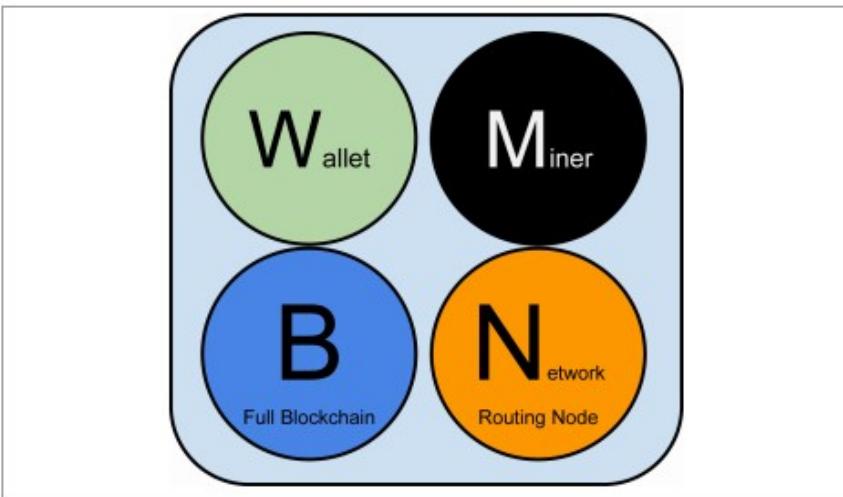
Architektura sieci Peer- to – Peer

Architektura Bitcoin jest skonstruowana jak architektura sieci peer-to-peer na szczeblu Internetu. Termin Peer-to-peer lub P2P oznacza, że komputery, które uczestniczą w sieci są rówieśnikami, że są one równe, że nie ma żadnych "specjalnych" węzłów oraz, że wszystkie węzły dzielą ciężar dostarczania usług sieciowych. Węzły sieciowe są ze sobą połączone w "płaskiej" topologii. Nie ma żadnego serwera, nie ma skonsolidowanej usługi, ani hierarchii w sieci. Węzły w sieci peer-to-peer dostarczają i konsumują usługi w tym samym czasie działając ze wzajemnością jako zachętą do uczestnictwa. Peer-to-peer są z natury odporne, zdecentralizowane i otwarte. Wybitnym przykładem architektury sieci P2P był wczesny Internet, gdzie węzły sieci IP były równe. Dzisiejsza architektura internetowa jest bardziej hierarchiczna, ale Protokół Internetowy wciąż zachowuje swoją istotę płaskiej topologii. Poza Bitcoin, największym i najbardziej skutecznym stosowaniem technologii P2P jest udostępnianie plików używając Napstera jako pionierem i BitTorrenta jako najnowszej ewolucji architektury.

Architektura sieci Bitcoin P2P jest czymś więcej niż tylko wyborem topologii. Bitcoin jest zaprojektowany jako cyfrowy system pieniężny peer-to-peer, a architektura sieci jest odbiciem i fundamentem tej charakterystyki podstawowej. Decentralizacja kontroli jest jedną z zasad projektowania rdzenia i może być osiągnięta i utrzymywana przez spłaszczanie, decentralizując porozumienie sieci P2P. Termin "sieć Bitcoin" odnosi się do zbioru węzłów obsługujących protokół P2P Bitcoin. W nawiązaniu do bitcoinowskiego protokołu P2P, istnieją inne protokoły, takie jak Stratum, które są stosowane dla górnictwa i lekkich lub ruchomych portfeli. Te dodatkowe protokoły są dostarczane przez bramy routingowe serwerów, które mają dostęp do sieci Bitcoin przy użyciu protokołu P2P Bitcoin, a następnie rozciągają tą sieć na węzły uruchamiane innymi protokołami. Na przykład serwery Stratum łączą węzły górnicze Stratum poprzez protokół Stratum z główną bitcoinową siecią i mostem protokołu Stratum do protokołu P2P Bitcoin. Używany termin "rozbudowana sieć Bitcoin" by opisać całą sieć, która obejmuje protokoły: P2P Bitcoin, protokoły zbiorników górniczych, protokół Stratum i wszelkie inne pokrewne protokoły łączące elementy systemu bitcoin.

Węzły - rodzaje i role

Chociaż węzły w bitcoinowskiej sieci P2P są równe, mogą one przyjmować różne role w zależności od ich funkcji nośnej. Węzeł Bitcoin jest zbiorem funkcji: przekierowywania, bazy danych łańcucha blokowego, wydobycia i portfela. Pełny węzeł z wszystkimi czterema funkcjami jest pokazany na [Rysunku 6-1](#)



Rysunek 6-1. Węzeł sieci Bitcoin złożony ze wszystkich czterech funkcji: portfela, górnika, pełnych łańcuchów blokowych i sieci trasującej

Każdy z węzłów zawiera funkcje routingową do partycypowania w sieci i może również zawierać inne funkcjonalności. Wszystkie sprawne węzły, propagatory transakcji i bloki, odkrywają i utrzymują połączenia z rówieśnikami. W przykładzie z pełnym węzłem na Rysunku 6-1, funkcja routingu jest wskazywana przez pomarańczowe kółko o nazwie "Network Routing Node".

Niektóre węzły, zwane pełnymi węzłami, utrzymują również kompletne i aktualne kopie łańcucha . Pełne węzły mogą autonomicznie i autorytatywnie sprawdzać każdą transakcję bez odniesienia zewnętrznego. Niektóre węzły utrzymują tylko podzbior łańcuchów i weryfikują transakcje przy użyciu metody zwanej uproszczoną weryfikacją płatności lub SPV. Węzły te są znane jako SPV lub lekkie węzły. W pełnym węźle przykładowym na rysunku, funkcja bazy danych łańcucha pełnego bloków jest oznaczona niebieskim kołem o nazwie "Full Blockchain". Na [Rysunku 6-3](#), węzły SPV są rysowane bez niebieskiego koła, pokazując, że nie mają one pełnej kopii łańcucha bloków.

Węzły górnicze konkurują ze sobą, aby stworzyć nowe bloki poprzez uruchomienie specjalistycznego sprzętu, aby rozwiązać algorytm proof-of-work. Niektóre węzły wydobywcze są również pełnymi węzłami, utrzymującymi pełną kopię łańcucha blokowego, podczas gdy inne są lekkie i uczestniczą w kopaniu zbiornika oraz w zależności od serwera zbiornika w utrzymaniu pełnego węzła.

Funkcja wydobycie jest pokazana w pełnym węźle jako czarny okrąg o nazwie "Górnik". Portfele użytkownika może być częścią pełnego węzła, jak to zwykle bywa z pulpitem klientów bitcoin. Coraz częściej wiele portfeli użytkowników, zwłaszcza tych działających na ograniczonych zasobach urządzeń, takich jak smartfony, to węzły SPV.

Funkcja portfela użytkownika została przedstawiona na rysunku 6-1 w zielonym kole o nazwie "Portfel". Oprócz głównych typów węzłów na protokole Bitcoin P2P, istnieją serwery i węzły uruchomione przez inne protokoły, takie jak wyspecjalizowane protokoły kopiące zbiornik i lekkie protokoły dostępne dla klienta. Rysunek 6-2 pokazuje najczęstsze typy węzłów w rozszerzonej sieci bitcoin.

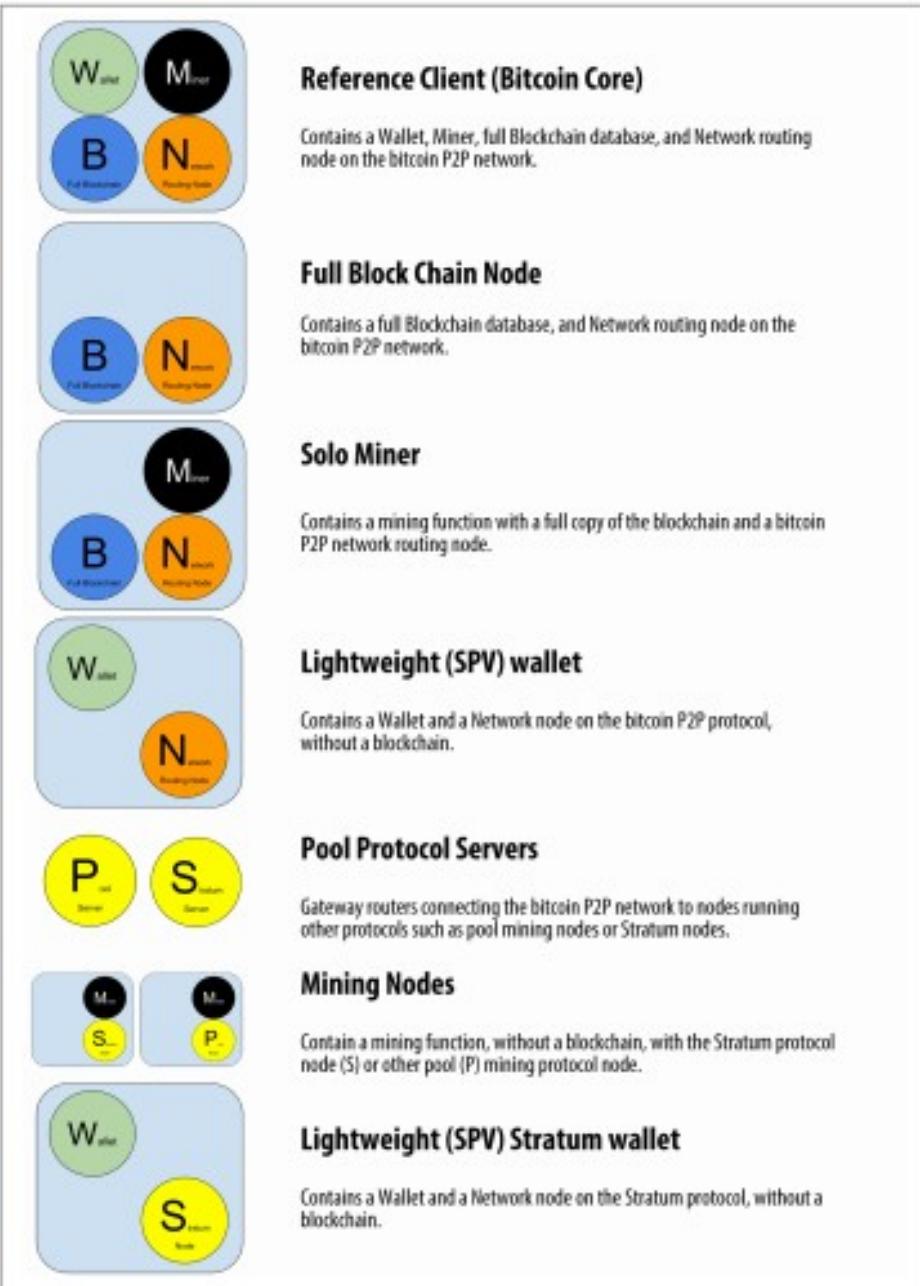
Rozszerzenie sieci

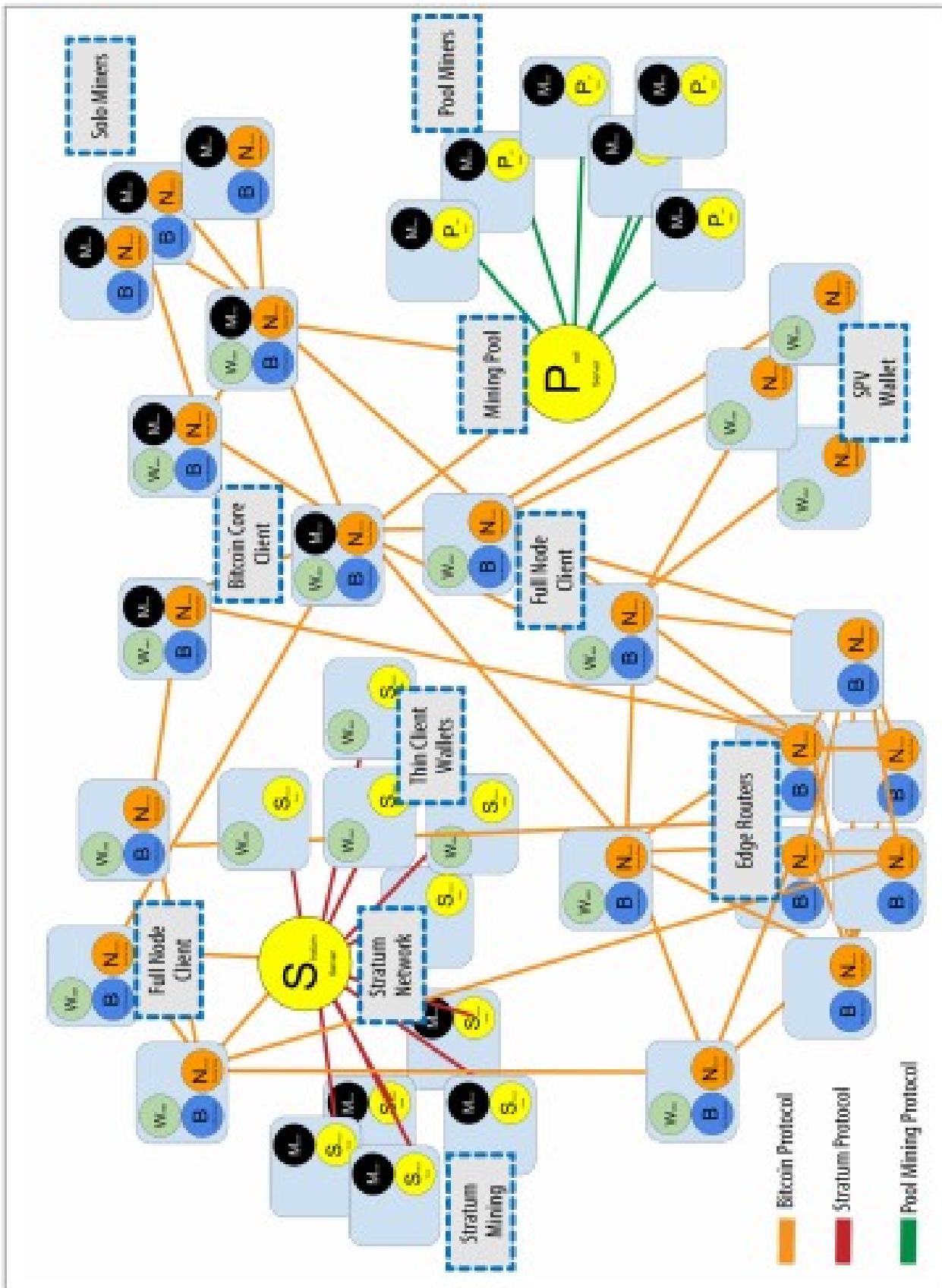
Główna sieć Bitcoin, prowadząca Protokół P2P Bitcoin, składa się z między 7000 a 10 000 węzłów słuchających działających w różnych wersjach klienta referencyjnego Bitcoin (Bitcoin Core) oraz kilkuset

węzłów uruchomionych przez różne inne implementacje protokołu P2P bitcoin, takie jak BitcoinJ, Libbitcoin i btcd. Niewielki odsetek węzłów sieci bitcoin P2P są również węzłami wydobywającymi, konkurując w procesie wydobywczym, weryfikacji transakcji oraz tworzenia nowych bloków. Różnorodny interfejs dużych firm z sieci bitcoin uruchamia klientom pełen węzeł oparty na kliencie Bitcoin Core z pełnymi kopiami łańcucha bloków i węzłem sieci, ale bez funkcji wydobycia lub portfela. Węzły te działają jako routery brzegowe sieci, umożliwiając różne inne usługi (wymiany, portfele, odkrywcy blokowi, przetwarzanie płatności kupców) budowę na wierzchu.

Rozszerzona sieć Bitcoin obejmuje sieć działającą na protokole P2P Bitcoin, opisanym wcześniej, tak samo jak węzły działające na specjalistycznych protokołach. Przydzielone do głównej sieci P2P Bitcoin są liczby zbiorników serwerów i bramek protokolarnych, które łączą węzły uruchamiając inne protokoły. Te inne protokoły węzłów są w większości zbiornikowymi węzłami wydobywczymi (patrz rozdział 8) i lekkimi portfelami klientów, które nie posiadają pełnej kopii łańcucha bloków.

Rysunek 6-3 przedstawia rozbudowaną sieć Bitcoin z różnymi typami węzłów, bram serwerów, routerów brzegowych i portfelami klientów i różnymi protokołami, które są używane do łączenia się ze sobą.





Wykrywanie sieci:

Gdy nowy węzeł uruchamia się, musi odkryć inne węzły w sieci Bitcoin w celu dołączenia do nich. Aby rozpocząć ten proces, nowy węzeł musi wykryć się co najmniej jeden istniejący węzeł sieci i połączyć się z nim. Położenie geograficzne innych węzłów nie ma znaczenia; topologia sieci Bitcoin nie jest zdefiniowane geograficznie. Dlatego wszelkie istniejące węzły Bitcoin mogą być wybrane losowo.

Aby połączyć się znanym peerem, węzły ustanawiają połączenie TCP, zazwyczaj do portu 8333 (port ten jest ogólnie znany jako jeden z używanych przez Bitcoin) lub alternatywnego portu, jeśli takowy jest istniejący. Po ustanowieniu połączenia, węzeł rozpocznie "handshake" (patrz Rysunek 6-4) przez przesłanie wersji wiadomości, która zawiera podstawowe informacje identyfikujące, w tym:

PROTOCOL_VERSION

Stała, która określa wersję bitcoinowskiego protokołu PTP, którą mówi Klient (np. 70002)

nLocalServices

Lista usług lokalnych obsługiwanych przez węzeł, obecnie tylko NODE_NETWORK

nTime

obecny czas

addrYou

Adres IP zdalnego węzła, jaki wynika z tego węzła

addrMe T

adres IP węzła lokalnego, jako wykrytego przez węzeł lokalny

subver

Subwersja pokazująca typ oprogramowania uruchomionego na tym węźle (na przykład "/ Satoshi: 0.9.2.1/")+

BestHeight

Wysokość bloku w tym węźle łańcucha blokowego

Zobacz GitHub na przykładzie wersji wiadomości sieciowej).

Węzeł Peer odpowie jako „verack” by uznać i nawiązać połączenie, oraz ewentualnie wysłać własną wersję wiadomości, jeśli chce odwzajemnić połączenie i podłączyć się jako Peer.

W jaki sposób nowy węzeł znajduje Peerów? Chociaż nie istnieją żadne specjalne węzły w Bitcoin, istnieją pewne długo działające, stabilne węzły, które są wymienione jako węzły nasiennne. Chociaż nowy węzeł nie musi łączyć się z węzłami nasiennymi, można ich używać do szybkiego wykrywania innych węzłów w sieci. W kliencie Bitcoin Core możliwość wykorzystania węzłów nasiennych jest kontrolowane przez opcję przełączającą –dnsseed, która jest ustawiona na 1, w celu używania węzłów nasiennych jako domyślne. Alternatywnie, węzeł ładujący, który nic nie wie o sieci musi posiadać adres IP co najmniej jednego węzła

bitcoin, po którym może nawiązać połączenia poprzez dalsze wprowadzanie. Argument wiersza polecenia -seednode może być używany do łączenia się z jednym węzłem tylko dla wprowadzenia używając go jako materiału siewnego DNS. Po użyciu węzła do utworzenia wstępnu, klient rozłączy się z nim i zacznie korzystać z nowo odkrytych peerów.

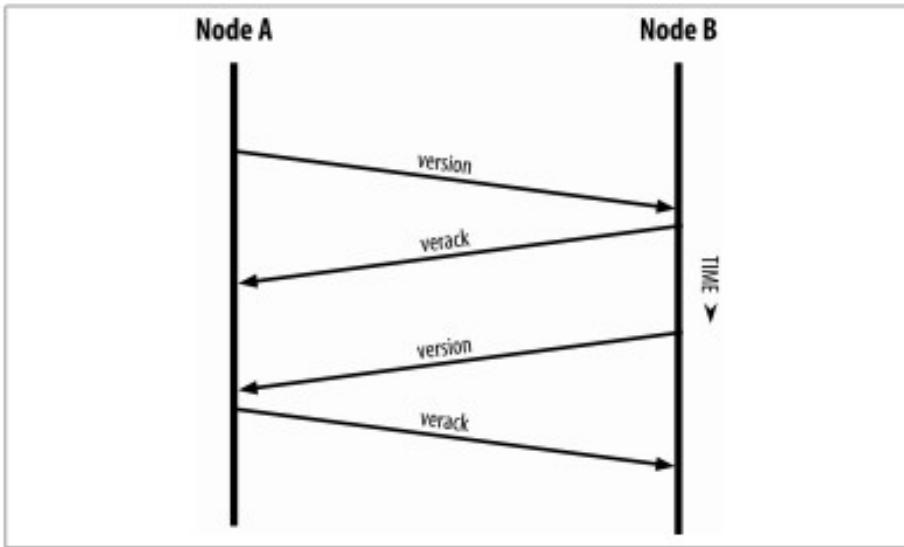
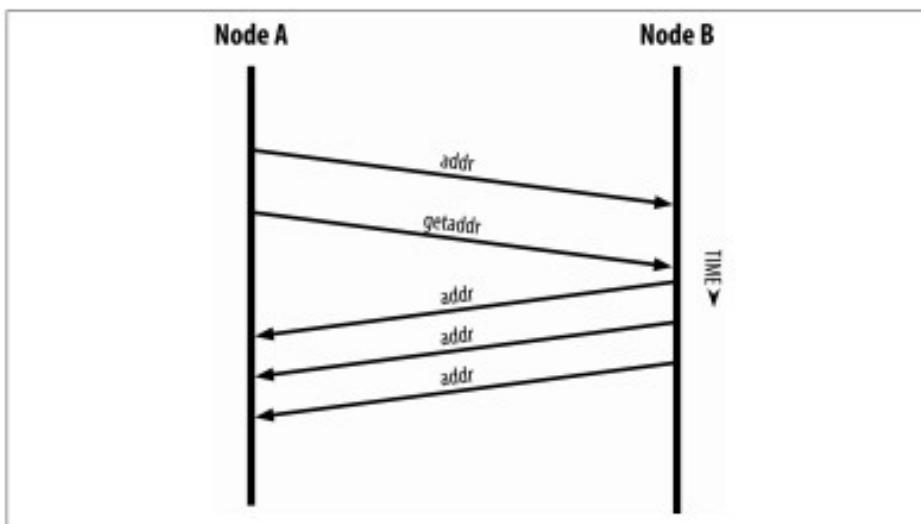


Figure 6-4. The initial handshake between peers

Gdy jedno lub więcej połączeń będzie osiągnięte, nowy węzeł wyśle wiadomość addr zawierającą własny adres IP do swoich sąsiadów. Sąsiedzi będą z kolei przekazywać wiadomość addr swoim sąsiadom, zapewniając, że nowo podłączony węzeł staje się dobrze znany i lepiej połączony. Dodatkowo, nowo podłączony węzeł może wysłać ge taddr do sąsiadów, prosząc ich, aby powrócili do listy adresów IP z innymi Peerami. W ten sposób, węzeł może znaleźć rówieśników do połączenia i ogłosić swoje istnienie w sieci dla innych węzłów, aby te mogły go odnaleźć. **Rysunek 6-5** przedstawia protokół adresu wykrycia:



Węzeł musi połączyć się z kilkoma różnymi perami w celu utworzenia różnych dróg do sieci bitcoin. Ścieżki nie są wiarygodne - węzły przychodzą i odchodzą, a więc węzeł musi wciąż odkrywać nowe węzły, gdyż traci stare połączenia, a także pomagać innym węzłom kiedy one się łączą. Potrzebne jest tylko jedno połączenie do załadowania, ponieważ pierwszy węzeł może zaoferować wstępny do swoich węzłów peerowych a Ci mogą zaoferować kolejne wstępny. Jest to także niepotrzebnym marnotrawstwem zasobów sieciowych, by połączyć się z większą ilością węzłów. Po ładowaniu, węzeł będzie pamiętał swoje najnowsze udane połączenie z peerem, tak, że jeśli jest uruchomiony może szybko przywrócić połączenia z dawnej sieci peer. Jeśli żaden z byłych kolegów nie odpowie na jego żądanie połączenia, węzeł może wykorzystać węzły nasion by ponownie załadować.

Na węźle działania klient Bitcoin Core można wymienić połączenia peera z komendą z każdym z 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" : 318131,
    "banscore" : 0,
    "syncnode" : true
},
{
    "addr" : "58.23.244.20:8333",
    "services" : "00000001",
    "lastsend" : 1405634127,
    "lastrecv" : 1405634124,
    "bytessent" : 4460918,
    "bytesrecv" : 8983575,
    "conntime" : 1405559628,
    "pingtime" : 0.00000000,
    "version" : 70001,
    "subver" : "/Satoshi:0.8.6/",
    "inbound" : false,
    "startingheight" : 311074,
    "banscore" : 0,
    "syncnode" : false
}
]
```

Aby wyłączyć automatyczne zarządzanie peerami oraz określić listę adresów IP, użytkownicy mogą dostarczyć opcję -connect = <adres ip> i podać jeden lub więcej z adresów IP. Jeśli opcja ta jest używana, węzeł połączy się tylko z wybranymi adresami IP, zamiast odkrywać i utrzymywać połączenia peer automatycznie.

Jeśli nie ma ruchu w połączeniach węzły będą okresowo wysyłać wiadomość w celu utrzymania połączenia. Jeśli węzeł nie komunikował się w połączeniu przez ponad 90 minut, to przyjmowany jest jako odłączony i nowy peer zostanie wymagany. Tym samym sieć dynamicznie dostosowuje się do przemijających węzłów i problemów z siecią oraz może organicznie rozwijać się i kurczyć w zależności od potrzeb, bez centralnego sterowania.

Pełne Węzły

Pełne węzły są węzłami, które utrzymują pełne łańcuchy blokowe wszystkich transakcji. Dokładniej, to prawdopodobnie powinno być nazywane "pełnym węzłem łańcuchów blokowych". We wczesnych latach Bitcoin, wszystkie węzły były pełnymi węzłami a obecnie klient Bitcoin Core jest pełnym węzłem łańcucha blokowego. W ciągu ostatnich dwóch lat nowe formy klientów Bitcoin zostały wprowadzone, jednakże on nie utrzymują pełnego łańcucha, ale działają jako klienci lekkiej wagi. Będziemy badać je bardziej szczegółowo w następnej części.

Pełne węzły łańcucha blokowego utrzymują kompletne i aktualne kopie łańcucha blokowego Bitcoin z wszystkimi transakcjami, które niezależnie budują się i podlegają weryfikacji, począwszy od pierwszego bloku (blok genesis) i rozbudowują się do ostatniego poznanego w sieci bloku. Pełny węzeł blockchain może samodzielnie i autorytywnie sprawdzać każdą transakcję bez regresu lub polegania na jakimkolwiek innym węźle lub źródle informacji.

Pełen węzeł łańcucha blokowego polega na sieci, aby otrzymywać powiadomienia o nowych blokach transakcji, które następnie się weryfikuje i wprowadza do swojej lokalnej kopii łańcucha blokowego.

Prowadzenie pełnego węzła łańcucha daje Ci czysto bitcoinowskie doświadczenie: niezależnej weryfikacji wszystkich transakcji bez konieczności polegania bądź uciekania się do zaufania jakichkolwiek innych systemów. Łatwo jest powiedzieć, czy używasz pełnego węzła ponieważ wymaga 20+ gigabajtów pamięci trwałej (miejscu na dysku), aby zachować pełny łańcuch blokowy. Jeśli potrzebujesz dużej ilości dysku twardego i proces trwa od dwóch do trzech dni, aby zsynchronizować się z siecią, wiesz, że uruchomiłeś pełny węzeł. To jest cena całkowitej niezależności i wolności od władz centralnej.

Istnieje kilka alternatywnych implementacji pełnego łańcucha blokowego klientów bitcoin, zbudowane przy użyciu różnych języków programowania i architektur oprogramowania. Jednak najczęstszą realizacją jest odniesienie klienta Bitcoin Core, znanego także jako klient Satoshi. Ponad 90% z węzłów w sieci bitcoin uruchamiają różne wersje Bitcoin Core. Jest to określone jako "Satoshi" w sub-wersyjnych ciągach wysłanych w wiadomości i wersjach wiadomości i przedstawionej jako komendę getpeerinfo jak widzieliśmy wcześniej; na przykład / Satoshi: 0.8.6 /.

Wymiana „Inwentarza”.

Pierwszą rzeczą jaką pełny węzeł będzie musiał wykonać gdy połączyc się z Peerami, to postarać się zbudować kompletny łańcuch blokowy. Jeśli jest to węzeł całkiem nowy i nie ma łańcucha blokowego, to zna tylko jeden blok, blok Genesis, który jest osadzony statycznie w oprogramowaniu klienta. Począwszy od bloku # 0 (blok Genesis), nowy węzeł będzie musiał pobrać setki tysięcy bloków do synchronizacji z siecią i przywrócenia pełnego łańcucha blokowego.

Proces synchronizowania łańcucha zaczyna się od wersji wiadomości, ponieważ zawiera BestHeight, bieżącą wysokość łańcucha blokowego danego węzła (liczba bloków). Węzeł będzie widział komunikaty wersji ze strony swoich Peerów, wiedząc, ile bloków ma każdy z nich i będzie mógł porównać, ile bloków ma w swoim łańcuchu blokowym. Węzły peerów będą wymianą a%605.420%% wiadomości bloków otrzymanych, które zawierają hashe (odcisk palca) na górnym bloku ich lokalnego łańcucha blokowego. Jeden z Peerów będzie w stanie zidentyfikować otrzymane hashe jako należące do bloku, który nie znajduje się na górze, ale raczej należy do starszego bloku, co sprawia, że można wydedukować, że jego lokalny łańcuch jest dłuższy niż jego peera.

Peer, która ma dłuższy łańcuch blokowy ma więcej bloków niż inny węzeł i może określić, które bloki z innego węzła potrzebują być dodane do "catch up" (doganiacza). To określi pierwszych 500 bloków do przekazania i transmisji ich hashów za pomocą inw (inwentarza) wiadomości. Węzeł gubiący te bloki będzie następnie zbierał je, wydając serię komunikatów Getdata żądających pełnych danych pakietowych i identyfikujących żądane bloki przy użyciu hashów z wiadomości inv.

Załóżmy na przykład, że węzeł ma tylko blok genesis. Wtedy pojawia się inv komunikat od jego peerów zawierający hashe o następnych 500 blokach w łańcuchu. Uruchomi ona prośby dotyczące bloków od wszystkich podłączonych peerów, rozkładając ograniczenia i zapewnieniając, że nie przytłacza jakiegokolwiek Peera swoimi prośbami. Węzeł śledzi ile bloków jest "w tranzycie" połączenia peer- peer, czyli oznacza to bloki, o które wnioskowano, ale nie odebrane, sprawdzając, czy nie przekracza limitu (MAX_BLOCKS_IN_TRANS IT_PER_PEER). W ten sposób, jeśli to wymaga dużo bloków, będzie żądać nowych tylko wtedy jak wcześniejsze żądania zostaną spełnione, umożliwiając Peerom kontrolować tempo aktualizacji i nie przytłaczając sieci. Gdy blok jest odbierany, jest on dodawany do łańcucha, jak zobaczymy w Rozdziale 7. Jako, że miejscowy łańcuch jest stopniowo budowany więcej bloków jest żądane i odbierane, a proces jest kontynuowany, aż węzły są złapane przez resztę sieci.

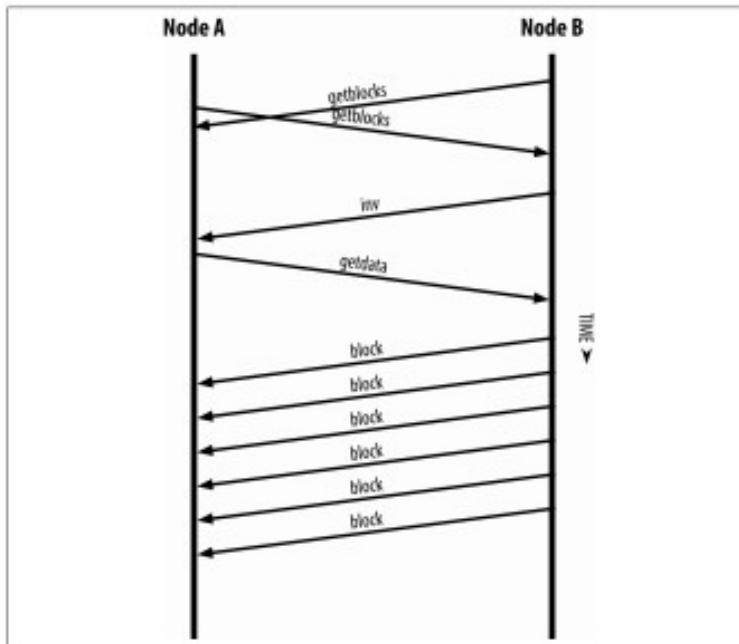
Ten proces porównywania lokalnego łańcucha z Peerami i pobierania wszelkich brakujących bloków dzieje się w każdej chwili, kiedy węzeł przechodzi w tryb offline na dowolny okres czasu. Niezależnie czy węzeł był niedostępny przez kilka minut i kilka bloków zostało zagubione, albo miesiąc i brakuje kilku tysięcy bloków, proces zaczyna poprzez od wysłania bloków biorących, otrzymania odpowiedź inv i rozpoczęcia pobieranie brakujących bloków. Rysunek 6-6 przedstawia inwentaryzację i protokół rozprzestrzeniania bloków.

Uproszczone weryfikacje płatności (SPV) węzłów.

Nie wszystkie węzły mają zdolność do przechowywania pełnego łańcucha bloków. Wielu klientów Bitcoin jest zaprojektowanych do pracy w urządzeniach o ograniczonej przestrzeni i mocy, takich jak smartfony, tablety czy systemy wbudowane. W takich urządzeniach, metoda uproszczonej weryfikacji płatności (SPV) jest używana w celu umożliwienia im działania bez zapisywania pełnego łańcucha blokowego. Te

typy klientów są nazywane klientami SPV lub lekkimi klientami. Jako adopcjonne przepięcia Bitcoin, węzeł SPV staje się najbardziej rozpowszechnioną formą węzła bitcoin, zwłaszcza dla portfeli Bitcoin.

Węzły SPV pobierają tylko nagłówki bloku, nie pobierając transakcji zawartych w każdym bloku. Powstały łańcuch bloków, beztransakcyjny, jest 1000 razy mniejszy niż pełny łańcuch bloków. Węzły SPV nie mogą zbudować pełnego obrazu wszystkich UTXO, które są dostępne do wydania, bo nie wiedzą o wszystkich transakcji w sieci. Węzły SPV sprawdzają transakcje przy użyciu nieco innej metodologii, która opiera się na Peerach, dla dostarczenia opinii na żądanie dotyczących istotnych części łańcuchów.



Rysunek 6-6. Węzeł synchronizujący łańcuch przez pobieranie bloków od Peerów

Przez analogię, pełny węzeł jest jak turysta w obcym mieście, wyposażony w szczegółową mapę każdej ulicy i każdego adresu. Dla porównania, węzeł SPV jest jak turysta w obcym mieście proszący przypadkowych nieznajomych o wskazówki krok po kroku, znając tylko jedną główną aleję. Jednakże obydwa turyści mogą zweryfikować istnienie ulicy odwiedzając ją, turysta bez mapy nie wie, co kryje się którejkolwiek z bocznych uliczek i nie wie, jakie inne ulice istnieją. Umieszczony z przodu 23 Church Street (Kościelna 23), turysta bez mapy nie może wiedzieć, czy istnieje tuzin innych "23 Church Street" (Kościelnych 23) w mieście i to ten adres jest właściwy. Szansą turysty bez mapy jest zadanie pytań wystarczającej ilości ludzi i nadzieję na to, że niektórzy z nich nie będą chcieli go okraść.

Uproszczona weryfikacja płatności sprawdza transakcje poprzez odniesienie do ich głębokości zamiast ich wysokości w łańcuchu. Podczas, gdy pełny łańcuch blokowy wybuduje w pełni potwierdzony łańcuch złożony z tysiące bloków i transakcji idących w dół łańcucha (wstecznie w czasie) przez całą drogę do bloku początkowego, węzeł SPV weryfikuje łańcuch złożony z wszystkich bloków (ale nie wszystkich transakcji) i odwołuje się do tego łańcucha zainteresowania transakcjami.

Na przykład, gdy bada transakcję w bloku 300000, wszystkie linki pełnych węzłów 300.000 idą w dół aż do genezy bloku i budują pełną bazę danych UTXO ustanawiając ważność transakcji potwierdzając, że UTXO pozostanie niewykorzystane. Węzeł SPV nie może sprawdzić, czy UTXO jest niewykorzystane. Zamiast tego, węzeł SPV ustanawiał powiązanie transakcji i bloku, który go zawiera, używając ścieżki Merkle (patrz: "Merkle trees – drzewak Merkle" na stronie 164). Następnie węzeł SPV czeka dopóki nie zobaczy sześciu bloków od 300,001 do 300,006 umieszczonego na wierzchu bloku zawierającego transakcję i sprawdzającego ją poprzez ustalenie jej głębokości w blokach 300,006 do 300,001. Fakt, że inne węzły w sieci akceptują blok 300.000 i ten wykonał pracę potrzebną do produkcji 6 bloków jest dowodem, przez proxy, że transakcja nie była dwukrotnie wydana.

Węzeł SPV nie może być przekonany, że istnieje transakcja w bloku gdy transakcja w rzeczywistości nie istnieje. Węzeł SPV stwierdzi istnienie transakcji w bloku żądając dowodu ścieżki Merkle przez walidację dowodu pracy w łańcuchu blokowym. Jednak istnienie transakcji może być "ukryte" z węzła SPV. Węzeł SPV na pewno może udowodnić, że transakcja istnieje, ale nie może sprawdzić, czy dana transakcja tak jak podwójne wydanie tego samego UTXO, nie istnieje, ponieważ nie mają rejestrów wszystkich operacji. Ta luka może być użyta w ataku denial-of-service lub w ataku podwójnych wydatków wobec węzłów SPV. W celu ochrony przed takim przypadkiem węzeł SPV musi podłączyć się losowo do różnych węzłów, w celu zwiększenia prawdopodobieństwa, że jest w połączeniu z co najmniej jednym uczciwym węzłem. Ta potrzeba podłączenia losowego oznacza, że węzły SPV również są narażone na ataki partycjonowania lub Sybil ataki, gdzie są podłączone do podrobionych lub fałszywych węzłów sieci i nie mają dostępu do uczciwych węzłów lub rzeczywistej sieci bitcoin.

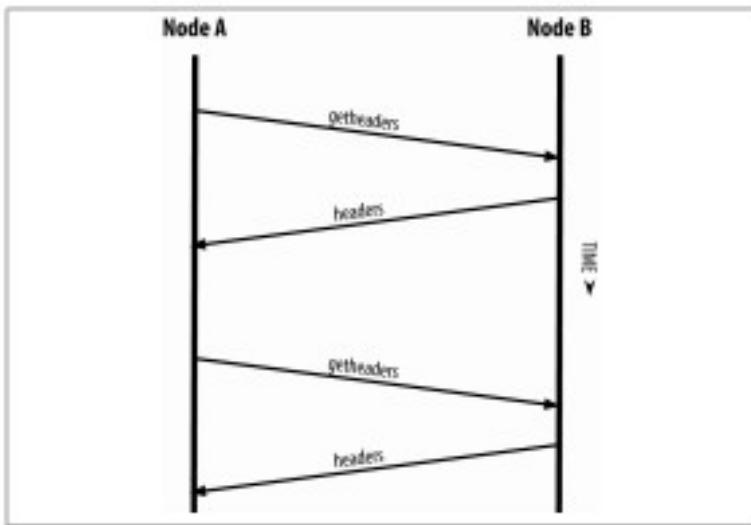
Dla większości celów praktycznych, dobrze połączone węzły SPV są wystarczająco bezpieczne, utrzymując właściwą równowagę między potrzebami zasobów, praktyczności i bezpieczeństwa. Dla nieomylnego bezpieczeństwa nic nie przebije uruchomienia pełnego łańcucha blokowego węzłów.



Pełny łańcuch węzłowy weryfikuje transakcję, sprawdzając zawarty w niej cały łańcuch tysięcy bloków w celu zagwarantowania, że UTXO nie jest wydawane, natomiast węzeł SPV sprawdza jak głęboko blok jest zakopany przez garść kilka bloków nad nim.

Aby uzyskać nagłówki bloków, węzły SPV używają wiadomości getheaders zamiast getblocks. Korespondujący Peerowie wysiączą ponad 2000 nagłówków blokowych za pomocą pojedynczych nagłówków wiadomości. Proces ten jest taki sam jak używany przez pełen węzeł do pobierania pełnych bloków. Węzły SPV także ustawiają filtr na połączenie z Peeri, do filtracji strumienia przyszłych bloków i transakcji przesyłanych przez Peerów. Wszelkie interesujące transakcje są pobierane za pomocą żądania Getdata. Peer w odpowiedzi generuje komunikat tx zawierający transakcje. **Rysunek 6-7** przedstawia synchronizację nagłówków bloków.

Rysunek 6-7.
nagłówki



Węzeł SPV synchronizuje bloków.

Ponieważ węzły SPV trzeba pobrać do konkretnych transakcji w celu sprawdzenia ich wybiórczo, one także tworzą zagrożenie prywatności. W przeciwieństwie do pełnych węzłów łańcucha blokowego, które zbierają wszystkie transakcje w ramach każdego bloku, węzły SPV żądając określenia specyficznych danych mogą nieumyślnie ujawnić adres w swoim portfelu. Na przykład, osoba trzecia monitorując sieć może śledzić wszystkich transakcji żądane przez portfel węzłem SPV i użyć tego do skojarzenia adresu Bitcoin użytkownika z jego portfelem, niszcząc jego prywatność.

Krótko po wprowadzeniu SPV / lekkich węzłów, deweloperzy Bitcoin dodali do adresu funkcję o nazwie Filtr bloom uwzględniając ryzyko zagrożenia węzłów SPV. Filtr Bloomu pozwalają węzłom SPV otrzymać podzbiór transakcji bez dokładnego ujawniania, które adresy są zainteresowane, poprzez mechanizm filtrowania, który używa prawdopodobieństwami zamiast ustalonych wzorców.

Filtr Bloom

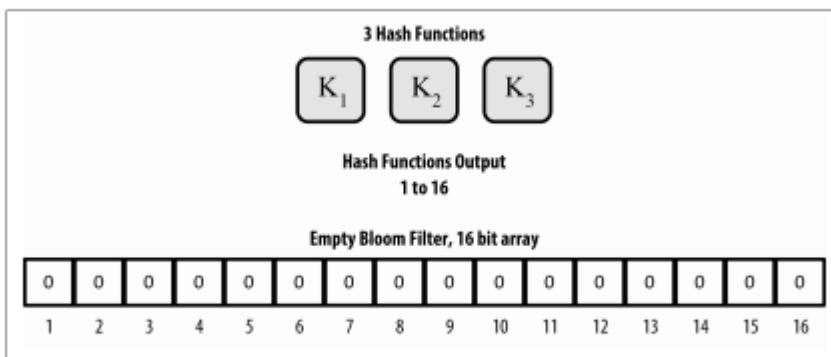
Filtr bloom jest probabilistycznym filtrem wyszukiwania, sposobem do opisania pożądanego wzoru bez określania go dokładnie. Filtry Bloomu oferują skuteczny sposób wyrażania wzorca wyszukiwania, przy jednoczesnej ochronie prywatności. Są one wykorzystywane przez węzły SPV, by zapytać swoich Peerów o transakcje pasujące do określonego wzorca, bez ujawniania dokładnie, które adresy są poszukiwane.

W naszym poprzedniej analogicznej sytuacji turysta bez mapy pyta o drogę do konkretnego adresu "23 Church St." Jeśli pyta obcych o drogę do tej ulicy, nieumyślnie ujawnia jej adres. Filtr Bloom zapytałby: "Czy istnieją jakieś ulice w tej dzielnicy, których nazwa kończy się na R-C-H?" Pytanie, jak to ujawnia nieco mniej informacji o żądanym miejscu niż prosiąc o "23 Church St." Stosując tę technikę, turysta może określić docelowy adres w większej szczegółowości jako "zakończenie U-R-C-H" lub mniej szczegółowo jako "zakończenie H." Przez zmieniającą się dokładność wyszukiwania, turysta odkrywa więcej lub mniej informacji, kosztem coraz mniej lub bardziej konkretnych rezultatów. Jeśli pyta o mniej specyficzny wzór, dostaje dużo więcej możliwych adresów i lepsze zachowanie prywatności, ale wiele z tych wyników nie mają żadnego znaczenia. Jeśli pyta o bardzo specyficzny wzór, dostaje mniej wyników, ale również traci prywatność.

Bardziej szczegółowy filtr bloom będzie produkować dokładny wynik, ale kosztem ujawnienia jakie adresy są używane w portfelu użytkownika. Mniej szczegółowy filtr bloom będzie produkować więcej danych na temat transakcji, mniej znaczących dla węzła, ale pozwoli węzlowi utrzymać lepszą prywatność.

Węzeł SPV zainicjuje filtr bloom jako "pusty" i w tym stanie filtr bloom nie dopasuje żadnych wzorów. Węzeł SPV będzie robić listę wszystkich adresów w swoim portfeli i stworzy wzorzec wyszukiwania pasujący do bloku wyjściowego transakcji odpowiadającemu każdemu adresowi. Zazwyczaj wzorcem wyszukiwania jest skrypt pay-to-public-key-hash, który jest oczekiwany przez skrypt blokujący, który będzie obecny w każdej transakcji płacąc public-key-hash (adres). Jeśli węzeł SPV śledzi balans adresu P2SH, wzorem wyszukiwania będzie skrypt pay-by-script-hash. Węzeł SPV następnie dodaje każdy z wyszukiwanych wzorców do filtra bloom, tak żeby filtr bloom rozpoznał wyszukiwany wzorzec, jeśli jest on obecny w transakcji. Na koniec, filtr bloom przesyła do peerów i do użytkowników peerów dopasowane transakcje dla transmisji ich do węzłów SPV.

Filtre Bloom są implementowane jako tablica o zmiennej wielkości cyfr binarnych N (pole bitowe) i zmiennej liczby funkcji hasha M. Funkcje hash są zaprojektowane by zawsze wytwarzaly sygnał wyjściowy, który jest pomiędzy 1 i N korespondując z tablicą z cyfr binarnych. Funkcje hash są generowane w sposób deterministyczny, tak że każdy węzeł realizujący filtr bloom zawsze będzie korzystać z tych samych funkcji hash i uzyska takie same wyniki dla konkretnego wejścia. Wybierając różne długości (N) Filtrów Bloom i innego numeru (M) funkcji hash, filtr bloom można dostosować, zmieniając poziom dokładności i także prywatności.



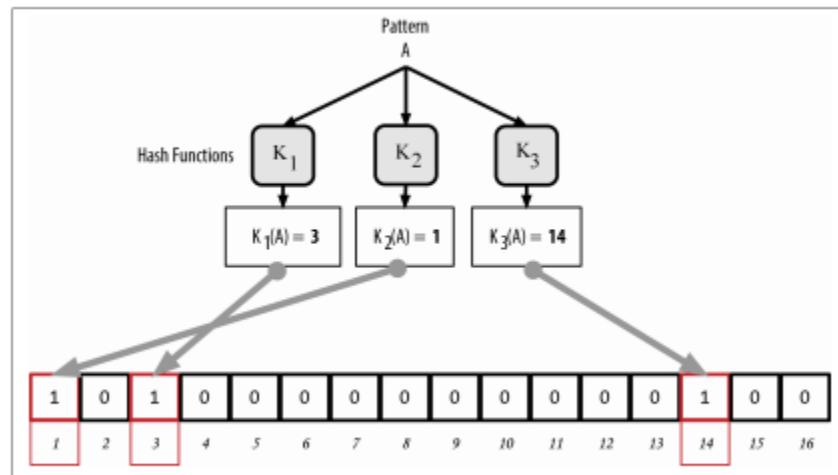
Na Rysunku 6-8, używamy bardzo małej tablicy 16 bitów i zestawu trzech funkcji skrótu w celu wykazania, jak działają filtry bloom.

Rysunek 6-8. Przykład uproszczonego filtra bloom, z polem 16-bitowym i trzy funkcje hasha.

Filtr Bloom inicjowany jest tak, żeby tablica bitów była cała zerowa. Aby dodać wzór do filtra bloom, wzór jest mieszany przez każdą funkcję hasha po kolej. Stosując pierwszą funkcję hasha do wyników wejściowych w liczbie pomiędzy 1 do N. odpowiedni bit w tablicy (indeksowane od 1 do N) jest znaleziony i ustawiony na 1, a tym samym nagrywa blok wyjściowy funkcji hasha. Następnie, kolejna funkcja hash jest używana do ustawiania innego kawałka i tak dalej. Gdy wszystkie funkcje skrótu M zostały zastosowane, wzorzec wyszukiwania zostanie "zapisany" w filtrze bloom jako bity M, które zostały zmienione od 0 do 1. Rysunek 6-9 jest przykładem dodawania wzorca "A" do prostego Filtra bloom ukazanego na rysunku 6-8. Dodanie drugiego wzoru jest tak proste jak powtarzanie tego procesu. Wzór jest mieszany przez każdą z funkcji hash po kolej, a wynik jest rejestrowany przez ustawienie bitów na 1. Należy pamiętać, że jak filtr bloom jest wypełniony większą ilością wzorów, wynik funkcji skrótu może pokrywać się z kawałkiem, który jest już ustawiony na 1, przy czym końcówka nie jest zmieniana.

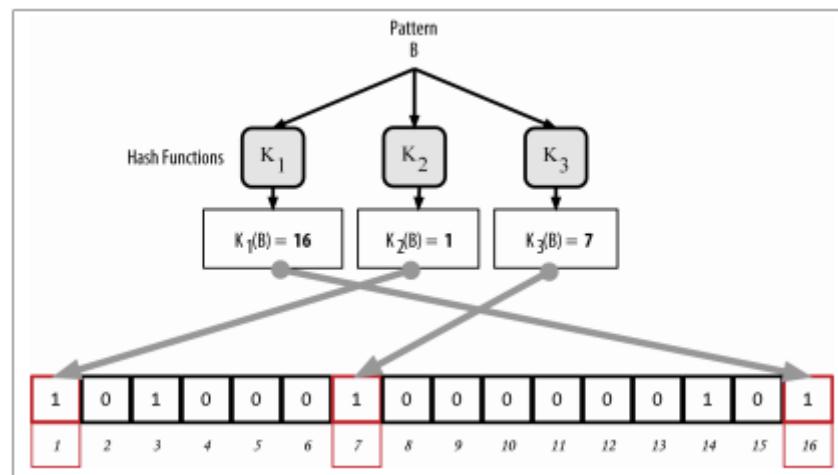
W istocie, im więcej wzorów jest zarejestrowanych na nakładających się bitach filtr bloom zaczyna być nasycony z większą liczbą bitów ustawionych na 1 i dokładność filtra maleje. To dlatego filtr ma

probabilistyczną strukturę danych - robi się mniej dokładny im więcej wzorów jest dodawanych. Określana dokładność zależy od liczby dodanych wzorów w stosunku do rozmiaru tablicy bitów (N) i liczby funkcji mieszania (M). Większa matryca i więcej funkcji hash może rejestrować kolejne wzory z większą dokładnością. Mniejsza tablica bitowa lub mniej funkcji hash odnotuje mniej wzorów i skutkuje mniejszą dokładnością.



Rysunek 6-9. Dodanie wzoru "A" do naszego prostego filtra bloom

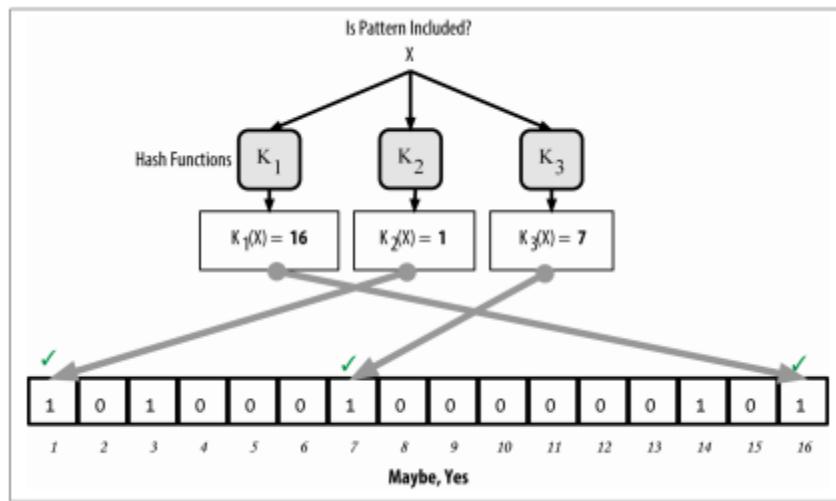
Rysunek 6-10 przedstawia przykład dodania drugiego wzoru "B" do prostego filtra bloom.



Aby sprawdzić, czy wzór jest częścią filtru bloom, jest on mieszany przez każdą z funkcji hash i uzyskany wzór bitowy jest testowany ponownie w tablicy bitów. Jeśli wszystkie bity indeksowane przez funkcje hash są ustawione na 1, to wzór jest prawdopodobnie zapisany w filtrze bloom.

Ponieważ bity mogą być ustawione z powodu nakładania się wielu wzorów, odpowiedź nie jest pewna, ale jest raczej probabilistyczna. W prostych słowach, pozytywne połączenie filtru bloom to wynik "Być może tak."

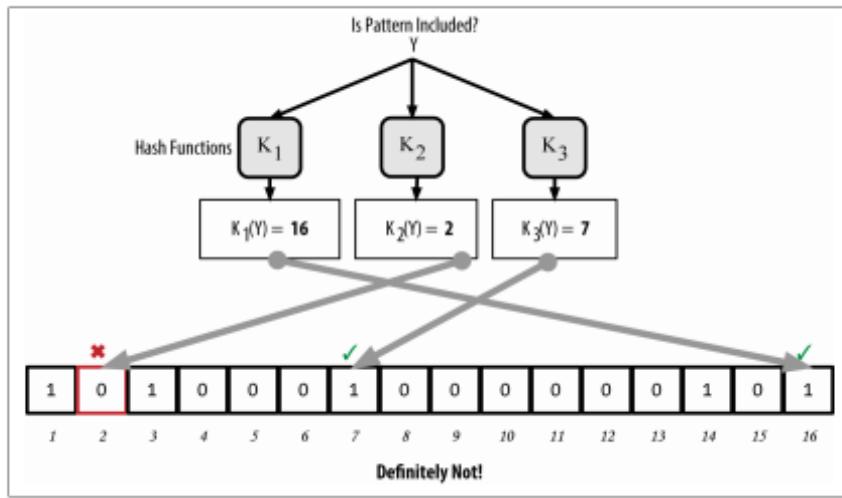
Rysunek 6-11 jest przykładem badania istnienia wzorca "X" w prostym filtrze Bloom. Korespondujące bity są ustawione na 1, więc wzór jest prawdopodobnie połączony.



Rysunek 6-11. Testowanie istnienie wzorca "X" w filtrze bloom. Wynikiem jest prawdopodobny pozytywny wynik, czyli "może".

Z drugiej strony, jeżeli próbka jest badana przeciwko filtrowi Bloom i dowolny z bitów jest ustawiony na 0, to to dowodzi, że wzorzec nie został nagrany w filtrze bloom. Negatywny wynik nie jest prawdopodobieństwem, jest pewności. W prostych słowach, negatywnym połączeniu w filtrze bloom jest "Zdecydowanie nie!"

Rysunek 6-12 jest przykładem badania istnienia wzoru "Y" w prostym filtrze bloom. Jeden z korespondujących bitów jest ustawiony na 0, więc wzór nie pasują z pewnością.



Rysunek 6-

12.

Testowanie istnienia wzorca "Y" w filtrze bloom. Rezultatem jest ostateczne negatywne połączenie, czyli "Zdecydowanie nie!"

Wdrożenie Bitcoin za pomocą filtrów Bloom jest opisana w Bitcoin Improvement Proposal 37 (BIP0037). Patrz Dodatek B lub odwiedź GitHub.

Filtr Bloom i aktualizacje inwentarza

Filtre Bloom są używane do filtrowania transakcji (oraz bloków je zawierających), które węzeł SPV otrzymuje od swoich peerów. Węzły SPV będą tworzyć filtr, do których pasować będą tylko adresy posiadane w portfelu węzła SPV. Węzeł SPV wyśle wiadomość filterload (filtr załączony) do uczestnika, zawierającą filtr bloom do wykorzystania na połączenie. Po tym jak filtr jest ustalony, peer będzie następnie testować zachowanie wyjścia każdej transakcji w stosunku do filtra bloom. Tylko transakcje pasujące do filtra są wysyłane do węzła.

W odpowiedzi na wiadomości z danymi z węzła, peerzy wyśliją wiadomość merkleblock, która zawiera tylko nagłówki bloków dla bloków odpowiadających filtrowi i ścieżkom Merkle (patrz "drzewo hash" na stronie 164) dla każdej pasującej transakcji. Peer będzie wówczas także nadawca wiadomości TX zawierających transakcje dopasowane przez filtr.

Ustawienia węzła filtru Bloom mogą interaktywnie dodawać wzory do filtra poprzez wysłanie wiadomość filteradd (dodaj filtr). Aby wyczyścić filtr bloom, węzeł może wysłać wiadomość filterclear (czyścić węzeł). Ponieważ nie jest możliwe usunięcie wzoru z filtra bloom, węzeł musi usunąć i ponownie wysłać nowy filtr bloom jeśli wzorzec nie jest dłużej pożądany.

Zbiorniki transakcyjne

Prawie każdy węzeł w sieci bitcoin utrzymuje tymczasową listę niepotwierdzonych transakcji zwanych zbiornikiem pamięci lub zbiornikiem transakcji. Węzły używają tego zbioru do śledzenia transakcji, które są znane w sieci, ale nie zostały jeszcze uwzględnione w łańcuchu blokowym. Na przykład, węzeł, który posiada portfel użytkownika będzie korzystać ze zbiornika do śledzenia transakcji płatności

przychodzących do portfela użytkownika, które zostały wrzucone do sieci, ale nie zostały jeszcze potwierdzone.

Kiedy transakcje są odebrane i zweryfikowane, są one dodawane do zbiornika transakcji i przekazywane do sąsiednich węzłów w celu rozprzestrzenienia się w sieci.

Niektóre implementacje węzłów utrzymują również oddzielną pulę osieroconych transakcji. Jeżeli wejście transakcji odnosi się do transakcji, która nie jest jeszcze znana transakcja jest sierotą, tak jakby utraciła rodziców, będzie przechowywana tymczasowo w puli sierocym dopóki rodzic transakcji nie przybędzie.

Gdy transakcja zostanie dodana do zbiornika transakcji, basen sierocy jest sprawdzany pod kątem ewentualnych sierot, które odwołują się do wyjścia tej transakcji (jej dzieci). Wszelkie dopasowania sieroty są następnie zatwierdzane. Jeżeli jest ważna, to usuwa się ją ze zbiornika sierocego i dodaje do puli transakcji zakańczając łańcuch, który rozpoczęła transakcja rodziców. W świetle nowo dodanej transakcji, która nie jest już sierotą, proces ten jest powtarzany rekursively poszukując dalszych zstępnych, dopóki nie zostanie znalezione żadne dalsze potomstwo. Poprzez ten proces, przybicie transakcji macierzystej wyzwala kaskadę rekonstrukcji z całego łańcucha transakcji współzależnych poprzez ponowne złączenie sieroty z rodzicami w drodze w dół łańcucha.

Zarówno zbiornik transakcji i zbiornik sierocy (miejsce gdzie jest realizowany) są przechowywane w pamięci lokalnej i nie są zapisywane w pamięci trwałej; Przeciwnie, są one dynamicznie wypełniane przychodząymi wiadomościami sieciowymi. Gdy rozpoczyna się węzeł, oba baseny są puste i stopniowo wypełniają się nowymi transakcjami otrzymanymi w sieci.

Niektóre implementacje klienta Bitcoin również utrzymują bazę UTXO lub zbiornik UTXO, który jest zbiorem wszystkich niewykorzystanych wyjść na łańcuchu blokowym. Choć nazwa "zbiornik UTXO" brzmi podobnie do zbiornika transakcji, reprezentuje inny zestaw danych. W przeciwieństwie do zbiorników transakcji i osieroconych, zbiornik UTXO nie jest zainicjowany jako pusty lecz zawiera miliony wpisów z niewykorzystanych wyjść transakcyjnych, w tym niektóre, których historia sięga 2009 roku.

Zbiornik UTXO może być trzymany w pamięci lokalnej lub w indeksowanej tabeli bazy danych w pamięci trwałej, podczas gdy transakcja i zbiorniki sieroce reprezentują lokalną perspektywę pojedynczego węzła i mogą znacznie różnić się dla każdego węzła, w zależności od tego, kiedy węzeł został uruchomiony lub ponownie uruchomiony, zbiornik UTXO reprezentuje porozumienie sieci i dlatego będzie różnić się tylko nieco między węzłami. Ponadto, transakcja i zbiornik sierocy zawierają jedynie transakcje niepotwierdzone, a zbiornik UTXO zawiera tylko potwierdzone wyjścia.

Wiadomości Alertowe

Komunikaty alertu są rzadko używanymi funkcjami, jednakże są realizowane przez większość węzłów. Wiadomości alertowe są "System komunikatu alarmowego" Bitcoin, z co oznacza którym rdzeniem deweloperskim Bitcoin należy wysłać wiadomość alarmową do wszystkich węzłów Bitcoin. Funkcja ta jest realizowana w celu umożliwienia trzonowi zespołu deweloperów powiadomienie wszystkich użytkowników Bitcoin dotyczące poważnego problemu w sieci bitcoin, takiego jak błąd krytyczny, który wymaga działania użytkownika. System ostrzegania stosowany był jedynie kilka razy, zwłaszcza na początku 2013 roku, kiedy błąd krytyczny w bazie spowodował widełki multiblokowe występujące w łańcuchu bitcoin.

Komunikaty alertowe są propagowane przez wiadomości alarmowe. Wiadomość zawiera kilka pól, w tym:

Identyfikator alertu- tak, aby powielone alarmy były wykryte,

Data wygaśnięcia - pewien czas, po upływie którego alert wygasza

RelayUntil - czas, po których wpis nie powinien być przekazany

MinVER, MaxVer - Zakres wersji protokołu Bitcoin, której ten alert dotyczy

subVer- wersja oprogramowania klienta, do której odnosi się ten alarm

Priorytet - Poziom priorytetu alertowego, obecnie nieużywany

Alerty są kryptograficznie podpisane za pomocą klucza publicznego. Odpowiedni klucz prywatny jest w posiadaniu kilku wybranych członków zespołu rozwoju lokalnego. Podpis cyfrowy zapewnia, że fałszywe alerty nie zostaną przekazane do sieci.

Każdy węzeł odbierający ten komunikat alertu będzie musiał go zweryfikować, sprawdzić jego wygaszenie i propagować go do wszystkich swoich peerów, co zapewnia szybkie rozprzestrzenienie w całej sieci. Oprócz propagowania alertu, węzły mogą realizować funkcję interfejsu użytkownika do zaprezentowania alertu dla użytkownika.

W kliencie Bitcoin Core alert jest skonfigurowany z opcją wiersza poleceń - alertnotify, która określa polecenie, które ma zostać uruchomione po otrzymaniu alertu. Komunikat alertu jest przekazywany jako parametr do komendy alertnotify. Najczęściej polecone alertnotify jest ustawiane do generowania wiadomości e-mail do administratora węzła, zawierając komunikat ostrzegawczy. Alarm jest również wyświetlany w oknie pop-up w graficznym interfejsie użytkownika (Bitcoin-Qt), jeśli jest on uruchomiony.

Inne implementacje protokołu bitcoin alert może obsłużyć wpisu na różne sposoby. Wiele systemów sprzętowych Bitcoin mining nie realizuje funkcji ostrzegania wiadomością, ponieważ nie mają interfejsu użytkownika. Zaleca się, żeby górnicy uruchomili takie systemy wydobywcze subskrybujące alerty poprzez operatora zbiornika górnictwa lub uruchamiając lekki węzeł tylko dla celów alarmowych.

Rozdział 7

Łańcuch blokowy

Wprowadzenie

Struktura danych łańcucha bloków jest uporządkowaną, wsteczną listą bloków transakcyjnych. łańcuchy mogą być przechowywane w pliku płaskim lub w zwykłą bazie danych. Klient Bitcoin Core przechowuje łańcuch metadanych przy użyciu bazy danych LevelDB Google. Bloki są połączone „wstecz”, a każdy odnosi się do poprzedniego bloku w łańcuchu. łańcuch często jest wizualizowany jako pionowy komin, z blokami warstwowymi na wierzchu każdego bloku i pierwszym blokiem służącym jako podstawa stosu. Wizualizacja bloków ułożone na wierzchu od siebie powoduje użycie wyrażeń takich jak "wysokość" w odniesieniu do odległości od pierwszego bloku, a "góra" lub "końcówka" w odniesieniu do ostatnio dodanego bloku.

Każdy blok w łańcuchu blokowym identyfikuje się przez hash, generowany za pomocą algorytmu kryptograficznego hasha SHA256 w nagłówku bloku. Każdy blok odwołuje się także do poprzedniego bloku, zwanym blokiem macierzystym, poprzez „previous block hash” umieszczonym w nagłówku bloku. Innymi słowy, każdy blok zawiera hash jego rodzica wewnątrz własnego nagłówka. Kolejność hashy łączących poszczególne bloki z jego rodzicem tworzy łańcuch wracający aż do pierwszego bloku, jaki kiedykolwiek stworzono, zwany blokiem genezy.

Chociaż blok ma tylko jednego rodzica, może tymczasowo mieć wiele dzieci. Każde z dzieci odnosi się do tego samego bloku, jak jego rodzic i zawiera ten sam (macierzysty) hash w „poprzednim hashu bloku” pola. Wiele dzieci powstają podczas rozwidlenia łańcucha, sytuacji czasowej, która występuje, gdy odkryje się różne bloki niemal równocześnie przez różnych górników (patrz "Rozwidlenie łańcucha" na stronie 199). Ostatecznie tylko jeden blok dziecko staje się częścią łańcucha i "widelec" zostaje rozwiązyany. Mimo, że blok może mieć więcej niż jedno dziecko, każdy blok może mieć tylko jednego rodzica. To dlatego, że blok ma tylko jeden "Poprzedni hash bloku" pole jest wewnątrz nagłówka bloku, a tym samym wpływa na hash aktualnego bloku.

Własna tożsamość dziecka zmienia się, jeśli tożsamość rodziców się zmienia. Gdy rodzic jest w jakikolwiek sposób modyfikowany, zmienia się hash rodzica. Zmieniony hash rodzica powoduje konieczność zmiany w "poprzednim hashu bloku" wskazanym przez dziecko. To z kolei powoduje zmianę hasha dziecka, co wymagałoby też zmiany wskaźnika wnuka, co z kolei zmienia wnuka i tak dalej. Ten efekt kaskadowy zapewnia, że gdy blok ma wiele pokoleń po nim, nie może być zmieniony bez wymuszania ponownego obliczenia wszystkich kolejnych bloków. Ponieważ takie przeliczanie wymagałoby ogromnej ilości obliczeń, istnienie długiego łańcucha bloków sprawia głęboką historię łańcuchową za niezmienne, co jest kluczowym elementem bezpieczeństwa Bitcoin.

Jedynym sposobem, aby myśleć o łańcuchu blokowym są warstwy w formacji geologicznej lub próbki rdzenia lodowca. Warstwy powierzchniowe mogą ulec zmianie wraz z porami roku, a nawet zostać zdmuchnięte zanim będą miały czas na osiedlenie się. Ale kiedy zagłębiasz się o kilka cali, głębokie warstwy geologiczne stają się coraz bardziej stabilne. Przez jakiś czas spójrz kilkaset stóp w dół, patrząc patrzysz na migawkę z przeszłości, która pozostała w nienaruszonym stanie przez miliony lat. W łańcuchu, najnowsze kilka bloków może ulec zmianie, jeśli istnieje podstawa do rekalkulacji łańcucha - ze względu na rozwidlenie. Górnne sześć bloków są jak kilka cali od wierzchniej warstwy gleby. Ale kiedy idziesz głębiej w łańcuch, więcej niż sześć bloków, bloki są coraz mniej możliwe, aby je zmienić. Powyżej 100 bloków jest ich tak dużo, że transakcja wymaga stabilności tak, że transakcja coinbase -transakcja zawierający świeżo wydobyte bitcoiny, może być wydana. Kilka tysięcy bloków wstecz (w miesiącu) i łańcuch jest rozliczany przez historię. To nigdy się nie zmieni.

Struktura bloku

Blok jest pojemnikiem na dane, który agreguje transakcje do wpisania do księgi publicznego i łańcucha bloków. Blok wykonany jest z nagłówka zawierającego metadane, po których następują długie listy transakcji składające się na większą część jego wielkości. Nagłówek bloku wynosi 80 bajtów, podczas gdy średnia wartość transakcji wynosi co najmniej 250 bajtów i średni blok zawiera ponad 500 transakcji. Kompletny blok ze wszystkich transakcji jest zatem 1000 razy większy niż nagłówek bloku. **Tabela 7-1** opisuje strukturę bloku.

Rozmiar	Obszar	Opis
4 bajty	Rozmiar bloku	Rozmiar bloku w bajtach podążającego tym obszarem
80 bajtów	Nagłówek bloku	Kilka obszarów zrobionych z nagłówka
1-9 bajtów (VarInt)	Licznik transakcji	Ile podąża transakcji
zmienny	transakcje	transakcje zapisywane w tym bloku

Nagłówek bloku

Nagłówek Bloku składa się z trzech zestawów metadanych bloku. Po pierwsze, istnieje odniesienie do hasza poprzedniego bloku, które łączy ten blok z poprzednim blokiem w łańcuchu. Drugi zestaw metadanych, tj. trudności, czasowe i chwilowe, odnoszą się do zawodów górniczych, jak opisano w **Rozdziale 8**. Trzeci element metadanych to korzeń drzewa Merkle, struktura danych wykorzystywana do efektywnego podsumowania wszystkich transakcji w bloku. Tabela 7-2 opisuje strukturę nagłówka bloku.

Rozmiar	Obszar	Opis
4 bajty	wersja	Numer wersji oprogramowania do śledzenia / aktualnienia protokołu

32 bajty	Hash poprzedniego bloku	Odniesienie do hasha poprzedniego (macierzystego) bloku w łańcuchu
32 bajty	Korzeń Merkle	Hash korzenia drzewa Merkle transakcji tego bloku
4 bajty	Datownik	Przybliżony czas utworzenia tego bloku (sekundy od epoki Uniksa)
4 bajty	Trudność docelowa	Algorytm docelowy proof-of-work utrudnieniem dla tego bloku
4 bajty	chwilowy	Licznik używany dla algorytmu proof-of-work

Wartość jednorazowa, cel trudności i datownik są wykorzystywane w procesie wydobywczym i zostaną omówione bardziej szczegółowo w [Rozdziale 8](#).

Identyfikatory bloku: Hash nagłówka bloku i wysokość bloku

Podstawowym identyfikatorem bloku jest jego kryptograficzny hash, cyfrowy odcisk palca, dwukrotnie skracany przez algorytm SHA256. Powstały hash 32- bajtowy jest nazywany hashem bloku, ale jest bardziej dokładne jest nazywanie go hashem nagłówków bloku, ponieważ tylko nagłówek bloku jest używany, aby go obliczyć. Na przykład, 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

jest hashem pierwszego bloku bitcoin jaki kiedykolwiek stworzono. Hash bloku identyfikuje blok jednoznacznie i jednoznacznie i może być niezależnie uzyskany przez dowolny węzeł przez proste hashowanie nagłówka bloku.

Należy zauważyć, że hash bloku w rzeczywistości nie jest zawarty wewnątrz struktury danych w bloku, ani wtedy, gdy blok jest przesyłany w sieci, ani gdy jest przechowywany w pamięci trwałej węzła jako część łańcucha bloków. Zamiast tego hash tego bloku obliczany jest przez każdy węzeł jako blok odebrany z sieci. Hash bloku może być przechowywany w oddzielnej tabeli bazy danych jako część metadanych bloku, w celu ułatwienia indeksowania i szybszego odzyskiwanie bloków z dysku.

Drugim sposobem identyfikacji bloku jest identyfikacja pozycji w łańcuchu blokowy, zwanej wysokością bloku. Pierwszy blok jaki kiedykolwiek stworzono jest na wysokości bloku 0 (zero) i jest tym samym blokiem, który wcześniej był określany przez następujący hash bloku 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f. Blok może być zatem identyfikowany na dwa sposoby: poprzez odniesienie do hasha bloku lub przez odwołanie się do wysokości bloku. Każdy kolejny blok dodany "w górę" tego pierwszego bloku jest o pozycję "wyższy" w łańcuchu, tak jak pudełka ułożone jedno na drugie. Wysokość blokowa w dniu 1 stycznia 2014 roku, wynosiła około 278.000, co oznacza, że były 278.000 bloki ułożone na górze pierwszego bloku utworzonego w styczniu 2009 roku.

W przeciwieństwie do hasza bloku, wysokość bloku nie jest unikalnym identyfikatorem. Chociaż pojedynczy blok zawsze będzie miał szczególną niezmienną wysokość bloku, rewers nie jest prawdą, wysokość bloku nie zawsze identyfikuje pojedynczy blok. Dwa lub więcej bloki, mogą mieć taką samą wysokość bloku, konkurując do tej samej pozycji w łańcuchu. Ten scenariusz jest omówione szczegółowo w rozdziale "Rozwidlenia łańcucha". Wysokość bloku nie jest częścią struktury danych w bloku; nie jest przechowywana w bloku. Każdy węzeł dynamicznie określa pozycję danego bloku (wysokość) w łańcuchu, gdy jest odbierany z sieci bitcoin. Wysokość bloku może być również przechowywane jako metadane w indeksowanej tabeli bazy danych w celu szybszego pobierania.



Hash bloku zawsze identyfikuje pojedynczy blok jednoznacznie. Blok też zawsze ma określoną wysokość bloku. Jednak nie jest to regułą, że określona wysokość bloku może zidentyfikować jeden konkretny blok. Dwa lub więcej bloków może konkurować o jedną pozycję w blokowym łańcuchu.

Blok genezy

Pierwszy blok w łańcuchu blokowym nazywamy blokiem genezy. Został on utworzony w roku 2009. Jest to wspólny przodek wszystkich bloków w łańcuchu blokowym, co oznacza, że jeśli zaczniesz w jakimkolwiek bloku, a następnie cofniesz się po łańcuchu to w końcu dotrzesz do bloku genezy.

Każdy węzeł zaczyna się zawsze od łańcucha blokowego co najmniej jednego bloku, ponieważ blok genezy jest statycznie zakodowany w oprogramowaniu klienta bitcoin, tak, żeby nie mógł być zmieniony. Każdy węzeł zawsze "zna hash bloku genezy i jego strukturę, czas ustalony, który został utworzony, a nawet każdą transakcję w nim. Tak więc, każdy węzeł ma punkt wyjścia dla łańcucha, bezpieczny "root" (korzeń), z którego można zbudować zaufany łańcuch.

Zobacz statycznie zakodowany blok genezy wewnątrz klienta Bitcoin Core w chainpar- ams.cpp.

Następujący identyfikator skrótu należy do bloku genezy:
00000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

Możesz szukać tego hasha bloku w każdej stronie bloku Explorer, takiej jak blockchain.info, a znajdziesz stronę opisującą zawartość tego bloku, z adresem URL zawierającym ten hash:

<https://blockchain.info/block/>
00000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
<https://blockexplorer.com/block/>
00000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

Korzystając z klienta odniesienia Bitcoin Core w wierszu poleceń:

Using the Bitcoin Core reference client on the command line:

```
$ bitcoind getblock  
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f  
{  
    "hash" : "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",  
    "confirmations" : 308321,  
    "size" : 285,  
    "height" : 0,  
    "version" : 1,  
    "merkleroot" : "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afde  
da33b",  
    "tx" : [  
        "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"  
    ],  
    "time" : 1231006505,  
    "nonce" : 2083236893,  
    "bits" : "1d00ffff",  
    "difficulty" : 1.00000000,  
    "nextblockhash" :  
        "0000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"  
}
```

Blok genezy zawiera ukrytą wiadomość w jego obrębie. Wejście transakcji coinbase zawiera tekst "The Times 03 / Jan / 2009 cancellor on brink of second bailout for banks." Ta wiadomość została przeznaczona do zaoferowania dowód najwcześniejszej daty, w której został stworzony, poprzez odniesienie nagłówka brytyjskiej gazety The Times. Służy także jako ironiczne przypomnienie znaczenia niezależnego systemu monetarnego, startu systemu Bitcoin, który zbiegł się w czasie ze światowym kryzysem finansowym. Wiadomość została osadzona w pierwszym bloku przez Satoshi Nakamoto, twórcę Bitcoin.

Łączenie bloków w łańcuchy

Pełne węzły Bitcoin utrzymują lokalną kopię łańcucha Bitcoin, zaczynającą się od bloku genezy. Lokalna kopia łańcucha jest stale aktualizowana w miarę znalezienia nowych bloków i wykorzystane do rozbudowania łańcucha. Ponieważ węzeł odbiera przychodzące z sieci bloki, to będzie weryfikować te bloki, a następnie połączy je z istniejącym łańcuchem. Aby nawiązać połączenie, węzeł zbada przychodzące nagłówki bloku i poszuka "poprzedniego hasha bloku".

Załóżmy na przykład, że węzeł ma 277,314 bloków w lokalnej kopii łańcucha blokowego. Ostatnim blokiem węzła, który jest znany jest blok 277,314, z hashem nagłówka bloku 000000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249.

Węzeł Bitcoin następnie odbiera nowy blok z sieci, który analizuje się następująco:

```
{  
    "size" : 43560,  
    "version" : 2,
```

```

"previousblockhash" :
    "0000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
"merkleroot" :
    "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
"time" : 1388185038,
"difficulty" : 1180923195.25802612,
"nonce" : 4215469401,
"tx" : [
    "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",
    "#[... many more transactions omitted ...]
        "
    ]
}

```

Patrząc na nowe bloki, węzeł znajdzie pole previousblockhash, który zawiera skrót hash bloku macierzystego. Jest to hash znany dla węzła, którego ostatni blok w łańcuchu jest na wysokości 277,314. Dlatego ten nowy blok jest dzieckiem ostatniego bloku na łańcuchu i rozszerza istniejący łańcuch. Węzeł dodaje ten nowy blok na koniec łańcucha, dzięki czemu łańcuch blokowy ma już nową wysokość 277,315. Rysunek 7-1 przedstawia łańcuch trzech bloków, połączonych przez odniesienia previousblockhash.

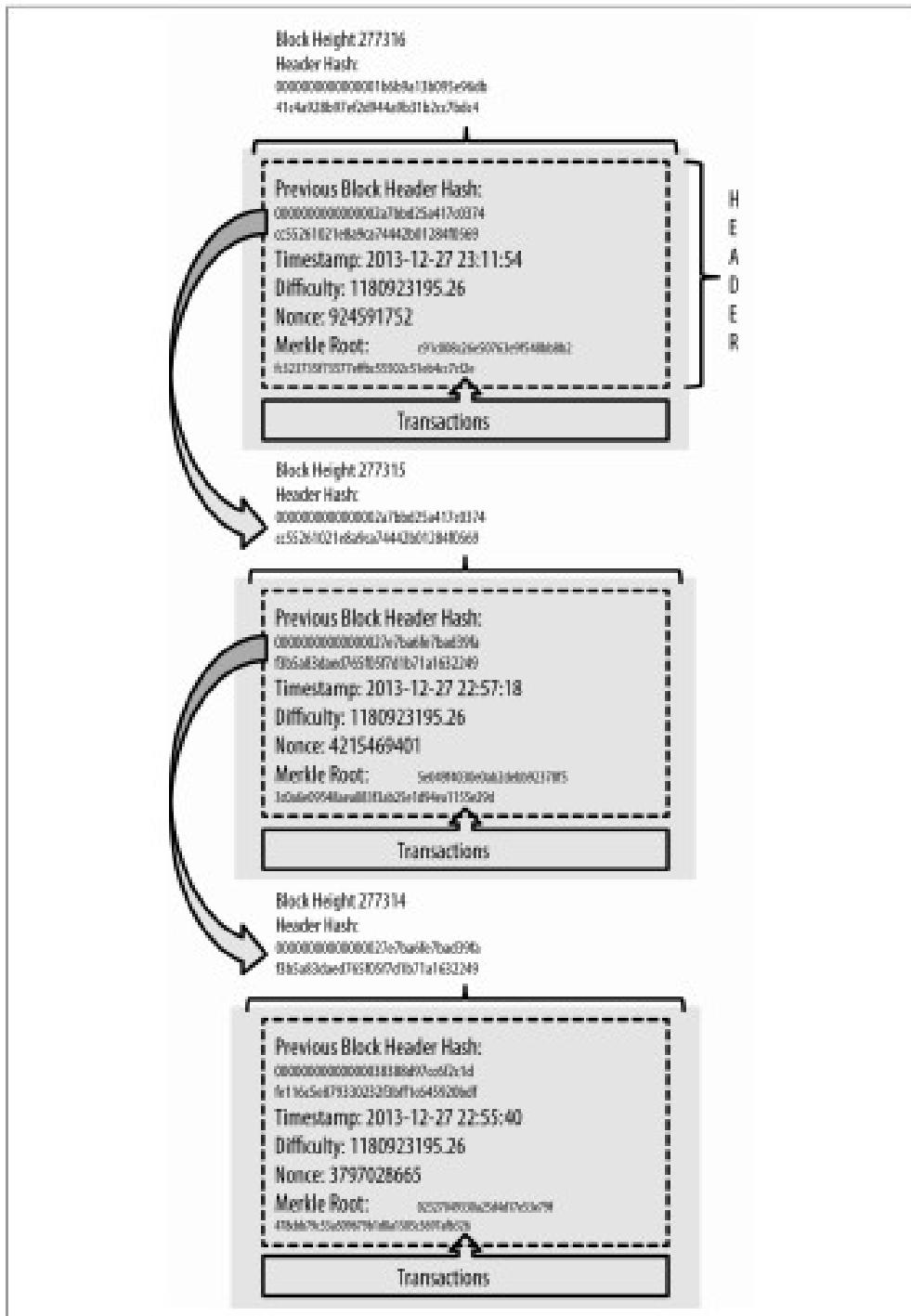
Drzewko Merkla

Każdy blok w łańcuchu blokowym bitcoin zawiera podsumowanie wszystkich transakcji w bloku, używając drzewka Merklego.

Te drzewka, znane również jako binarne drzewa hash, to struktury danych wykorzystywane do skutecznej syntezы i weryfikacji integralności dużych zbiorów danych. Drzewa hash są drzewami binarnymi zawierającymi kryptograficzne skróty. Określenie "drzewo" jest używane w informatyce do opisania rozgałęzionej struktury danych, ale drzewa te są zwykle wyświetlane góry nogami "korzeń" u góry i "liście" w dolnej części diagramu, jak widać w następujących przykładach.

Drzewka Merklego są wykorzystywane w Bitcoin w celu podsumowania wszystkich transakcji w bloku, tworząc ogólny cyfrowy odcisk palca całego zestawu transakcji, zapewniając bardzo skuteczny sposób na sprawdzenie, czy transakcja jest zawarta w bloku. Drzewo hash konstruuje się przez rekurencyjnie mieszanie pary węzłów, dopóki będzie tylko jeden hash, zwany głównym korzeniem lub korzeniem Merklego. Algorytmem hasha kryptograficznego, który używany przez drzewo merkle jest SHA256 zastosowany dwukrotnie, znany również jako dwukrotne SHA256.

Gdy elementy danych N są zakodowane i zsumowane w drzewo Merkle, można sprawdzić, czy każdy jeden element danych jest zawarty w drzewie z co najwyżej $2 * \log(N)$ obliczeń, co sprawia, że struktura danych jest bardzo wydajna.



7-

Rysunek
1. Bloki

połączone w łańcuch, poprzez odniesienie do hashu nagłówka bloku poprzedniego.

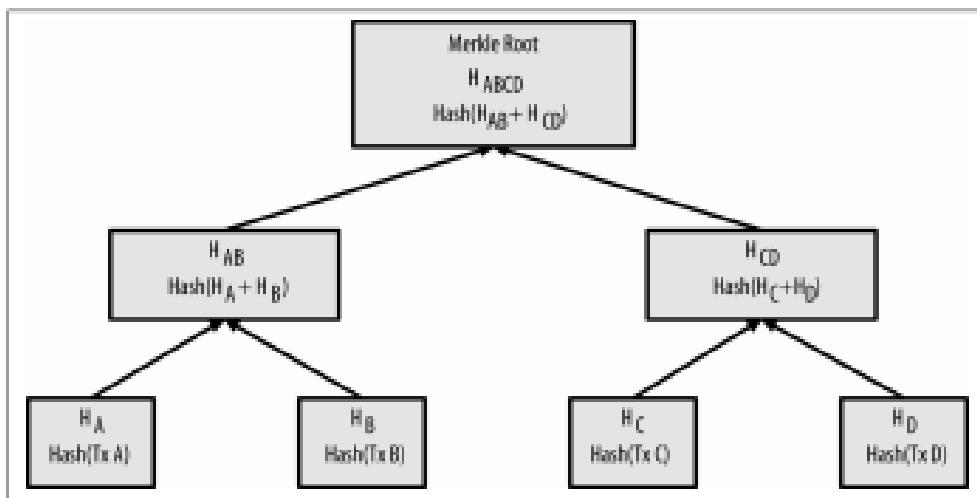
Drzewko Merklego jest konstruowane oddolnie. W poniższym przykładzie zaczynamy od czterech transakcji, A, B, C i D, które tworzą liście drzewa Merkle, jak pokazano na Figurze 7-2. Operacje te nie są przechowywane w drzewach Merkle; ich dane są skracane a otrzymany hash jest przechowywany w każdym węźle liści jako H_A , H_B , H_C i H_D :

$$H \sim A \sim = \text{SHA256}(\text{SHA256}(\text{transakcja } A)) .$$

Kolejne pary węzłów liściowych są następnie spięte w węźle macierzystym, przez złączenie dwóch hashów i zmieszanie ich ze sobą. Na przykład, aby zbudować węzeł macierzysty H ciąg jest dwukrotnie mieszany by wyprodukować hash węzła macierzystego dwa 32-bajtowe hashe dzieci są łączone, aby utworzyć ciąg 64-bajtowy.

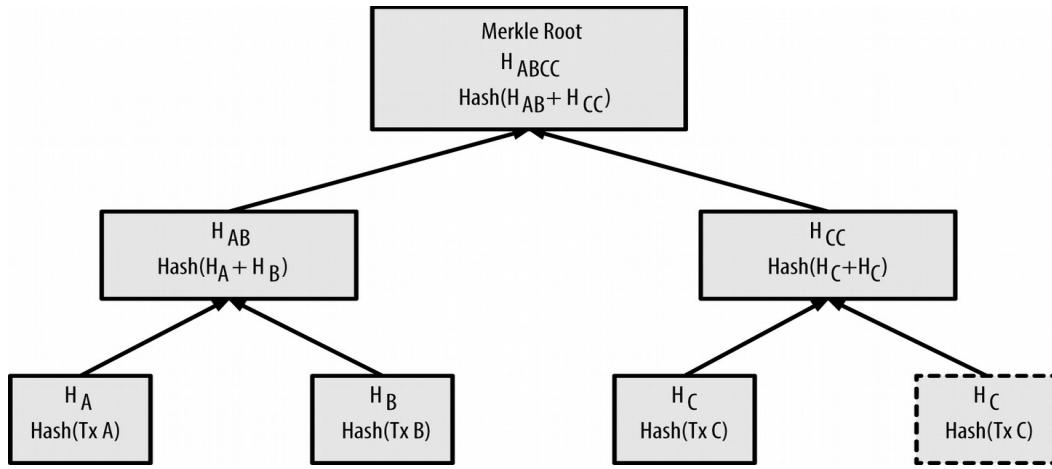
$$H \sim AB = \text{SHA256} (H \sim A \sim + H \sim B \sim)$$

Proces trwa dopóki tylko jeden węzeł jest u góry, węzeł znany jako korzeń Merkle. Ten 32-bajtowy jest przechowywany w nagłówku bloku i zestawiający wszystkie dane wszystkich czterech operacji.



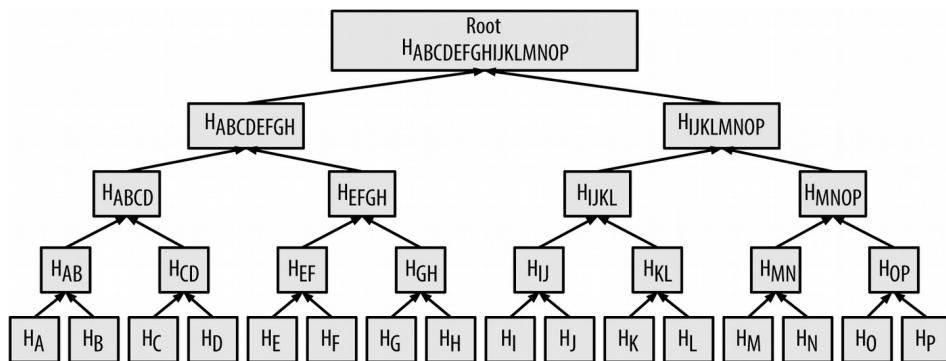
Rysunek 7-2. Obliczanie węzłów w drzewie skrótów

Ponieważ drzewo skrótów jest drzewem binarnym potrzebna jest parzysta liczba węzłów liściowych. Jeśli liczba transakcji do zsumowania jest nieparzysta, ostatni hash transakcji będzie duplikowany w celu stworzenia parzystej liczby węzłów liściowych, znanej również jako zrównoważenie drzewa. Jest to pokazane na rysunku 7-3, gdzie Transakcja C jest duplikowana.



Rysunek 7-3. Powielenie jednego elementu danych dla osiągnięcia parzystej liczby elementów danych

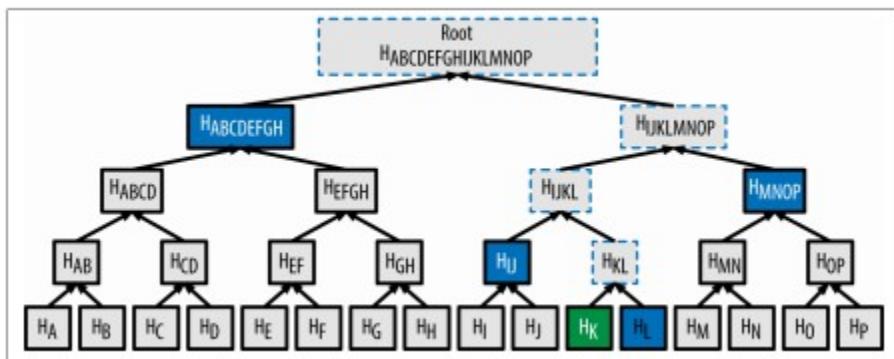
Ta sama metoda konstruowania drzewa z czterech transakcji może być zastosowana do konstrukcji drzewa o dowolnej wielkości. W sieci Bitcoin jest powszechnie posiadane od kilkuset do ponad tysiąca transakcji w jednym bloku, które są przedstawione w taki sam sposób, tworząc zaledwie 32 bajtów danych jako pojedynczy korzeń skracający. Na rysunku 7-4, widać drzewo zbudowane z 16 transakcji. Należy zauważyć, że chociaż korzeń wygląda na większy niż węzły liściowe na schemacie, jest dokładnie takiego samego rozmiaru - 32 bajtów. Czy istnieje jedna transakcja lub sto tysięcy transakcji w bloku, korzeń skracający zawsze podsumowuje je do 32 bajtów.



Rysunek 7-4. Drzewo hash (skrótów) podsumowując wiele elementów danych

Aby udowodnić, że dana transakcja jest zamieszczona w bloku, węzeł musi tylko produkować log (N) 32-bajtowych skrótów, stanowiących ścieżkę uwierzytelniania lub ścieżkę skrótów łączącą specyfikę transakcji do korzenia drzewa. Jest to szczególnie ważne, jeśli liczba transakcji wzrasta, ponieważ baza-2 logarytmu liczby transakcji zwiększa się znacznie wolniej. Pozwala to węzłom Bitcoin efektywnie produkować ścieżki złożone z 10 lub 12 hashów (skrótów) (320-384 bajtów), które mogą dostarczyć dowód pojedynczej transakcji spośród ponad tysiąca transakcji w bloku o wielkości megabajta.

Na Rysunku 7-5, węzeł może udowodnić, że transakcja K jest zawarta w bloku produkując skróconą ścieżkę, która jest złożona tylko z czterech 32-bajtowy hashów (w sumie 128 bajtów). Ścieżka składa się z czterech hashów – skrótów (zaznaczonych na niebiesko na rysunku 7-5) H_L , H_{IJ} , H_{MNOP} and $H_{ABCDEFGH}$ cztery skróty są przewidziane jako ścieżka uwierzytelniania, każdy węzeł może udowodnić, że H (zaznaczone na zielono na wykresie) jest zawarte w korzeniu skrótu (Merkle) poprzez wyliczenie czterech dodatkowych par mieszanych w diagramie H_K . Z tymi H_{KL} , H_{IJKL} , $H_{IJKLMNOP}$ i korzeniem drzewa Merkle (przedstawionych linią przerywaną)



Rysunek 7-5. Skrót ścieżki wykorzystywany jest do udowodnienia włączenia elementu danych.

Kod w przykładzie 7-1 pokazuje proces tworzenia drzewa skrótów z hashów węzłów liściowych do korzenia, przy użyciu biblioteki libbitcoin dla niektórych funkcji pomocniczych.

Przykład 7 – 1 Budowa drzewa skrótów

```

#include <bitcoin/bitcoin.hpp>

bc::hash_digest create_merkle(bc::hash_digest_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_digest_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::make_serializer(concat_data.begin());
            concat.write_hash(*it);
            concat.write_hash(*(it + 1));
            assert(concat.iterator() == concat_data.end());
            // 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_hex(hash) << std::endl;
        std::cout << std::endl;
        // -----
    }
    // Finally we end up with a single item.
    return merkle[0];
}

int main()
{
    // Replace these hashes with ones from a block to reproduce the same merkle
    root.
    bc::hash_digest_list tx_hashes{
        bc::de
        code_hash("000000000000000000000000000000000000000000000000000000000000000000000000"),
        bc::de
        code_hash("000000000000000000000000000000000000000000000000000000000000000000000001"),
        bc::de
        code_hash("000000000000000000000000000000000000000000000000000000000000000000000002"),
    };
    const bc::hash_digest merkle_root = create_merkle(tx_hashes);
    std::cout << "Result: " << bc::encode_hex(merkle_root) << std::endl;
    return 0;
}

```

Przykład 7-2 pokazuje wynik kompilacji i uruchamiania kodu skrótowego. Przykład 7-2. Kompilacja i uruchomienie przykładowego kodu skrótowego (Merkle)

Example 7-2 shows the result of compiling and running the merkle code.

Example 7-2. Compiling and running the merkle example code

```
$ # Compile the merkle.cpp code
$ g++ -o merkle merkle.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Run the merkle executable
$ ./merkle
Current merkle hash list:
32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d27886
38861db96905c8dc8b99398ca1cd5bd5b84ac3264a4e1b3e65afa1bcee7548c4

Current merkle hash list:
d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22184fb105014ba3

Result: d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3
```

Liczba transakcji	Przybliżony rozmiar bloków	Rozmiar ścieżki – hashe	Rozmiar ścieżki – bajty
16 transakcji	4 kilobajtów	4 hashy	128 bajtów
512 transakcji	128 kilobajtów	9hashy	288 bajtów
2048 transakcji	512 kilobajtów	11 hashy	352 bajty
65,535 transakcji	16 megabajtów	16 haszy	512 bajtów

Jak widać z tabeli, kiedy rozmiar bloku zwiększa się gwałtownie od 4 KB z 16 transakcji do wielkości bloku 16 MB złożonego z 65.535 transakcji, Merkle (skrót) ścieżki wymagany do udowodnienia włączenia transakcji wzrasta znacznie wolniej od 128 bajtów do zaledwie 512 bajtów. Z drzew Merkle, węzeł może pobrać tylko nagłówki bloku (80 bajtów na blok) i nadal być w stanie zidentyfikować włączenie transakcji do bloku pobierając niewielką ścieżkę skrótu (Merkle) z pełnego węzła bez przechowywania lub przekazywania zdecydowanej większości łańcucha bloków, nawet kilku gigabajtów. Węzły, które nie utrzymują pełnego łańcucha, zwane uproszczoną weryfikacją płatności (węzłami SPV), używają ścieżki skrótu Merkle do weryfikacji transakcji bez pobierania pełnych bloków.

Drzewka skrótów (Merklego) i SPV

Drzewka skrótów są szeroko wykorzystywane przez węzły SPV. Węzły te nie zawierają wszystkich transakcji i nie pobierają pełnych bloków, tylko nagłówki bloków. Aby zweryfikować czy dana transakcja znajduje się w bloku bez konieczności pobierania wszystkich transakcji należących do bloku wykorzystują ścieżkę autentykacji, czyli ścieżkę Merklego.

Weźmy na przykład, węzeł SPV, który jest zainteresowany płatnościami przychodząymi na adres zawarty w jego portfelu. Węzeł SPV ustali filtr bloom na jego połączenie z peerami by ograniczyć transakcje tylko do tych zawierających zainteresowane adresy. Gdy peer widzi transakcję która łączy filtr bloom, to wysyła ten blok za pomocą wiadomości merkleblock. Komunikat merkleblock zawiera nagłówek bloku, jak również ścieżkę, która łączy transakcję z zainteresowanym korzeniem merkle w bloku. Węzeł SPV może

korzystać z tej ścieżki, aby połączyć transakcję z blokiem i sprawdzić, że transakcja jest zawarta w bloku. Węzeł SPV wykorzystuje również nagłówek bloku do połączenia bloku z resztą łańcucha bloku. Połączenie tych dwóch powiązań - między transakcją a blokiem i pomiędzy blokiem i łańcuchem bloków dowodzi, że transakcja jest rejestrowana w łańcuchu blokowym. W sumie, węzeł SPV będzie otrzymywał mniej niż kilobajt danych dla nagłówka bloku i ścieżki Merkle, suma danych, która jest więcej niż tysiąc razy mniejsza niż w pełnym bloku (obecnie koło 1 megabajta).`

Rozdział 8

Górnictwo i konsensus

Wstęp

Górnictwo to proces, w którym nowy bitcoin jest dodawany do podaży pieniądza. Górnictwo służy również do zabezpieczenia systemu Bitcoin przed oszustwami transakcyjnymi lub transakcjami wydającymi tę samą ilość Bitcoin więcej niż jeden raz, znanych jako dwukrotnie wydające. Górnicy zapewniają zasilanie sieci bitcoin w zamian za możliwość nagrody Bitcoin.

Górnicy walidują nowe transakcje oraz ich zapis w globalnej księdze. Nowy blok zawierający transakcje, które nastąpiły od ostatniej blokady, jest "wydobywany" co 10 minut, dodając tym samym te transakcje do łańcucha bloków. Transakcje, które stają się częścią bloku i są dodane do łańcucha bloków są uważane za "potwierzone", co pozwala nowym właścicielom Bitcoin wydać Bitcoin które otrzymali w tych transakcjach.

Górnicy otrzymują dwa rodzaje nagród: nowe monety utworzone w każdym nowym bloku i opłaty transakcyjne ze wszystkich transakcji zawartych w bloku. Aby zdobyć tę nagrodę, górnicy konkurują ze sobą, aby rozwiązać trudny problem matematyczny na podstawie algorytmu hasha kryptograficznego. Rozwiążanie tego problemu, zwanego dowodem pracy, jest zawarte w nowym bloku i działa jako dowód, że górnik włoży znaczny wysiłek w obliczenia. Konkurs na rozwiązanie algorytmu proof-of-pracy, aby zdobyć nagrodę oraz prawo do rejestrowania transakcji na łańcuchu jest podstawą modelu zabezpieczeń Bitcoin.

Proces generacji nowej monety nazywamy wydobyciem, ponieważ nagroda jest zaprojektowana do symulacji malejących przychodów, podobnie jak wydobycie metali szlachetnych. Podaż pieniądza Bitcoin jest tworzona poprzez wydobycie, tak jak bank centralny wydaje nowe pieniądze drukując banknoty. Kwota nowo utworzonych Bitcoinów górnik może dodać do bloku, który zmniejsza się mniej więcej co cztery lata (lub dokładnie co 210.000 bloki). Zaczęło się od 50 Bitcoinów w bloku w styczniu 2009 roku i zmniejszyło się o połowę do 25 Bitcoin na bloku w listopadzie 2012. Kolejny podział nastąpi w 2016 roku, do 12,5 Bitcoin. Na podstawie tego wzoru, nagrody za wydobycie zmniejszają się wykładniczo, aż około roku 2140, kiedy wszystkie Bitcoin (+20,99999998 mln) zostaną wydane. Po 2140, żadne nowe bitcoins nie będą wydawane.

Bitcoin górnicy również zarobić opłat od transakcji. Każda transakcja może obejmować opłaty transakcji, w postaci nadwyżki Bitcoin pomiędzy wejściami tą transakcją i uliczne put. Zwycięski Bitcoin górnik dostaje się "zachować zmiany" w transakcjach zawartych w zwycięskiego bloku. Obecnie opłaty stanowią 0,5% lub mniej przychodu Bitcoin górnika, zdecydowana większość pochodzi z nowo wybitych bitcoins. Jednak, jak nagroda decreases w czasie, a liczba transakcji na wzrost blokowych, większa część zysków Bitcoin górniczych będą pochodzić z opłat. Po 2140, wszelkie zarobki górnika będą w formie opłat transakcyjnych.

Słowo "wydobycie" jest nieco mylące. Poprzez wyobrażenie wydobycia metali szlachetnych, skupiamy naszą uwagę na nagrodzie dla górnictwa, nowych bitcoinach w każdym bloku. Chociaż wydobycie jest motywowane przez tę nagrodę, głównym celem wydobycia nie jest nagroda lub generowanie nowych monet. Jeśli widzisz górnictwo tylko jako proces, w którym tworzone są monety, jesteś mylisz znaczenia

(zachęty) celu procesu. Górnictwo jest głównym procesem zdecentralizowanym biura informacyjnego, w którym transakcje zostają zatwierdzone i rozliczone. Górnictwo zabezpiecza system Bitcoin i umożliwia powstawanie zgody całej sieci bez centralnego organu.

Górnictwo to wynalazek, który sprawia, że sieć Bitcoin jest wyjątkowa, bo ma zdecentralizowany mechanizm bezpieczeństwa, który jest podstawą dla transakcji peer-to-peer w gotówce cyfrowej. Nagroda w postaci nowo wybitych monet i opłat transakcyjnych jest programem motywacyjnym, który stymuluje działania górników w celu zabezpieczenia sieci, przy jednoczesnej realizacji dostaw pieniężnej.

W tym rozdziale będziemy najpierw badać górnictwo jako mechanizm pieniężnych dostaw, a następnie spojrzymy na najważniejsze funkcje wydobycie: zdecentralizowanie powstającego mechanizmu konsensusu, który stanowi podstawę bezpieczeństwa Bitcoin.

Ekonomia Bitcoin i tworzenie waluty

Bitcoins są "wybite" podczas tworzenia każdego bloku w stałym oraz malejącym tempie.

Każdy blok, tworzony średnio co 10 minut, zawiera zupełnie nowe bitcoiny tworzone z niczego. Co 210.000 bloków, albo raz na cztery lata, emisja kursu zmniejsza się o 50%. W ciągu pierwszych czterech lat funkcjonowania systemu sieci, każdy blok zawierał 50 nowych bitcoinów.

W listopadzie 2012 roku, kurs emisji nowych Bitcoinów został obniżony do 25 bitcoinów na blok i będzie zmniejszać się ponownie na 12,5 bitcoinów w bloku 420.000, który będzie wydobyty w 2016 roku. Wskaźnik nowych monet maleje w ciągu wykładniczym zmniejszając się ponad 64 razy dopóki blok 13,230,000 (wydobywa się mniej więcej 2137 rocznie), osiągnie minimum jednostkowe 1 Satoshi. W końcu, po 13,44 milionach bloków w przybliżeniu 2140, wszystkie 2,099,999,997,690,000 satoshi, czyli prawie 21 milionów bitcoinów, będą wydane. Następnie bloki nie będą zawierały żadnych nowych bitcoinów i górnicy będą nagradzani wyłącznie za pośrednictwem opłat transakcyjnych. Rysunek 8-1 pokazuje całkowity obieg Bitcoinów w czasie w którym emisja monet maleje.

W przykładowym kodzie w [Przykładzie 8-1](#), możemy obliczyć całkowitą ilość Bitcoinów, które będą wydane.

Przykład 8-1 – skrypt do obliczania całkowitej ilości Bitcoinów, które będą wydane.

Example 8-1. A script for calculating how much total bitcoin will be issued

```
# Original block reward for miners was 50 BTC
start_block_reward = 50
# 210000 is around every 4 years with a 10 minute block interval
reward_interval = 210000

def max_money():
    # 50 BTC = 50 0000 0000 Satoshis
    current_reward = 50 * 10**8
    total = 0
    while current_reward > 0:
        total += reward_interval * current_reward
        current_reward /= 2
    return total

print "Total BTC to ever be created:", max_money(), "Satoshis"
```

Przykład 8-2 pokazuje blok wyjściowy produkowane przez uwalnianie skryptu

[Example 8-2](#) shows the output produced by running this script.

Example 8-2. Running the max_money.py script

```
$ python max_money.py  
Total BTC to ever be created: 209999997690000 Satoshi
```



Figure 8-1. Supply of bitcoin currency over time based on a geometrically decreasing issuance rate

Kończąca

się i malejąca

emisja tworzy stały dopływ pieniężny, który jest odporny na inflację. W przeciwieństwie do waluty fiduciarnej, którą można wydrukować w nieskończoność przez bank centralny, Bitcoin może nie być napompowane przez drukowanie pieniędzy.

Waluta deflacyjna

Najważniejszymi i najbardziej omawianymi konsekwencjami stale zmniejszającej się emisji pieniężnej jest to, że waluta będzie miała tendencje do deflacji. Deflacja jest zjawiskiem aprecjacji wartości ze względu na niedopasowanie podaży i popytu, który napędza wartość (i) kurs danej waluty. W przeciwieństwie do inflacji, deflacja cen oznacza, że pieniądze mają większą siłę nabywczą. Wielu ekonomistów twierdzi, że deflacyjna gospodarka jest katastrofą, że należy unikać jej za wszelką cenę. To dlatego, że w okresie szybkiej deflacji, ludzie mają tendencję do gromadzenia pieniędzy zamiast wydawać je w nadziei, że ceny spadną. Takie zjawisko rozłożono w czasie japońskiej "straconej dekady", kiedy całkowite załamanie popytu pchnęło walutę na spiralę deflacyjną.

Eksperci Bitcoin twierdzą, że deflacja nie sama z siebie jest zła. Przeciwnie, deflacja jest związany z załamaniem popytu, ponieważ jest to jedynym przykładem deflacji, który musimy przestudiować. W walucie FIAT z możliwością nieograniczonego druku, bardzo trudno jest wprowadzić spiralę deflacyjną, chyba że nastąpi kompletne załamanie popytu i głęboka niechęć do drukowania pieniędzy. Deflacja w Bitcoin nie jest spowodowana załamaniem popytu, lecz przewidywaniem ograniczonej podaży.

W praktyce stało się oczywiste, że instynkt gromadzenia spowodowany deflacją waluty można przewyciążyć poprzez zniżkę nadaną przez sprzedawców, aż zniżki wpłyną na instynkt kupującego. Ponieważ sprzedający jest również zmotywowany do gromadzenia, rabat staje się zrównoważeniem ceny, w której oba skarbowe instynkty są dopasowane. Z 30% zniżką na cenę bitcoin, większość sprzedawców Bitcoin nie ma problemów z przewyciążeniem skarbowego instynktu i generowania przychodów. To pokazuje, czy aspekt deflacji w walucie jest rzeczywistym problemem, gdy nie jest napędzany gwałtownymi spadkami gospodarczymi.

Consensus decentralizacji

W poprzednim rozdziale przyjrzaliśmy się łańcuchowi bloków, globalnej księdze publicznej (liście) wszystkich transakcji, które wszyscy w sieci bitcoin przyjmują jako autorytatywne zapisy własności.

Ale jak wszyscy w sieci zgadzają się na jedną uniwersalną "prawdę" o tym, kto co posiada, bez konieczności zaufania komukolwiek? Wszystkie tradycyjne systemy płatności zależą od modelu zaufania, który ma władzę centralną świadczącą usługi izb rozrachunkowych, w zasadzie weryfikując i czyszcząc wszystkie transakcje. Bitcoin nie ma władzy centralnej, ale jakoś każdy pełny węzeł posiada kompletną kopię księgi publicznej i może zaufać jako autorytatywnemu zapisowi. łańcuch bloków nie jest tworzony przez władze centralne, ale jest montowany niezależnie przez każdy węzeł sieci. Niekiedy każdy węzeł w sieci, działając na informacjach przekazywanych przez niebezpieczne połączenia sieciowe może dojść w do tych samych wniosków i zamontować kopię tej samej księgi publicznej jak każdy inny. Ten rozdział analizuje proces, w którym sieć Bitcoin osiąga globalny konsensus bez organu centralnego.

Głównym wynalazkiem Satoshi Nakamoto jest zdecentralizowany mechanizm dla powstającego konsensusu. Wyłoni się, ponieważ porozumienie nie zostanie osiągnięte wprost-nie ma wyborów lub stałego momentu wystąpienia konsensusu. Zamiast tego, jest on efektem wyłaniającym się z asynchronicznych interakcji tysięcy niezależnych węzłów, wszystkich podążających prostymi zasadami. Wszystkie właściwości Bitcoin, w tym waluty transakcji, płatności oraz model bezpieczeństwa, który nie zależy od organu centralnego lub zaufania, wynikają z tego wynalazku.

Zdecentralizowany konsensus bitcoina wyłania się z interakcji czterech procesów, które występują niezależnie od węzłów w całej sieci:

- Niezależna weryfikacja każdej transakcji, przez każdy pełny węzeł, w oparciu o kompleksową listę kryteriów
- Niezależna agregacja tych transakcji w nowe bloki przez węzły górnicze, połączone z wykazanymi obliczeniami przez algorytm proof-of-work
 - Niezależna weryfikacja nowych bloków przez każdy węzeł i montaż w łańcuchu
 - Niezależny wybór, przez każdy węzeł sieci z najbardziej skumulowanymi obliczeniami wykazanymi poprzez dowód pracy

W kilku następnych częściach będziemy badać te procesy i jak współdziałają one tworząc wschodząca własność konsensusu całej sieci, który pozwala każdemu węzłowi Bitcoin na montaż własnej kopii autorytatywnej, zaufanej, publicznej, globalnej księgi.

Niezależna weryfikacja transakcji

W [Rozdziale 5](#), widzieliśmy, jak oprogramowanie portfela tworzy transakcje poprzez zbieranie UTXO, zapewniając odpowiednie odblokowania skryptów, a następnie budowę nowych wyjść przypisanych do nowego właściciela. W rezultacie transakcje są następnie wysyłane do sąsiednich węzłów w sieci Bitcoin tak, że mogą być rozpowszechniane w całej sieci bitcoin.

Jednak przed przekazaniem transakcji do jej sąsiadów, każdy węzeł, który odbiera transakcję najpierw zweryfikuje transakcję. To gwarantuje, że tylko prawidłowe transakcje będą propagowane w całej sieci, a

nieprawidłowe transakcje są odrzucane na w kontakcie z pierwszym węzłem.

Każdy węzeł sprawdza każdą transakcję zwracając uwagę na długą kontrolną listę kryteriów:

- Składnia tej transakcji i jej struktury danych muszą być poprawne.
- Ani wykazy wejścia ani wyjścia nie są puste.
- Wielkość transakcji w bajtach jest mniejsza niż MAX_BLOCK_SIZE.
- Każda wartość wyjściowa, a także całkowita, musi mieścić się w dopuszczalnym przedziale wartości (mniej niż 21 milionów monet, więcej niż 0).
- Żaden z wejść nie mają hasha = 0, N = -1 (transakcje bazowych monet nie powinny być przekazywane).
 - nLockTime jest mniejszy lub równy do INT_MAX.
 - wielkość transakcji w bajtach jest większa niż lub równa 100.
 - Liczba operacji podpisu zawarte w transakcjach jest mniejsza niż limit operacji podpisu.
 - Skrypt odblokowujący (scriptSig) może wypchnąć tylko liczby na stos, a skrypt blokujący (scriptPubkey) musi pasować formą isStandard (ta odrzuca "nietypowe" transakcje).
 - Transakcja dopasowania w puli czy w bloku w głównej gałęzi, musi istnieć.
 - Dla każdego wejścia, jeżeli powołane blok wyjściowy istnieje w jakiekolwiek innej transakcji w puli, transakcja musi zostać odrzucona.
 - Dla każdego wejścia należy szukać w głównej gałęzi i puli transakcji wyjścia. Jeśli wyjściu transakcji jest pominięte przez jakiekolwiek wejście ta transakcja będzie sierotą. Należy dodać je do zbiornika transakcji sieroczych, jeśli dopasowanie nie znalazło się w puli.
 - Dla każdego wejścia, jeżeli powołanym blok wyjściowy m jest blok wyjściowy coinbase, jest obowiązkowe co najmniej jedno zatwierdzenie COINBASE_MATURITY (100).
 - Dla każdego wejścia, blok wyjściowy musi istnieć i nie może być już wydane.
 - Używanie odpowiedniego bloku wyjściowego transakcji do uzyskania wartości wejściowej, tak samo jak sumy, jest zezwoloną rangą wartości (mniej niż 21 mln monet, więcej od 0).
 - Odrzucenie, gdy suma wartości wejściowych jest mniejsza od sumy wartości wyjściowych.
 - Odrzucenie, jeśli opłata transakcyjna będzie zbyt niska, aby dostać się do pustego bloku.
 - odblokowujący skrypt dla każdego z wejść musi weryfikować korespondującą skrypt blokujący wejście.

Te warunki mogą być widoczne w szczegółach w funkcjach AcceptToMemoryPool, transakcjach czekowych i czekowych wejściach w odniesieniu do klienta Bitcoin. Należy pamiętać, że warunki zmieniają się w czasie, uwzględniając nowe rodzaje ataków denial-of-service a czasem go łagodząc przepisy w taki sposób, aby zwiększyć liczbę rodzajów transakcji.

Przez niezależną weryfikację każdej transakcji po jej otrzymaniu i przed puszczeniem jej dalej każdy węzeł tworzy zbiór ważnych nowych transakcji (zbiornik transakcji), z grubsza w tej samej kolejności.

Węzły górnicze (wydobywcze)

Niektóre z węzłów sieci bitcoin wyspecjalizowanymi węzłami zwany górnymi. W rozdziale 1 wprowadziliśmy Jingq, studenta inżynierii komputerowej w Szanghaju w Chinach, który jest górnikiem Bitcoin. Jing zarabia Bitcoiny uruchamiając "platformę górną", która jest wyspecjalizowanym komputerowym systemem sprzętowym przeznaczonym do kopania bitcoinów. Sprzęt Jinga jest połączony do serwera z systemem pełnych węzłów Bitcoin. W odróżnieniu od Jinga, niektórzy górnicy

kopią bez pełnego węzła, jak to widzimy w "Pulach górniczych". Podobnie jak każdy inny pełny, węzeł Jinga odbiera i przekazuje transakcje niepotwierdzone w sieci bitcoin. Węzeł Jinga jednak agreguje te transakcje w nowe bloki.

Węzeł Jinga wysłuchuje nowych bloków, propagowanych w sieci bitcoin, jak zresztą robią wszystkie węzły, jednak pojawienie się nowego bloku ma szczególne znaczenie dla węzła górnictwa.

Konkurencja wśród górników skutecznie kończy się z rozprzestrzenieniem nowego bloku, który działa jak ogłoszenie zwycięzcy. Dla górników, otrzymanie nowych bloków blokowe oznacza, że ktoś wygrał a ktoś przegrał rywalizację. Jednak koniec jednej rundy jest także początkiem następnej rundy. Nowy blok nie jest po prostu w flagą w czarno-białą kratę znaczącą koniec wyścigu; jest to również pistolet startowy w wyścigu po następnym bloku.

Agregacja transakcji na bloki

Po sprawdzeniu transakcji, węzeł Bitcoin doda je do zbiornika pamięci lub zbiornika transakcji, w którym transakcje czekają aż będą mogły być uwzględnione (wydobyte) do bloku. Węzeł Jinga zbiera, weryfikuje i przekazuje nowe transakcje, podobnie jak każdy inny węzeł. W przeciwieństwie do innych węzłów, węzeł Jinga jednak będzie następnie agregował te transakcje do bloków kandydujących.

Prześledźmy bloki, które zostały utworzone w czasie gdy Alicja kupowała filiżankę kawy w kawiarni Boba (patrz "Zakup filiżance kawy" na stronie 16). Transakcja Alicji została zawarta w bloku 277,316. W celu wykazania pojęć zawartych w tym rozdziale, założymy, że blok był wydobywany przez system górnictwa Jinga i następująca transakcja Alicji stała się częścią tego nowego bloku.

Węzeł wydobywczy Jinga utrzymuje lokalną kopię łańcucha bloków, listę wszystkich bloków utworzonych od początku układu bitcoin w roku 2009. W czasie kiedy Alicja kupuje kawę, węzeł Jinga zebrał łańcuch aż do bloku 277,314. Węzeł Jinga słucha transakcji, starając się wydobyć nowy blok, a także słucha za blokami odkrytymi przez inne węzły. Ponieważ węzeł Jinga jest węzłem wydobywającym, odbiera blok 277,315 za pośrednictwem sieci bitcoin. Pojawienie się tego bloku oznacza koniec konkurencji dla bloku 277,315 i na początku zawodów w tworzeniu bloku 277,316.

W trakcie poprzednich 10 minut, podczas gdy węzeł Jinga szukał ozwijania do bloku 277,315, gromadzono też transakcje w ramach przygotowania następnego bloku. Do tej pory zebrano kilkaset transakcji w puli pamięci. Po otrzymaniu bloku 277,315 i sprawdzeniu go, węzeł Jinga będzie również sprawdzać wszystkie transakcje w puli pamięci i usunie te, które zostały zawarte w bloku 277,315. Transakcje pozostawione w puli pamięci są niepotwierdzone i czekają, na rejestracje w nowym bloku.

Węzeł Jinga natychmiast tworzy nowy, pusty blok, kandydata do bloku 277,316. Ten blok jest nazywana blokiem kandydatów, ponieważ nie jest jeszcze ważnym blokiem, ponieważ nie zawiera ważnego potwierdzenia pracy. Blok staje się ważny tylko wtedy, gdy górnikowi uda się znaleźć rozwiązanie algorytmu proof-of-work.

Data transakcji, opłaty i priorytet

Aby zbudować blok kandydata, węzeł Jinga za wybiera transakcję ze zbiornika pamięci stosując priorytet metryki dla każdej transakcji i dodając najpierw najwyższy priorytet transakcji. Transakcje są traktowane priorytetowo w oparciu o "wiek" UTXO, które są wydawane w ich wejściach, pozwalając starym i wysokowartościowym wejściom na bycie priorytetowym przez nowsze i mniejsze wejścia. Priorytetowe transakcje mogą być wysyłane bez żadnych opłat, jeśli w bloku jest wystarczająco dużo miejsca.

Priorytet transakcji jest obliczany jako suma wartości i wieku wejścia podzielony przez całkowitą wielkość transakcji:

$$\text{Priorytet} = \frac{\text{suma (wartość wejścia * Wiek wejście)}}{\text{wielkość transakcji}}$$

W tym równaniu, wartość wejścia mierzy się w stacji bazowej, satoshi (1 / 100m Bitcoin). Wiek z UTXO jest liczbą bloków, które upłyнуły od momentu, kiedy UTXO zostało nagrane na łańcuchu blokowym, mierząc ile bloków "zagłębiło" się w łańcuchu. Wielkość transakcji jest mierzona w bajtach.

Dla transakcji uznanych za "wysoki priorytet" priorytet musi być większy niż 57.600.000, co odpowiada jednemu Bitcoinowi (100m satoshi) w wieku jednego dnia (144 bloków), w transakcji w całkowitym rozmiarze 250 bajtów.

$$\text{Wysoki priorytet} > 100,000,000 \text{ satoshi} * 144 \text{ blocks} / 250 \text{ bytes} = 57,600,000$$

Pierwsze 50 kilobajtów przestrzeni transakcyjnej w bloku jest przeznaczonych na transakcje o wysokim priorytecie. Węzeł Jinga będzie wypełnić pierwsze 50 kilobajtów, priorytetując najwyższe transakcje jako pierwsze, niezależnie od opłaty. To pozwala transakcjom o wysokim priorytecie być przetwarzanym, nawet jeśli wnoszą one zerowe opłaty.

Węzeł wydobywczy Jinga jest następnie wypełniany resztą bloków do maksymalnej wielkości bloków (`MAX_BLOCK_SIZE` w kodzie) z transakcji, które wprowadzą co najmniej minimalną opłatę z priorytetem najwyższej opłaty za kilobajtów transakcji.

Jeżeli istnieje jakakolwiek przestrzeń pozostała w bloku, węzeł Jinga może wybrać wypełnienie go z transakcji bez opłaty. Niektórzy górnicy wybierają transakcję bez opłat na zasadzie best-effort (najlepszego działania). Inni górnicy mogą zignorować transakcję bez opłat.

Wszelkie transakcje pozostały w puli pamięci, po tym jak przestrzeń bloku jest wypełniona, pozostanie w puli do czasu włączenia do następnego bloku. Ponieważ transakcje pozostają w puli pamięci, „wiek” wejścia, jak i UTXO dostają się głębiej w łańcuchu przez dodanie nowych bloków na wierzch. Ponieważ priorytet transakcji zależy od wieku jej wejścia transakcje pozostające w puli będą starsze, a i tym samym zwiększy im się priorytet. Ostatecznie transakcja niezawierająca opłat może osiągnąć wystarczająco wysoki priorytet, by być zamieszczona w bloku za darmo.

Transakcje Bitcoin nie mają ważności limitu czasu. Transakcja, która jest ważna teraz będzie ważna bezterminowo. Jednakże, jeśli transakcja jest rozprzestrzaniana przez sieć tylko jeden raz, to utrzymuje się

tylko tak długo, ile jest trzymana przez pamięć węzła wydobywczego. Kiedy węzeł wydobywczy zostanie ponownie uruchomiony, puli pamięci czyści się, ponieważ jest to przejściowa forma nietrwałego przechowywania. Chociaż ważne transakcje mogą być propagowane przez sieć, jeżeli nie są wykonywane, ostatecznie mogą nie dostać się do puli pamięci górnika. Oczekuje się, że oprogramowanie rozprowadzi takie transakcje lub odtworzy je z wyższymi opłatami, jeżeli nie są one z powodzeniem realizowane w rozsądny czasie.

Gdy węzeł Jinga zaagreguje wszystkie transakcje z puli pamięci, nowy blok-kandydat będzie miał 418 transakcji z całością opłat transakcyjnych równą 0.09094928 Bitcoin. Możesz zobaczyć ten blok w łańcuchu blokowym za pomocą interfejsu wiersza poleceń klienta BitcoinCore, jak pokazano w Przykładzie 8-3.

Example 8-3. Block 277,316

Transakcja generowania

Pierwsza transakcja dodana do bloku jest specjalną transakcją, zwaną transakcją generowania lub transakcją coinbase. Transakcja ta jest konstruowana przez węzeł Jinga i jest jego nagrodą za trud górniczej pracy. Węzeł Jinga stwarza transakcję generacji jako zapłatę dla własnego portfela: "Zapłać Jingowi na adres 25.09094928 Bitcoin." Łączna wysokość wynagrodzenia, które Jing zbiera z górnictwa bloku jest sumą nagrody coinbase (25 nowych Bitcoins) oraz opłat transakcyjnych (0.09094928) ze wszystkich transakcji zawartych w bloku, jak pokazano w Przykładzie 8-4: \$

Bitcoin-cli
d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f
getrawtransaction
1

Przykład 8-4. Generacja transakcji

Example 8-4. Generation transaction

W przeciwieństwie do zwykłych transakcji, transakcja generacji nie zużywają UT-XO jako wejścia. Zamiast tego ma tylko jedno wejście, zwany coinbase, który tworzy bitcoiny z niczego. Transakcja generacji ma jednego blok wyjściowy, wystawione na własny adres bitcoin górników. Blok wyjściowy transakcji generacji wysyła wartość 25.09094928 bitcoins na adres bitcoin górnika, w tym przypadku 1MxTkeEP2PmHSMze5tUZ1hAV3YT Ku2Gh1N.

Wynagrodzenie Coinbase i opłaty

Aby skonstruować transakcję generacji, węzeł Jinga pierwszy oblicza łączną wysokość opłat transakcyjnych przez dodanie wszystkich wejść i wyjść z transakcji 418, które zostały dodane do bloku. Opłaty są obliczane jako:

Wszystkie Opłaty = SUMA (Wejścia) - suma (wyjścia)

W bloku 277,316, łączne opłaty transakcyjne są równe 0.09094928 bitcoins. Następnie węzeł Jinga oblicza właściwą nagrodę dla nowego bloku. Nagrodą jest obliczana na podstawie wysokości bloku, zaczynając od 50 bitcoinów na 1 blok i zmniejsza się o połowę co 210.000 bloków. Ponieważ ten blok jest na wysokości 277,316, właściwa nagroda wynosi 25 bitcoins.

Obliczenia mogą być widoczne w funkcji GetBlockValue w kliencie Bitcoin Core, jak pokazano w [Przykładzie 8-5](#).

Przykład 8-5. Obliczanie nagrody – funkcja GetBlockValue - Bitcoin Core Client, main.cpp line (linia) 1305

```
int64_t GetBlockValue(int nHeight, int64_t nFees)
{
    int64_t nSubsidy = 50 * COIN;
    int halvings = nHeight / Params().SubsidyHalvingInterval();

    // Force block reward to zero when right shift is undefined.
    if (halvings >= 64)

        return nFees;

    // Subsidy is cut in half every 210,000 blocks which will occur approximately
    // every 4 years.
    nSubsidy >>= halvings;

    return nSubsidy + nFees;
}
```

Następnie oblicza się liczbę przepołowień, które wystąpiły dzieląc bieżącą wysokość bloku przez przedział połowiczny (SubsidyHalvingInterval). W przypadku bloku 277,316, z przerwą połowicznym co 210.000 bloków, wynik jest jednym przepołowieniem.

Maksymalna dozwolona liczba przepołowień jest równa 64, więc kod nakłada nagrody zerowe (wraca tylko opłata) jeśli ilość 64 przepołowień zostanie przekroczena.

Następnie funkcja używa binary-right-shift (binarnego przesunięcia w prawo), by podzielić nagrodę (nSubsidy) przez dwa w każdej rundzie zmniejszającej o połowę. W przypadku bloku 277,316, byłoby to przesunięciem binarnym w prawo nagrody w wysokości 5 mld satoshi (przepołowionej raz) co daje 2,5

mld satoshi lub 25 bitcoins. Operator binary-right-shift jest używany, ponieważ bardziej nadaje się do podziału przez dwa niż całkowite lub zmiennoprzecinkowe dzielenie.

Wreszcie, nagroda coinbase (`nSubsidy`) jest dodawana do opłat transakcyjnych (`nFees`) i suma jest zwracana.

Struktura transakcji generowania

Zgodnie z tymi kalkulacjami węzeł Jinga konstruuje transakcję generowania, by wypłacić sobie 25.09094928 bitcoinów.

Jak widać w [Przykładzie 8-4](#), transakcja generacyjna ma specjalny format. W przeciwieństwie do transakcji wprowadzającej określających wydawanie poprzedniego UTXO, ta transakcja ma wejście "coinbase". Zbadaliśmy bloki wejściowe transakcji w tabeli 5-3. Porównajmy regularne wejście transakcji z wejściem transakcji generacyjnej. Tabela 8-1 przedstawia strukturę regularnej transakcji, gdy [Tabela 8-2](#) przedstawia strukturę bloki wejściowe transakcji generacyjnej.

Rozmiar	Obszar	Opis
32 bajty	Hash transakcji	Wskaźnik do transakcji zawierającej UTXO, które ma być wydane
4 bajty	Wskaźnik wyjścia	Numer wskaźnikowy UTXO które ma być wydane, pierwsza liczba to 0
1-9 bajtów (VarInt)	Rozmiar skryptu odblokowującego	Rozmiar skryptu odblokowującego wyrażony w bajtach do zastosowania
zmienny	Skrypt odblokowujący	Skrypt, który spełnia warunki określone w skrypcie blokującym UTXO.
4 bajty	Numer sekwencji	Obecnie wyłączona funkcja wymiany Tx, ustawiona do 0xFFFFFFFF

W transakcji generacji, pierwsze dwa pola są ustawione na wartości, które nie stanowią odniesienia UTXO. Zamiast "Hasza transakcja", pierwsze pole jest wypełnione przez 32 bajtów wszystkich ustawionych na zero. "Indeks wyjścia" jest wypełniony przez 4 bajty ustawione na 0xFF (255 dziesiętne). "Skrypt odblokowania" zastępuje się danymi coinbase, arbitralnego pola danych wykorzystywanych przez górników.

Dane Coinbase

Transakcje generacyjne nie mają skryptu odblokowującego (a.k.a., `scriptSig`) pole. Zamiast tego, to pole jest zastąpione danymi coinbase, które muszą wynosić od 2 do 100 bajtów. Z wyjątkiem pierwszych kilku bajtach reszta danych coinbase może być wykorzystywane przez górników w dowolny żądany sposób; są to dane arbitralne.

W bloku genezy, na przykład, Satoshi Nakamoto dodaje tekst "The Times 03 / Jan / 2009Chancellor on brink of second bailout for banks" w danych coinbase, wykorzystując go jako dowód w terminie i

przekazując wiadomość. Obecnie górnicy wykorzystują dane coinbase by zatrzymać dodatkowe wartości jednorazowe ciągów identyfikujących pulę górniczą, o czym przekonamy się w następnych rozdziałach. Następne kilka cyfr szesnastkowych (03858402062) są używane do kodowania dodatkowego nonce (patrz "rozwiążanie extra nonce" na stronie 206) lub losową wartość, stosowaną w celu znalezienia odpowiedniego dowodu na rozwiązanie robocze.

Końcowa część danych coinbase (2f503253482f) jest zakodowana w ciągu znaków ASCII / P2SH /, co oznacza, że węzeł górniczy, który wydobywa ten blok obsługuje pay-to-script-hash (P2SH) poprawiając zdefiniowany w BIP0016. Wprowadzenie możliwości P2SH wymagało "głosowania" przez górników w celu zatwierdzenia BIP0016 lub BIP0017. Osoby popierające realizację BIP0016 miały zatrzymać / P2SH / w ich coinbase danych. Osoby popierające realizację BIP0017 z P2SH miały zawierać ciąg p2sh / CHV w ich coinbase danych. BIP0016 został wybrany jako zwycięzca i wielu górników kontynuowało włączanie ciągu / P2SH / w ich coinbase, by wskazać wsparcie dla tej cechy. Przykład 8-6 używa biblioteki libbitcoin wprowadzonej w "Klientach Alternatywnych, biblioteki i zestawy narzędzi" na stronie 56, aby wyodrębnić dane coinbase z bloku genezy, wyświetlając komunikat Satoshi. Należy zauważyć, że biblioteka libbitcoin zawiera statyczną kopię bloku genezy, więc na przykład kod można pobrać bezpośrednio z bloku genezy z biblioteki. Przykład 8-6. Wyodrębnianie danych coinbase z bloku genezy

```
/*
Display the genesis block message by Satoshi.
*/
#include <iostream>
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Create genesis block.
    bc::block_type block = bc::genesis_block();
    // Genesis block contains a single coinbase transaction.
    assert(block.transactions.size() == 1);
    // Get first transaction in block (coinbase).
    const bc::transaction_type& coinbase_tx = block.transactions[0];
    // Coinbase tx has a single input.
    assert(coinbase_tx.inputs.size() == 1);
    const bc::transaction_input_type& coinbase_input = coinbase_tx.inputs[0];
    // Convert the input script to its raw format.
    const bc::data_chunk& raw_message = save_script(coinbase_input.script);
    // Convert this to an std::string.
    std::string message;
    message.resize(raw_message.size());
    std::copy(raw_message.begin(), raw_message.end(), message.begin());
    // Display the genesis block message.
    std::cout << message << std::endl;
    return 0;
}
```

Kompilujemy kod z kompilatora GNU C ++ i uruchamiamy wynikowy plik wykonywalny, jak pokazano w przykładzie 8-7. Przykład 8-7. Kompilacja i uruchomienie przykładowego kodu Satoshi-words

```
$ # Compile the code
$ g++ -o satoshi-words satoshi-words.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Run the executable
$ ./satoshi-words
^D<>^A^DEThe Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

Konstruowanie nagłówka bloku

Aby skonstruować nagłówek bloku, węzeł wydobycia musi być wypełniony w sześciu polach jak wymieniono w tabeli 8-3.

Tabela 8-3. Struktura nagłówka bloku

Rozmiar	Obszar	Opis
4 bajty	wersja	Numer wersji oprogramowania do śledzenia / uaktualnienia protokołu
32 bajty	Hash poprzedniego bloku	Odniesienie do hasha poprzedniego (macierzystego) bloku w łańcuchu
32 bajty	Korzeń Merkle	Hash korzenia drzewa Merkle transakcji tego bloku
4 bajty	Datownik	Przybliżony czas utworzenia tego bloku (sekundy od epoki Uniksa)
4 bajty	Trudność docelowa	Algorytm docelowy proof-of-work utrudnieniem dla tego bloku
4 bajty	Chwilowy / Nouce	Licznik używany dla algorytmu proof-of-work

W czasie, w którym blok 277,316 był wydobywany numer wersji opisujący struktury bloku jest w wersji 2, który jest zakodowany w formacie little-endian w 4 bajty jako 0x02000000. Następnie węzeł wydobycia musi dodać "hash poprzedniego bloku." To jest skrót nagłówka bloku 277,315 poprzedni blok otrzymanego od sieci, którego węzeł Jinga zaakceptował i wybrał jako rodzica kandydata bloku 277,316. Hashem nagłówka bloku dla bloku 277,315 jest: 00000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569

Kolejnym krokiem jest podsumowanie wszystkich transakcji z drzewa Merkle, aby dodać korzeń Merkle do nagłówka bloku. Transakcja generacji znajduje się na liście jako pierwsza transakcji w bloku. Następnie 418 transakcji zostaje dodane po nim, dając sumę 419 transakcji w bloku. Jak widzieliśmy w "Drzewie Merkle" na stronie 164, musi być parzysta liczba "liści" węzłów w drzewie, więc ostatnia transakcja została powielona, tworząc 420 węzłów, z których każdy zawiera hash jednej transakcji. Skróty transakcyjne są następnie łączone w pary, tworzone na każdym poziomie drzewa, aż wszystkie transakcje są zestawione w jeden węzeł na "korzeniu" drzewa. Korzeń drzewa merkle podsumowuje wszystkie transakcje w pojedynczą o wartości 32 bajtów, które możesz zobaczyć jako wymienione jako "korzeń Merkle" w przykładzie 8-3 i tu:

c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e

Węzeł wydobywczy będzie następnie dodawał 4-bajtowy czasownik, zakodowany jako czasowy stempel "Epoki Unix ", który opiera się o liczbę sekund, które upłyнуły od północy dnia 1 stycznia 1970 roku, strefy UTC / GMT. Czas 1388185914 jest zgodny dla piątku, 27 grudnia 2013, 23:11:54 UTC / GMT. Węzeł następnie wypełnia cel trudności, który określa wymagane dla proof-of-work przeszkody, aby uczynić ten blok ważnym. Trudność jest zapamiętywana w pamięci jako metryka "bitów trudności", która jest kodowania wykładnikiem mantysy. Kodowanie ma wykładnik 1-bajtowy, a następujący przez 3-bajtowe mantisy (współczynnik). W blokach 277,316, na przykład, wartość bitów trudności to: 0x1903a30c. Pierwsza część 0x19 jest wykładnikiem szesnastkowym, a następna część, 0x03a30c jest współczynnikiem.

Koncepcja celu trudności jest opisana w "Cel trudności i ponownego kierowania" na stronie 195 i "trudność bitowa" oraz „trudność reprezentacji” na stronie 194. Ostatnie pole to nonce, który jest inicjowany do zera. Z wszystkimi innymi wypełnionymi polami, nagłówek bloku jest zakończony i proces wydobycia może się rozpocząć. Celem jest, aby znaleźć wartość dla nonce, którego rezultat hasha nagłówka bloku, jest niższy niż cel trudności. Węzeł wydobycia będzie trzeba przetestować bilionem lub trylionem wartości nonce przed tym jak zostanie stwierdzone, że spełnia ten wymóg.

Wydobycie bloku

Teraz, gdy blok kandydatujący został skonstruowany przez węzeł Jinga, jest to czas dla górnictwa sprzętu platformy, na to, by "przekopać" blok, aby znaleźć rozwiązańcie algorytmu proof-of-work, który sprawia, że blok ważny. W książce badaliśmy kryptograficzną funkcję kodującą, która jest stosowana w różnych aspektach systemu bitcoin. Funkcja skrótu SHA256 jest funkcja stosowaną w procesie wydobywczym Bitcoin. W najprostszym ujęciu, wydobycie to proces wielokrotnego mieszania nagłówków bloku, zmieniając jeden parametr, aż uzyskany hash odpowiadać będzie konkretному celowi. Wyniku funkcji hash nie można z góry określić, ani nie może być utworzony wzorzec, który będzie produkować określzoną wartość hash. Ta cecha funkcji hash oznacza, że jedynym sposobem, aby stworzyć efekt mieszania pasujący konkretному celowi jest próbowanie w kółko, przypadkowo modyfikując wejście, aż przez przypadek pojawi się pożądanym efektem mieszkania-hasha.

Algorytm Proof-of-work

Algorytm mieszania (hash) bierze wejście danych o arbitralnej długości i tworzy wynik o stałej długości deterministycznej-cyfrowy odcisk palca wejścia. Dla każdego konkretnego wejścia, uzyskany hash będzie zawsze taki sam i może być łatwo obliczany i weryfikowany przez kogokolwiek, kto wykonuje taki sam hash algorytm (algorytm mieszający). Kluczową cechą algorytmu hash kryptograficznego jest to, że jest to praktycznie niemożliwe, aby znaleźć dwa różne wejścia, które produkują te same odciski papilarnie. W konsekwencji, to jest praktycznie niemożliwe, aby wybrać wejście w taki sposób, aby uzyskać żądany odcisk palca, inny niż próbując losowych danych wejściowych. Z SHA256 blok wyjściowy jest zawsze długości 256 bitów, niezależnie od wielkości wsadu. W przykładzie 8-8, będziemy używać interpretera Pythona do obliczenia skrótu SHA256 z frazą "Jestem Satoshi Nakamoto".

Przykład 8-8. Przykład SHA256

```
Python 2.7.1
>>> import hashlib
>>> print hashlib.sha256("I am Satoshi Nakamoto").hexdigest()
5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e
```

Przykład 8-8 pokazuje wynik obliczania hasha "Jestem Satoshi Nakamoto": 5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e. Ten 256-bitowy numer jest hashem lub streszczeniem tego wyrażenia i zależy od każdej części tego zdania. Dodanie jednej litery, znak interpunkcyjnego lub dowolnego innego znaku będzie produkować inny hash. Teraz, jeśli zmienimy frazę, należy się spodziewać zobaczenia zupełnie innych hashów. Spróbujmy tego dodając liczbę do końca naszej frazy, za pomocą prostych skryptów Pythona w przykładzie 8-9.

Example 8-9. SHA256 A script for generating many hashes by iterating on a nonce

```
# example of iterating a nonce in a hashing algorithm's input

import hashlib

text = "I am Satoshi Nakamoto"

# iterate nonce from 0 to 19
for nonce in xrange(20):

    # add the nonce to the end of the text
    input = text + str(nonce)

    # calculate the SHA-256 hash of the input (text+nonce)
    hash = hashlib.sha256(input).hexdigest()

    # show the input and hash result
    print input, '=>', hash
```

Działanie tego spowoduje mieszanie kilku zdań wykonanych inaczej, dodając kilka numerów na końcu tekstu. Przez zwiększenie liczby, możemy uzyskać różne hashe (mieszania), jak pokazano w [Przykładzie 8-10](#).

```
$ python hash_example.py

I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583fffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbab...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

Przykład 8-10. Blok wyjściowy SHA256 skryptu do generowania wielu skrótów przez iterację jednorazową

Każda formuła daje zupełnie inny efekty mieszania. Wydają się to całkowicie losowe, ale można odtworzyć dokładne wyniki w tym przykładzie na dowolnym komputerze z Pythonem i zobaczyć dokładnie taki sam skrótu. Liczba jako zmienna w takim przypadku jest nazywana nonce.Nonce ,która służy do zmiany sygnału wyjściowego, jest funkcją kryptograficzną, w tym przypadku zmienia SHA256 odcisku palca wyrażenia. Aby dokonać wyzwania na obecnym algorytmie, ustawmy dowolny cel: znalezienie frazy, która wytwarza szesnastkową wartość mieszania, która zaczyna się od zera. Na szczęście, to nie jest trudne! Przykład 8-10 pokazuje, że wyrażenie "Jestem Satoshi Nakamoto13" produkuje hash 0ebc56d59a34f5082aaef3d66b37a661696c2b618e62432727216ba9531041a5, który pasuje do naszych kryteriów. Musielibyśmy wykonać 13 prób, aby go znaleźć. W kategoriach prawdopodobieństwa, jeśli blok wyjściowy z funkcji hash jest równomiernie spodziewalibyśmy się znaleźć wynik z 0 jako szesnastkowym prefiksem każdego 16 hashu (jeden z 16 cyfr szesnastkowych 0 do

Aby podać prostą analogię, wyobraźmy sobie grę, w której gracze rzucają kostką parę razy, starając się wyrzucić poniżej określony cel. W pierwszej rundzie, celem jest 12. Dopóki nie rzucasz podwójnego sześć wygrywasz. W następnej rundzie celem jest 11. Gracz musi wyrzucić 10 lub mniej, aby wygrać co także jest łatwym zadaniem. Powiedzmy kilka rund później cel jest do 5 oczek. Teraz, ponad połowa kości rzuca więcej niż 5, a zatem trudniej jest wygrać. To trwa wykładowczo- im więcej rzutów kostką tym mniejsze szans na osiągnięcie małego celu. W końcu, gdy celem jest 2 (minimalny możliwy) tylko jeden rzut z każdego 36, lub 2% z nich spowoduje wygrywający wynik. W przykładzie 8-10, zwycięski "nonce" to 13, a wynik ten może być potwierdzony przez każdego niezależnie. Każdy może dodać liczbę 13 jako przyrostek do frazy "Jestem Satoshi Nakamoto" i obliczyć hash, sprawdzając, czy jest ona mniejsza niż cel. Pomyślny wynik jest również proof of work, ponieważ dowodzi, że wykonaliśmy pracę, by znaleźć tego nonca. Choć zajmuje tylko jedno obliczenie skrótu w celu sprawdzenia, zajęło nam 13 obliczeń hashy, aby znaleźć nonce, który pracował. Gdybyśmy mieli niższy cel (większe trudności) znalezienie odpowiedniego nonca zajęłoby nam dużo więcej obliczeń hashy dla zweryfikowania kogokolwiek. Ponadto, znając cel, każdy może ocenić poziom trudności za pomocą statystyk i wiedzieć, jak wiele pracy było potrzebne do znalezienia takiego nonca. Bitcoin proof-of-work jest bardzo podobny do tego wyzwania przedstawionego w przykładzie 8-10. Górnik buduje blok-kandydat wypełnionego transakcjami. Następnie górnik oblicza skrót nagłówku tego bloku i widzi, jeżeli jest on mniejszy niż aktualny cel. Jeżeli hash nie jest niższy niż cel, górnik zmodyfikuje nonce (zwykle tylko powodując podwyżkę przez jeden) i próbuje ponownie. W obecnej trudnej sytuacji w sieci bitcoin górnicy muszą próbować kwadrylion razy zanim znajda nonce, które skutkuje wystarczająco niskim nagłówkiem bloku hash. Bardzo uproszczony algorytm proof-of-work jest realizowany w Pythonie w przykładzie 8-11. Przykład 8-11. Uproszczone wdrożenie proof-of-work

```
#!/usr/bin/env python
# example of proof-of-work algorithm

import hashlib
import time

max_nonce = 2 ** 32 # 4 billion

def proof_of_work(header, difficulty_bits):
    # calculate the difficulty target
    target = 2 ** (256-difficulty_bits)

    for nonce in xrange(max_nonce):
        hash_result = hashlib.sha256(str(header)+str(nonce)).hexdigest()

        # check if this is a valid result, below the target
        if hash_result[:difficulty_bits] == '0'*difficulty_bits:
            print "Found hash %s starting with %s" % (hash_result, hash_result[:difficulty_bits])
            return hash_result
```

```

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

```

Uruchomiąc ten kod, można ustawić żądaną trudność (w bitach, ile z wiodących bitów musi być równe zero) i zobaczyć, jak długo trzeba czekać, aby Twój komputer znalazł rozwiązanie. W przykładzie 8-12, można zobaczyć, jak to działa na przeciętnym laptopie.

Przykład 8-12. Uruchomienie przykładu proof-of-work dla różnych trudności

```

[...]
Difficulty: 8 (3 bits)
Starting search...
Success with nonce 9
Hash is 1c1c105e65b47142f028a8f93ddf3dabb9268491bc64474738133ce5256cb3c1
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 029ae6e5004302a120630adcbb888452346ab1cf0b94c5189ba8bac1d47e7903
Elapsed Time: 0.0006 seconds
Hashing Power: 58164 hashes per second
[...]
Difficulty: 4194384 (22 bits)
Starting search...
Success with nonce 1759164
Hash is 0000008bb0f0e731f0496b8e530da984e85fb3cd2bd81882fe8ba3610b6cef3
Elapsed Time: 13.3281 seconds
Hashing Power: 132068 hashes per second
Difficulty: 8388608 (23 bits)
Starting search...
Success with nonce 14224729
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
[...]
Hash is 0000001f0ea21e676b6dde5ad429b9d131a9f2b000002ab2f169cba22b1e21a
Elapsed Time: 665.0949 seconds
Hashing Power: 127141 hashes per second
Difficulty: 67108864 (26 bits)
Starting search...
Success with nonce 84561291

```

Jak widać, zwiększenie trudności

o 1 bit powoduje gwałtowny wzrost czasu potrzebnego na znalezienie rozwiązania. Jeśli myślisz o całej 256-bitowej przestrzeni liczb, za każdym razem kiedy ograniczasz jeszcze jeden kawałek do zera, należy zmniejszyć przestrzeń poszukiwań o połowę. W przykładzie 8-12, potrzeba 84 mln prób mieszania, by znaleźć wartości jednorazowe, który produkuje hash z 26 wiodących bitów jako zero. Nawet przy prędkości większej niż 120.000 mieszań na sekundę, to nadal wymaga 10 minut na laptopie klientów, aby znaleźć to rozwiązanie. W chwili pisania tego tekstu, sieć stara się znaleźć nagłówek bloku, którego skrót jest mniejszy niż 0000000000000004c296e6376db3a241271f43fd3f5de7ba18986e517a243baa7. Jak widać, istnieje wiele zer na początku tego hasha, co oznacza, że dopuszczalny zakres mieszań jest znacznie mniejszy, a więc jest to trudniejsze, by dotrzeć do poprawnego indeksu. To zajmie średnio ponad 150 kwadrylionów obliczeń hash na sekundę, aby odkryć następny blok. To wydaje się niemożliwe, ale na szczęście sieć przynosi 100 petahashes na sekundę (PH / sek) o mocy obliczeniowej do zniesienia, który będzie w stanie znaleźć blok w ciągu średnio 10 minut.

Trudność reprezentacji

W przykładzie 8-3 widzieliśmy, że blok zawiera cel trudności w notacji zwanej "bity trudności" lub po prostu "bity", który w bloku 277,316 ma wartość 0x1903a30c. Ten zapis wyraża cel trudności jako format Współczynnik / wykładnik z pierwszych dwóch cyfr szesnastkowych dla wykładnika i najbliższych sześciu

cyfr hex jako współczynnika. W tym bloku, zatem wykładnikiem jest 0x19, a współczynnik wynosi 0x03a30c. Wzorem do obliczenia trudności celu od tej reprezentacji jest:

```
target = coefficient * 2^(8 * (exponent - 3))
```

przy użyciu tego wzoru przy wartości trudności bitu 0x1903a30c, otrzymujemy:

target = 0x03a30c * 2^(0x08 * (0x19 - 0x03)) \Rightarrow target = 0x03a30c * 2^(0x08 * 0x16) \Rightarrow target = 0x03a30c * 2^0xB0 \wedge

który w systemie dziesiętnym wynosi=>

target = 238,348 * 2^176^ => target =
22,829,202,948,393,929,850,749,706,076,701,368,331,072,452,018,388,575,715,328

przełączając na tryb szesnastkowy:

Oznacza to, że w prawidłowym bloku dla wysokości 277,316 jest jeden, który ma hash nagłówka bloku, który jest mniejszy niż wartość docelowa. W binarnym liczba ta miałyby więcej niż pierwszych 60 bitów ustawionych na zero. Przy tym poziomie trudności, pojedynczy górnik przetwarza 1 trylion hashów na sekundę (1 tera-hash przez sekundę lub jeden TH / s) znajdzie tylko rozwiązanie raz na 8,496 bloków lub średnio raz na 59 dni.

Trudność docelowa i przekierowywanie

Jak widzieliśmy, cel determinuje poziom trudności, a tym samym wpływa na to jak długo trwa znalezienie rozwiązania algorytmu proof-of-work. Prowadzi to do oczywistych pytań: Dlaczego jest trudność z regulacją, kto dastosowuje i w jaki sposób? Bloki Bitcoin są generowane średnio co 10 minut. To jest jak bicie serca dla Bitcoin i wzmacnia częstotliwość emisji waluty i szybkość transakcji osiedlenia. To musi być stałą, nie tylko w krótkim okresie, ale w ciągu wielu dziesięcioleci. Przez cały ten czas, oczekuje się, że moc komputera będzie nadal rosła w szybkim tempie. Ponadto, liczba uczestników w górnictwie i komputerów używanych również stale się zmienia. Aby zachować czas generacji na 10 minutach, trudności w górnictwie muszą zostać dostosowane w celu uwzględnienia tych zmian. W rzeczywistości, trudność jest dynamicznym parametrem, który będzie okresowo dostosowywany, gdy spotka cel - blok.

W prostych słowach, trudność docelowa jest ustawiona tak, by jakiekolwiek zasilanie wydobycia spowodowało 10-minutowym odstępem. Jak więc jest takie dostosowanie wykonane w całkowicie zdecentralizowanej sieci? Trudnościowe przekierowanie odbywa się automatycznie, niezależnie nakażdym pełnym węźle. Co 2,016 bloki, wszystkie węzły ponownie kierują trudność proof-of-work. Równanie ponownego kierowania środkami trudności czasu zabraloby czas na znalezienie ostatnich 2,016 bloków i porównałoby to z przewidywanym czasem 20,160 minut (dwa tygodnie oparte na żądanym czasie 10-minutowego bloku). Stosunek pomiędzy rzeczywistym przedziałem czasu i żądanym przedziałem czasu jest obliczana i następnie odpowiednia korekta (w górę lub w dół) jest wykonana biorąc pod uwagę trudności. W prostych słowach: Jeśli sieć znajduje klocki szybciej niż co 10 minut, ze trudność wzrasta. Jeśli odkrycie bloku jest wolniejszy niż oczekiwano, trudności maleją. Równanie można podsumować następująco:

The equation can be summarized as:

```
New Difficulty = Old Difficulty * (Actual Time of Last 2016 Blocks / 20160 minutes)
```

Example 8-13 shows the code used in the Bitcoin Core client.

Example 8-13. Retargeting the proof-of-work difficulty—GetNextWorkRequired() in pow.cpp, line 43

```
// Go back by what we want to be 14 days worth of blocks
const CBlockIndex* pindexFirst = pindexLast;
for (int i = 0; pindexFirst && i < Params().Interval()-1; i++)
    pindexFirst = pindexFirst->pprev;
assert(pindexFirst);
```

Przykład 8-13 pokazuje kod użyty przez klienta Bitcoin Core

```
// Limit adjustment step
int64_t nActualTimespan = pindexLast->GetBlockTime() - pindexFirst->GetBlockTime();
LogPrintf(" nActualTimespan = %d before bounds\n", nActualTimespan);
if (nActualTimespan < Parans().TargetTimespan()/4)
    nActualTimespan = Parans().TargetTimespan()/4;
if (nActualTimespan > Parans().TargetTimespan()*4)
    nActualTimespan = Parans().TargetTimespan()*4;

// Retarget
uint256 bnNew;
uint256 bnOld;
bnNew.SetCompact(pindexLast->nBits);
bnOld = bnNew;
bnNew *= nActualTimespan;
bnNew /= Parans().TargetTimespan();

if (bnNew > Parans().ProofOfWorkLimit())
    bnNew = Parans().ProofOfWorkLimit();
```

Parametry interwałów (2 016 bloków) oraz Docelowy TargetTimespan (dwa tygodnie wyrażone jako 1 209 600 sekund) są zdefiniowane w *chainparams.cpp*.

Aby uniknąć ekstremalnie dużej niestabilności trudności korekta zmieniająca cel musi być mniejsza niż cztery (4) dla danego cyklu. Jeżeli wymagana korekta trudności ma czynnik większy od czterech, zostanie ona skorygowana o czynnik maksymalny, którego wartość nie może być przekroczena. Kolejne korekty będą zrealizowane w kolejnych okresach ponieważ brak równowagi będzie utrzymywać się przez kolejnych 2 016 bloków. Dlatego duże rozbieżności pomiędzy mocą funkcji hashującej a trudnością mogą utrzymywać się przez kilkanaście cykłów bloków wielkości 2 016, aby się zrównoważyć.



Trudność związana ze znalezieniem bloku bitcoinów to ok. 10 minut przetwarzania w przypadku całej sieci w oparciu o czas wymagany do znalezienia poprzednich 2 016 bloków, korygowany co kolejne 2 016 bloków.

Należy podkreślić, że trudność docelowa jest niezależna od liczby transakcji lub ich wartości. To oznacza, że wartość mocy hashującej, a tym samym elektryczności zużytej w związku z zabezpieczeniem bitcoinów jest całkowicie niezależna od liczby transakcji.

System bitcoin jest całkowicie skalowalny, może być szerzej stosowany, a jednocześnie pozostawać zachowywać zabezpieczenia bez jakiegokolwiek zwiększenia mocy hashującej w porównaniu z obecnym poziomem. Wzrost mocy hashującej reprezentuje siły rynkowe w związku z konkurencją pomiędzy kilkoma górnikami zmagającymi ze sobą na rynku, którzy chcą otrzymać tę samą nagrodę. Tak długo jako górnicy sprawują kontrolę nad wystarczającą ilością mocy funkcji hashującej i działają w oparciu o uczciwe zasady w wyścigu po nagrodę, wystarczy zapobiegać jedynie atakom "przejęcia", aby zabezpieczyć system bitcoin.

Trudność docelowa jest ściśle powiązana z kosztami elektryczności oraz kursem wymiany bitcoinów w stosunku do waluty, w której opłacane są rachunki za elektryczność. Wysokowydajne systemy górnicze są niemal tak wydajne jak to tylko możliwe mając na względzie obecnie dostępne generacji układów scalonych (silicon fabrication), zamieniające elektryczność na obliczenia funkcji hashujących z możliwie największą prędkością. Zasadniczy wpływ na rynek wydobycia ma cena jednej kilowatogodziny w systemie bitcoin, ponieważ to ona określa rentowność wydobycia i tym samym stanowi zachętą do uczestnictwa lub opuszczenia tego rynku.

Skuteczne wydobycie bloku

Jak mieliśmy okazję zaobserwować wcześniej, węzeł Jinga skonstruował blok kandydata i przygotował go do wydobycia. Jing posiada kilka urządzeń wydobywczych wyposażonych w układy scalone specyficzne dla aplikacji, w których setki układów wykorzystują algorytm SHA256 równolegle z niewiarygodną prędkością. Te wyspecjalizowane urządzenia są podłączone do jego węzła miningowego (wydobywczego) za pośrednictwem portu USB. Następnie węzeł wydobywczy działający na komputerze Jinga przesyła nagłówek bloku do oprogramowania wydobywczego, które uruchamia testy trylionów wartości jednorazowych na sekundę.

Niemal jedenaście minut po uruchomieniu wydobycia bloku 277 316, jedno z urządzeń wydobywczych znajduje rozwiązanie i przesyła je do węzła wydobywczego. Po wprowadzeniu do bloku nagłówka, wartość jednorazowa 4 215 469 401 generują wartość hash bloku wynoszącą:

0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569

Która jest niższa niż wartość docelowa:

00000000000000003A30C000000000000000000

Wezeł wydobywczy Jing przesyła ten blok dalej do wszystkich swoich

W kolejnej części zajmiemy się procesami wykorzystywanymi przez każdy z węzłów w celu zweryfikowania bloku oraz dokonania wyboru najdłuższego łańcucha tworzącego consensus stanowiący element składowy zdecentralizowanego bloku łańcucha.

Weryfikacja (walidacja) nowego bloku

Trzeci krok w mechanizmie consensusu bitcoin to niezależna weryfikacja każdego nowego bloku przez poszczególne węzły sieci. W miarę przechodzenia nowego bloku przez sieć, każdy węzeł wykonuje serie testów w celu zwalidowania go przed rozesyaniem do peerów. Proces ten zapewnia rozprzestrzenianie się jedynie zweryfikowanych bloków w sieci. Niezależny proces weryfikacji zapewnia także górnikom, którzy działają w uczciwy sposób włączenie ich bloków do łańcucha i tym samym zdobycie nagrody. Bloki górników nie działających na zasadach fair play są odrzucane i tracą nie tylko nagrodę, ale także marnują pracę, która wykonali w celu znalezienia rozwiązania *proof-of-work*, tym samym ponosząc koszty elektryczności bez żadnego wynagrodzenia.

Kiedy węzeł otrzymuje nowy blok, będzie on zweryfikowany względem długiej listy kryteriów, które muszą być spełnione; w przeciwnym razie, blok zostaje odrzucony. Kryteria te mogą być zaobserwowane w kliencie Bitcoin Core w funkcjach CheckBlock oraz CheckBlockHeader i będą składać się z:

- Składniowo ważną strukturę danych bloku
- Funkcję hashującą nagłówka bloku, która wynosi mniej niż trudność docelowa (egzekwuje *proof of work*)
- Znacznik czasu bloku, który ma wartość niższą niż dwie godziny w przyszłości (zostawiając margines na błędy)
- Wielkość bloku mieszcząca się w akceptowalnych granicach
- Pierwszą transakcję (i tylko pierwszą), która jest transakcją generowania monet
- Wszystkie transakcje w bloku są ważne w oparciu o listę transakcyjną omówioną w części „**Niezależna weryfikacja transakcji**”

Niezależna walidacja każdego nowego bloku przez wszystkie węzły w sieci gwarantuje brak możliwości dla górników do popełniania oszustwa. W poprzednich częściach widzieliśmy w jaki sposób górnicy zapisują transakcję, które przyznają im nowe bitcoiny utworzone w ramach bloku oraz pobierają opłaty transakcyjną. Dlaczego górnicy sami nie przygotują transakcji na tysiąc bitcoinów zamiast poprawnej nagrody? Ponieważ każdy węzeł weryfikuje bloki w oparciu o te same reguły. Nieważna transakcja w coinbase może unieważnić cały blok, spowodować jego odrzucenie i tym samym transakcja może nigdy nie zostać zarejestrowana w księdze. Górnicy muszą zbudować doskonały blok w oparciu o reguły, które są stosowane przez wszystkie węzły i wydobywać je za pomocą odpowiedniego rozwiązania dla *proof of work*. Aby tego dokonać, zużywają znaczną ilość elektryczności na cele wydobywcze i jeżeli będą oszukiwać, zużyty prąd oraz nakład pracy zostanie zmarnowany. Oto dlaczego niezależna weryfikacja stanowi kluczowy element zdecentralizowanego consensusu.

Gromadzenie oraz wybór bloków łańcucha

Ostatni krok w zdecentralizowanym mechanizmie consensus bitcoin to łączenie bloków w łańcuchy oraz wybór łańcuchów za pomocą *proof of work*. Po zweryfikowaniu przez węzeł nowego bloku, zostanie podjęta próba połączenia bloku do istniejącego łańcucha.

Węzły utrzymują trzy zestawy bloków: te, które są podłączane do głównych łańcuchów, te które tworzą rozgałęzienia od głównych bloków (łańcuchy wtórne) i w końcu te, które nie posiadają znanych rodziców w znany łańcuchu (sieroty). Nieważne bloki są odrzucane, gdy tylko jedno z kryteriów walidacji nie zostanie spełnione i tym samym nie są dołączane do żadnego łańcucha.

„Główny łańcuch” to zawsze ten blok łańcuchów, który posiada najwyższą trudność kumulacyjną z nim skojarzoną. W większości sytuacji, jest to także łańcuch z największą ilością bloków, chyba że istnieją dwa równej długości łańcuchy i jeden posiada więcej rozwiązań *proof of work*. łańcuch główny będzie także mieć więcej gałęzi z blokami, które są “rodzeństwem” dla bloków w łańcuchu głównym. Bloki te są ważne, ale nie stanowią części łańcucha głównego. Są one przechowywane i dostępne w przyszłości w przypadku, gdy jeden z nich zostanie rozszerzony i przekroczy stopień trudności w łańcuchu głównym. W dalszej części („**Rozwidlenia łańcucha blokowego**”), zobaczymy w jaki sposób łańcuchy drugorzędne pojawiają się jako wynik niemal jednoczesnego wydobycia bloków na tej samej wysokości.

Kiedy nowy blok jest odbierany, węzeł podejmie próbę umieszczenia go w istniejącym łańcuchu blokowym. Węzeł zapozna się z „poprzednią wartością hash pola” bloku, który stanowi wartość referencyjną dla rodzica nowego bloku. Następnie węzeł podejmie próbę znalezienia rodzica w istniejącym łańcuchu blokowym. Przez większość czasów rodzic będzie stanowić wskazówkę pozwalającą określić łańcuch główny, co oznacza, że ten nowy blok rozszerzy łańcuch główny. Dla przykładu: nowy blok 277 316 ma odniesienie do funkcji hashującej swojego bloku macierzystego 277 315. Większość węzłów otrzymujących 277 316 będzie już mieć wcześniej zapisany blok 277 315 jako wskazówkę pozwalającą ustalić łańcuch główny i przyłączyć nowy blok rozszerzając łańcuch.

Niekiedy, jak się o tym przekonamy w części zatytułowanej „Rozwidlenia łańcuchów blokowych”, nowy blok rozbudowuje łańcuch, który nie jest łańcuchem głównym. W takim przypadku, węzeł przyłączy nowy blok do łańcucha wtórnego, który rozszerza, a następnie porówna trudność łańcucha wtórnego z łańcuchem głównym. Jeżeli łańcuch wtórny ma większą trudność skumulowaną niż łańcuch główny węzeł skupi się na łańcuchu wtórnym co oznacza, że dokona wyboru łańcucha wtórnego jako swojego nowego łańcucha głównego, przekształcając stały łańcuch główny na łańcuch wtórny. Jeżeli węzeł jest jednocześnie górnikiem, to skonstruuje on blok rozszerzający nowy dłuższy łańcuch.

Jeżeli zostanie pobrany nowy blok, a w istniejących łańcuchach nie zostanie znaleziony żaden element macierzysty, blok ten zostanie uznany za “sierotę”. Bloki - sieroty są zapisywane w puli bloków sierot, gdzie pozostają do chwili znalezienia (otrzymania) rodzica. Po otrzymaniu rodzica i połączeniu go z istniejącym łańcuchem, sierota może być wyciągnięta z puli sierot i połączona z rodzicem sprawiając, że stanie się częścią łańcucha. Bloki sierot zwykle pojawiają się, kiedy dwa wydobyte bloki w krótkim odstępie czasu są odebrane w odwrotnej kolejności (dziecko przed rodzicem).

Wybierając łańcuch o najwyższym poziomie trudności, wszystkie węzły w końcu osiągną ogólno-sieciowy consensus. Tymczasowe rozbieżności pomiędzy łańcuchami są ostatecznie rozwiązywane, w miarę jak dodawane są *proof of work* rozszerzając jeden z możliwych łańcuchów. Węzły wydobywcze “głosują” wykorzystując swoją moc wydobywania dokonując wyboru łańcucha, który ma być rozszerzony po wydobyciu kolejnego bloku. Po wydobyciu i rozszerzeniu łańcucha, nowy blok stanowi reprezentację ich głosu.

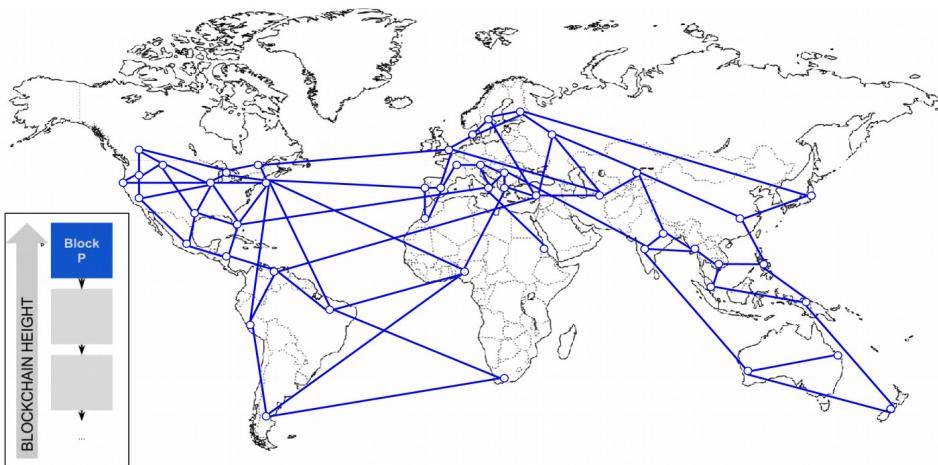
W kolejnym rozdziale przyjrzymy się w jaki sposób rozbieżności pomiędzy konkurencyjnymi łańcuchami (widelcami) są rozwiązywane w drodze niezależnego wyboru najdłuższego łańcucha trudności.

Rozwidlenia łańcuchów blokowych

Ponieważ łańcuch blokowy to zdecentralizowana struktura danych, różne kopie tej struktury są nie zawsze jednakowe. Bloki mogą pojawiać się różnych węzłach o różnym czasie sprawiając, że węzły będą mieć odmienne perspektywy na łańcuch blokowy. Aby rozwiązać to zadanie, każdy węzeł zawsze dokonuje wyboru oraz podejmuje próbę rozszerzenia łańcucha bloków, który reprezentuje najwięcej rozwiązań *proof of work*, znanych także pod nazwą najdłuższego łańcucha lub najdłuższego łańcucha o skumulowanej trudności. Podsumowując trudność zapisaną w każdym bloku łańcucha, węzeł może obliczyć łączną liczbę *proof of work*, które zostały wydane/wysłane w celu zbudowania tego łańcucha. Tak długo jak wszystkie węzły dokonują wyboru najdłuższego łańcucha o skumulowanej trudności, globalna sieć bitcoin w końcu zbiegnie się w jednym punkcie osiągając spójność. Widelce pojawiają się jako tymczasowe rozbieżności pomiędzy wersjami łańcucha blokowego, które są rozwiązywane poprzez wtórną zbieżność w miarę dodawania bloków do jednego z odgałęzień.

Na kolejnych kilku wykresach, śledzimy przebieg i postępy w rozprzestrzenianiu się zdarzenia rozgałęzienia po sieci. Wykres jest uproszczoną reprezentacją systemu bitcoin jako sieci globalnej. W rzeczywistości, topologia sieci bitcoin nie jest zorganizowana geograficznie. Zamiast tego, tworzy ona siatkę wzajemnie połączonych ze sobą węzłów, które mogą znajdować się dużym oddaleniu geograficznym od siebie. Reprezentacja topologii geograficznej to uproszczenie wykorzystywane w celu zilustrowania rozwidlenia. W rzeczywistej sieci bitcoin, “odległość” pomiędzy węzłami jest mierzona jako liczba “skoków” pomiędzy węzłami, które nie znajdują się w ich lokalizacji fizycznej. W celu zilustrowania problemu przyjmijmy, że różne bloki są przedstawione różnymi kolorami, rozszerzającymi się po sieci i zabarwiającymi połączenia przez które przechodzą.

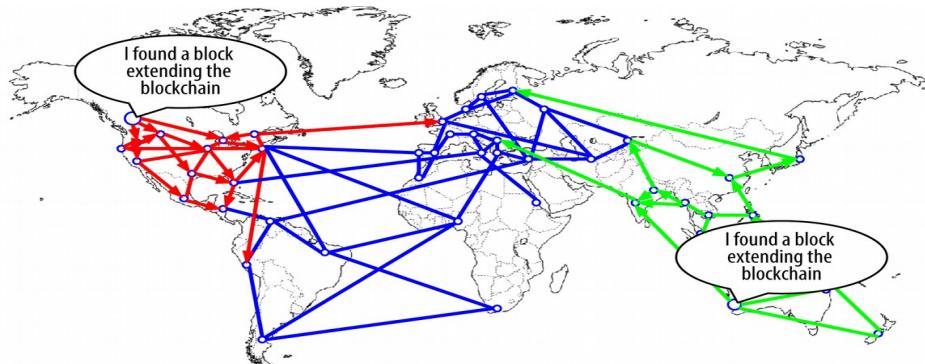
Na pierwszym wykresie (Rys. 8-2), sieć posiada ujednolicony łańcuch blokowy z niebieskim blokiem na końcu głównego łańcucha.



Rys. 8-2. Wizualizacja zdarzenia rozwidlenia łańcucha przed rozwidleniem

“Rozwidlenie” pojawia się zawsze, gdy mamy do czynienia z dwoma blokami kandydatów konkurującymi ze sobą o utworzenie najdłuższego łańcucha blokowego. Następuje to w normalnych warunkach zawsze, gdy dwóch górników rozwiązuje algorytm *proof of work* w krótkich odstępach czasu. Kiedy dwaj górnicy znajdują rozwiązanie dla swoich odpowiednich bloków kandydatów, natychmiast zaczynają wysyłać swój “zwycięski” blok do bezpośrednich sąsiadów, którzy zaczynają go powielać w sieci. Każdy węzeł otrzymujący ważny blok uwzględnia go w swoim łańcuchu blokowym, rozszerzając ten łańcuch o jeden blok. Jeżeli następnie węzeł ten zauważ kolejny blok kandydujący rozszerzający tego samego rodzica, przyłącza on drugiego kandydata do łańcucha wtórnego. W efekcie, niektóre węzły będą widzieć “jeden” blok kandydujący jako pierwszy, a pozostałe węzły będą widzieć drugi blok kandydata i pojawią się dwie konkurencyjne wersje łańcucha blokowego.

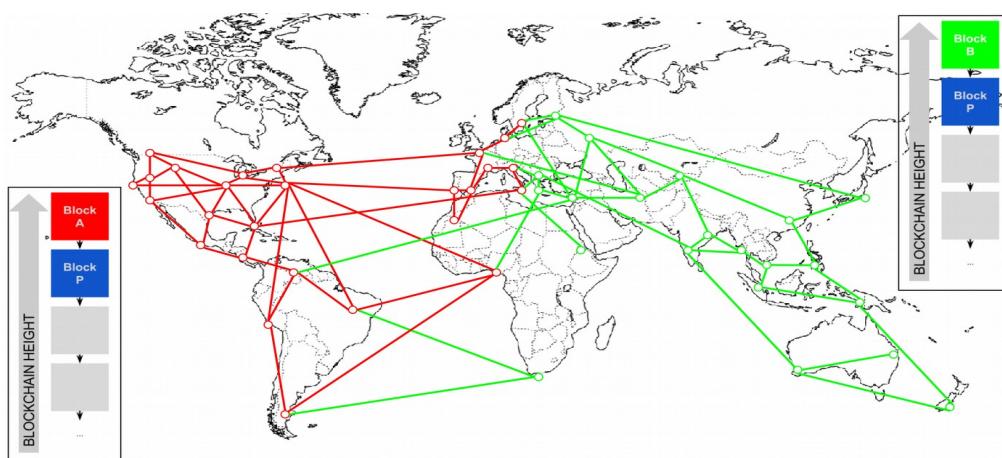
Na Rys. 8-3, widzimy dwóch górników, którzy wydobywają dwa różne bloki niemal jednocześnie. Obydwa bloki są dziećmi bloku błękitnego, którego celem jest rozszerzenie łańcucha poprzez dodanie elementu na szczyt błękitnego bloku. Aby pomóc w śledzeniu go, jeden jest przedstawiony, jako blok czerwony wywodzący się z Kanady, a drugi jest oznaczony na zielono jako blok wywodzący się z Australii.



Rys. 8-3. Wizualizacja zdarzenia rozwidlenia łańcucha: dwa bloki znalezione jednocześnie

Załóżmy na przykład, że górnik ten (w Kanadzie) znajduje rozwiązanie *proof-of-work* dla bloku „czerwonego”, który rozszerza łańcuch blokowy, dodając na szczycie bloku macierzystego „błękitny.” Niemal jednocześnie, górnik z Australii, który także rozszerzał blok „błękitny” znajduje rozwiązanie dla bloku „zielonego”- jego bloku kandydującego. Teraz mamy dwa potencjalne bloki – jeden nazywamy „czerwonym” (ten z Kanady), a drugi „zielonym” (ten z Australii). Obydwa bloki są ważne, zawierają ważne rozwiązanie dla *proof of work* oraz rozszerzają tego samego rodzica. Obydwa bloki najprawdopodobniej zawierają te same transakcje z kilkoma zapewne różnicami w postaci kolejności transakcji.

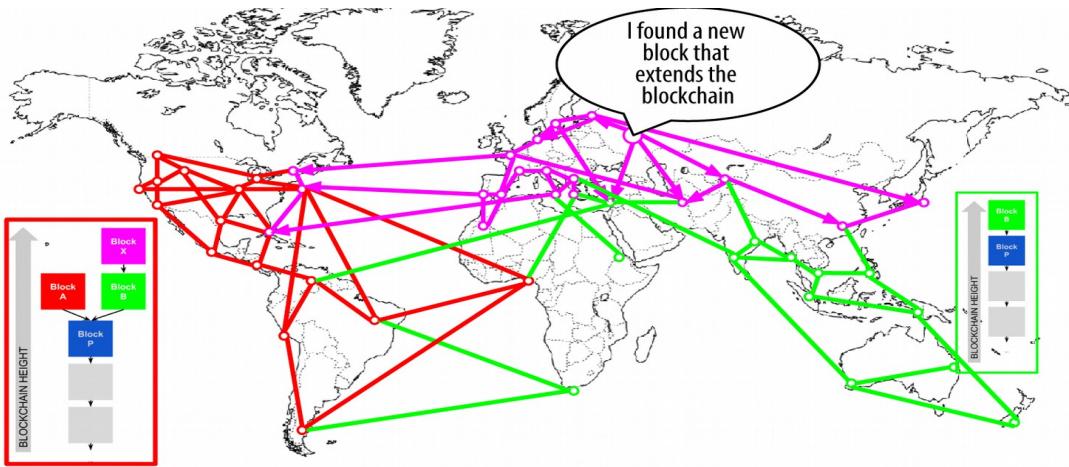
W miarę rozprzestrzeniania się bloków, niektóre węzły otrzymują blok „czerwony” jako pierwszy, inne „zielony”. Zgodnie z [Rys. 8-4](#), sieć dzieli się na dwie odmienne perspektywy łańcucha blokowego - jedna jest zakończona blokiem czerwonym, a druga zielonym.



Rys. 8-4. Wizualizacja zdarzenia rozwidlenia łańcucha: dwa bloki rozprzestrzeniają się dzieląc sieć

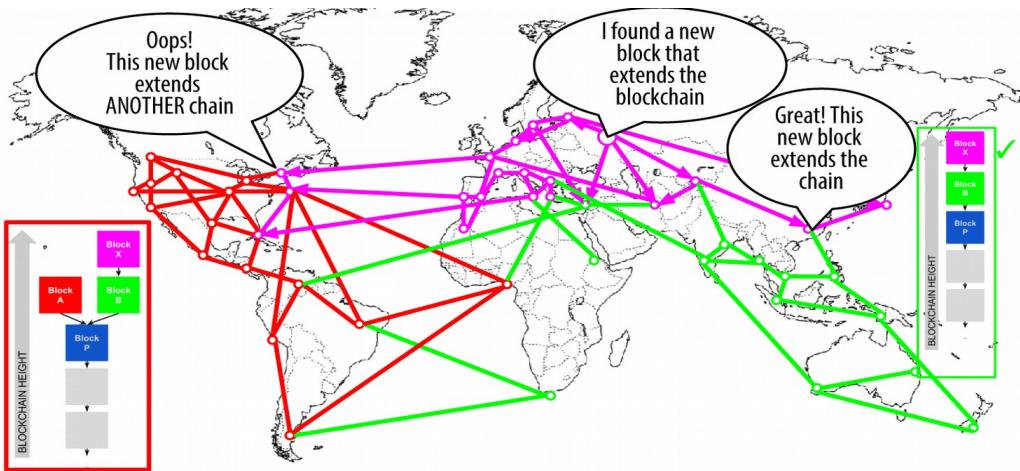
Od tej chwili węzły sieci znajdujące się najbliżej (topologicznie, niegeograficznie) węzła kanadyjskiego usłyszzą o bloku „czerwonym” najpierw i utworzą nowy łańcuch blokowy z największą skumulowaną trudnością, jako „czerwony” i jednocześnie ostatni blok w łańcuchu (np. błękitno-czerwony) ignorując „zielony” blok kandydujący, który przybywa nieco później. Tymczasem, węzły znajdujące się najbliżej węzła australijskiego uzają ten blok za zwycięzcę i rozszerzą łańcuch blokowy w kolorze „zielonym” jako ostatnim blokiem (np. błękitno-zielonym), ignorując „czerwony”, kiedy pojawi się kilka sekund później. Wszyscy górnicy, którzy ujrzaли „czerwony” jako pierwszy natychmiast zbudują bloki kandydujące zawierające odniesieni do „czerwonego” jako rodzica i podejmą próbę rozwiązania *proof of work* dla tych bloków. Górnicy, którzy zaakceptowali „zielony” zaczną budowanie „zielonego” na szczycie i rozszerzą ten łańcuch.

Rozwidlenia są niemal zawsze rozwiązywane w ramach jednego bloku. Jako część mocy hashującej sieci są one dedykowane do dobudowywania na szczycie „czerwonego” jako rodzic, podczas gdy inna część mocy hashującej koncentruje się na dobudowywaniu „zielonego.” Nawet, jeśli moc hashująca się niemal zawsze równo dzielona, istnieje prawdopodobieństwo, że jeden zespół górników znajdzie rozwiązanie i będzie rozsyłać zanim inni górnicy znajdują odmienne rozwiązania. Założymy, że przykładowo, górnicy dobudowujący na szczycie łańcucha „zielony” blok znajdują nowy blok „różowy”, który rozszerza łańcuch (np. błękitno zielono-różowy). Zaczynają natychmiast rozsyłać ten nowy blok dalej i cała sieć zaczyna go widzieć jako ważne rozwiązanie przedstawione na [Rys. 8-5](#).



Rys. 8-5. Wizualizacja zdarzenia rozwidlenia łańcucha: nowy blok rozszerza jedno rozwidlenie

Wszystkie węzły, które wybrały blok "zielony", jako zwycięzcę w poprzedniej rundzie będą po prostu rozszerzać łańcuch o jeden blok. Wszystkie węzły, które wybrały blok "czerwony" jako zwycięzcę, będą teraz widzieć dwa łańcuchy: błękitno-zielono-różowy oraz błękitno-czerwony. łańcuch błękitno-zielono-różowy jest obecnie dłuższy (wyższa trudność skumulowana) niż łańcuch błękitno-czerwony. W efekcie, węzły te ustawią łańcuch błękitno-zielono-różowy jako główny łańcuch i zamienią błękitno-czerwony na łańcuch wtórnego zgodnie z Rys. 8-6. Proces ten nazywamy rekonwergencją łańcucha, ponieważ węzły te są zmuszone do zmiany swojej perspektywy łańcucha i uwzględnienia w łańcuchu nowych dowodów potwierdzających poprawność dłuższego łańcucha. Wszyscy górnicy pracujący nad rozszerzeniem łańcucha błękitno-czerwonego teraz wstrzymują swoje prace ponieważ ich blok kandydujący jest „sierotą” a jego rodzić (czerwony) nie jest już w najdłuższym łańcuchu. Transakcja w "czerwonym" bloku jest ponownie kolejkowana w celu przetworzenia w kolejnym bloku, ponieważ blok ten nie stanowi już części łańcucha głównego. Cała sieć dokonuje rekonwergencji na pojedynczy łańcuch blokowy błękitno-zielono-różowy, z "różowym" jako ostatnim blokiem w łańcuchu. Wszyscy górnicy natychmiast zaczną pracę nad blokami kandydującymi, które określają "różowy" jako rodzica, aby rozszerzyć łańcuch błękitno-zielono-różowy.



Rys. 8-6. Wizualizacja zdarzenia rozwidlenia łańcucha: sieć rekonwerguje na nowym najdłuższym łańcuchu

Istnieje teoretyczna możliwość, aby rozwidlenie rozbiegało się w kierunku dwóch bloków, jeżeli są one znalezione niemal jednocześnie przez górników na przeciwnych "stronach" poprzedniego rozwidlenia. Nie mniej jednak szansa, aby to wystąpiło jest niewielka. Ponieważ jedno-blokowe rozwidlenie może pojawiać się co tydzień, dwublokowe rozwidlenie jest niezwykle rzadkie.

Interwał blokowy 10 minutowy ma stanowić kompromis pomiędzy czasem szybkiego potwierdzenia (rozliczenie transakcji) a prawdopodobieństwem wystąpienia rozwidlenia. Szybszy blok czasowy, może szybciej pokazać transakcje, ale powodować częstsze wystąpienie rozwidleń łańcuchów podczas, gdy dłuższy czas przyczyni się do zredukowania liczby rozwidleń, ale za to spowolni rozliczenia.

Mining i wyścig hashowania

Mining bitcoinów to niezwykle konkurencyjna dziedzina. Moc hashująca wzrasta wykładniczo co roku od momentu pojawienia się systemu. W niektórych latach wzrost odzwierciedla całkowitą zmianę technologii, jak miało to miejsce w roku 2010 i 2011, kiedy wielu górników przeszło z miningu CPU na mining GPU oraz mining typu *field programmable gate array* (FPGA). W 2013 r., wprowadzenie wydobycia typu ASIC doprowadziło do kolejnego skokowego postępu poprzez umieszczenie funkcji SHA256 bezpośrednio na układzie scalonym zaprojektowanym specjalnie dla celów wydobywczych. Pierwszy tego typu układ był w stanie dostarczyć większej siły wydobywczej niż cała sieć działająca w 2010 roku.

Poniżej przedstawiona jest lista łącznej mocy hashującej sieci bitcoin na przestrzeni pierwszych pięciu lat jej działania:

2009

0.5 MH/s–8 MH/s (16× wzrost)

2010

8 MH/s–116 GH/s (14 500× wzrost)

2011

16 GH/s–9 TH/s (562× wzrost)

2012

9 TH/s–23 TH/s (2.5× wzrost)

2013

23 TH/s–10 PH/s (450× wzrost)

2014

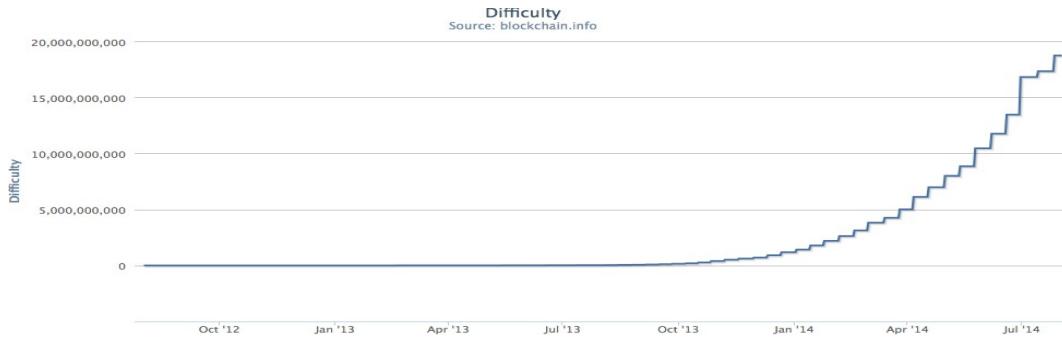
10 PH/s–150 PH/s w sierpniu (15× wzrost)

Na wykresie na Rys. 8-7 jest przedstawiony przyrost mocy hashującej sieci bitcoin w okresie ostatnich dwóch lat. Jak widać, konkurencja pomiędzy górnikami oraz wzrost sieci był wynikiem przyrostu wykładniczego mocy hashującej (liczba funkcji hashujących realizowanych w ciągu 1 sekundy w sieci).



Rys. 8-7. Całkowita moc hashująca wyrażona w gigahashach na sekundę w okresie ostatnich dwóch lat

Ponieważ wartość mocy hashującej stosowana w procesie mining bitcoinów zwiększyła się zasadniczo, wzrósł także stopień trudności z nim związany. Stopień trudności przedstawiony na Rys. 8-8 jest obliczony jako współczynnik trudności obecnej do wartości trudności minimalnej (trudność pierwszego bloku).



Rys. 8-8. Bitcoin's mining difficulty metric, over two years

Na przestrzeni ostatnich dwóch lat, mikroprocesory miningowe ASIC stały się gęstsze zbliżając się do najnowszych osiągnięć w tej dziedzinie osiągając rozmiar (rozdzielcość) rzędu 22 nanometrów (nm). Obecnie, producenci ASIC stawiają sobie jako cel przejęcie produkcji mikroprocesorów ogólnego przeznaczenia i projektowanie mikroprocesorów o rozmiarze 16nm, ponieważ rentowność wydobycia stanowi jeszcze większą siłę napędową tego sektora, niż systemy obliczeniowe ogólnego użytku. W dziedzinie wydobycia bitcoinów pozostało niewiele miejsca na ogromne, skokowe postępy, ponieważ sektor osiągnął próg, po przekroczeniu którego ma zastosowanie prawo Moore'a stanowiące, że gęstość obliczeniowa będzie dwukrotnie gęstnieć co ok. 18 miesięcy. Nie mniej jednak siła wydobywcy sieci w dalszym ciągu będzie postępować w tempie wykładniczym z uwagi na wyścieg ku coraz bardziej zagęszczonym mikroprocesorom, któremu towarzyszy równie intensywny wyścieg ku tworzeniu coraz bardziej skoncentrowanych centrów danych, których te nowe procesory znajdą swoje zastosowanie. Problem nie polega już na sposobie wydobycia bloków za pomocą pojedynczego mikroprocesora, ile takich mikroprocesorów można zmieścić w budynku zachowując jednocześnie możliwość rozproszenia ciepła i odpowiednią moc.

Rozwiązywanie kwestii dodatkowych wartości jednorazowych

Od roku 2012, sektor wydobycia bitcoinów ewoluje w stronę rozwiązywania fundamentalnego w charakterze ograniczenia struktury nagłówka bloku. W początkowej fazie rozwoju systemu bitcoin, górnik miał możliwość znalezienia bloku powtarzając wartości jednorazowe, aż do chwili znalezienia wynikowej wartości hash, poniżej wartości docelowej. W miarę wzrostu trudności, cykle wydobywcze górników obejmowały wszystkie 4 miliardy wartości bez znalezienia bloku. Problem ten został w prosty sposób rozwiązany poprzez aktualizację bloku znacznika czasu podającego upływający czas. Ponieważ znacznik czasowy stanowi część nagłówka, zmiana mogła umożliwić górnikom ponowne wykorzystanie kolejnych wartości jednorazowych i uzyskiwanie różnych wyników. Kiedy urządzenia wydobywcze przekroczyły 4 GH/s, podejście to okazało się niezwykle niedogodne, ponieważ wartości jednorazowe wyczerpują się w czasie krótszym, niż jedna sekunda. Z uwagi na to, że sprzęt wydobywczy ASIC zaczął doganiać, a potem przekroczyć prędkość hashowania TH/s, oprogramowanie wydobywcze zaczęło wymagać coraz więcej przestrzeni na wartości jednorazowe, aby umożliwić znalezienie ważnych bloków. Znacznik czasowy można nieco rozszerzyć (naciągnąć), ale wysunięcie go zbyt daleko w przeszłość mogłoby przyczynić się do unieważnienia bloku. Rozwiązywanie to miało wykorzystywać transakcję

coinbase jako źródło dodatkowych wartości jednorazowych. Ponieważ skrypty coinbase mogą przechowywać od 2 do 100 bajtów, górnicy zaczęli wykorzystywać tę przestrzeń, jako dodatkową przestrzeń dla wartości jednorazowych umożliwiając im zbadanie o wiele większej liczby wartości nagłówka bloku w celu znalezienia ważnych bloków. Transakcja coinbase stanowi część drzewka skrótów, co oznacza, że każda zmiana w skrypcie coinbase będzie powodować zmianę w korzeniu drzewka. Osiem bajtów dodatkowych wartości jednorazowych plus 4 bajty "standardowych" wartości jednorazowych umożliwiają górnikom zbadanie łącznej liczby 2^{96} (8 po których następuje 28 zer) możliwości na sekundę bez konieczności modyfikowania znacznika czasu. Jeżeli, kiedykolwiek w przeszłości górnicy uruchomią wszystkie te możliwości, będą mogli także zmodyfikować znacznik czasowy. Skrypt coinbase zawiera także więcej miejsca, które może być wykorzystane w przyszłości do rozszerzenia dodatkowej przestrzeni wartości jednorazowych.

Pula minigowa

W tym niezwykle konkurencyjnym środowisku, poszczególni górnicy pracujący w pojedynkę (nazywani także górnikami solo) nie mają żadnych szans. Prawdopodobieństwo, że znajdą blok, który pozwoli im pokryć koszty energii oraz sprzętu jest tak niskie, że graniczny hazard – podobnie jak gra w loterię. Nawet najszybszy konsumentki system wydobywczy nie może równać się z systemami komercyjnymi, które posiadają dziesiątki tysięcy tych mikroprocesorów w gigantycznych magazynach usytuowanych w pobliżu elektrowni wodnych. Górnicy obecnie współpracują ze sobą tworząc pulę górniczą gromadzącą w jednym miejscu moc hashującą oraz dzieląc się nagrodami z tysiącami innych uczestników. Uczestnictwo w puli gwarantuje górnikom mniejszy udział w nagrodzie, ale z reguły nagrody te otrzymują codziennie przez co niepewność jest zredukowana.

Przyjrzymy się szczególnej sytuacji. Założymy, że górnik nabył sprzęt wydobywczy o kombinowanej prędkości hashowania rzędu 6 000 gigahashy na sekundę (GH/s) lub 6 TH/s. W sierpniu 2014 r. sprzęt kosztował w przybliżeniu 10 000 dolarów. Urządzenia zużywają 3 kilowaty (kW) energii podczas pracy, 72 kWh dziennie; średni koszt zużytnej energii wynosi 7 lub 8 dolarów dziennie. Przy obecnym poziomie trudności w systemie bitcoin, zwycięzca będzie w stanie wydobyć działając w pojedynkę jeden blok na 155 dni lub co 5 miesięcy. Jeżeli górnik znajdzie blok w tym czasie, jego nagroda wyniesie 25 bitcoinów po 600 dolarów za bitcoina, co oznacza jednorazową wypłatę w wysokości 15 000 dolarów oraz pokrycie kosztów oprogramowania oraz energii zużytnej w tym czasie; zysk netto wyniesie zatem ok. 3 000. Szansa na znalezienie bloku w okresie 5 miesięcy zależy od szczęścia górnika. Może się zdarzyć, że znajdzie dwa bloki w ciągu pięciu miesięcy odnosząc spory zysk. Może się także okazać, że nie znajdzie żadnego bloku w okresie 10 ponosząc straty finansowe. Co więcej, trudność związana z algorytmem *proof-of-work* prawdopodobnie wzrośnie znacznie w tym okresie stosownie do bieżącej prędkości przyrostu mocy hashującej, co oznacza, że górnik ma najwyżej sześć miesięcy, aby wyjść na zero zanim jego sprzęt stanie się przestarzały i będzie wymagać wymiany na nowszy i mocniejszy. Jeżeli górnik ten jest członkiem puli górników zamiast czekać na jednorazowy przypływ dużej gotówki raz na pięć miesięcy w kwocie 15 000, przyłączając się do puli będzie w stanie zarobić ok. 500 do 750 dolarów tygodniowo. Regularne wypłaty pomogą mu pokryć koszty sprzętu oraz elektryczności poniesione w tym czasie bez żadnego większego ryzyka. Sprzęt stanie się przestarzały po upływie sześciu do dziewięciu miesięcy, a ryzyko dalej pozostaje na wysokim poziomie – nie mniej jednak dochód jest regularny i pewny przez cały ten okres.

Pule górników koordynują setki, a nawet tysiące górników za pośrednictwem wyspecjalizowanych protokołów poolminingowych. Poszczególni górnicy konfigurują swój sprzęt, aby połączyć się z serwerem puli, po utworzeniu konta w puli. Ich sprzęt wydobywczy jest podłączony do puli serwera jednocześnie wydobywając i synchronizując swoje wysiłki przy współpracy z innymi górnikami. W związku z tym, górnicy tworzący daną pulę dzielą się pracą w celu wydobycia bloku, a następnie nagrodą.

Successful blocks pay the reward to a pool bitcoin address, rather than individual miners. The pool server will periodically make payments to the miners' bitcoin addresses, once their share of the rewards has reached a certain threshold. Typically, the pool server charges a percentage fee of the rewards for providing the pool-mining service.

Górnicy uczestniczący w puli dzielą się pracą związaną z poszukiwaniem rozwiązania, dla bloku kandydującego zdobywając „udziały” za wkład w prace wydobywcze. Pula miningowa określa niższe cele trudności związane ze zdobyciem udziału – z reguły są one ponad 1 000 razy łatwiejsze niż trudność sieci bitcoin. Kiedy ktoś w puli skutecznie wydobędzie blok, nagroda jest przyznawana puli, a następnie dzielona pomiędzy górników proporcjonalnie do liczby ich udziału w ogólnym sukcesie.

Pule są otwarte dla każdego górnika, dużego i małego, profesjonalisty lub amatora. Pula w związku z tym będzie mieć uczestników posiadających małe urządzenia wydobywcze oraz innych dysponujących zaawansowanym sprzętem wydobywczym. Niektórzy będą wydobywać wykorzystując kilkadziesiąt kilowatów elektryczności, inni zaś będą prowadzić centra danych pochłaniające megawaty energii. W jaki sposób pula górników dokonuje pomiaru indywidualnego wkładu i sprawiedliwego podziału nagród bez ryzyka oszustwa? Odpowiedź tkwi w algorytmie *proof-of-work* sieci bitcoin, który mierzy wkład każdego z górników puli, stosując niższy poziom trudności w taki sposób, że każdy, nawet najdrobniejszy górnik ma swój udział na tyle często, aby jego wkład w pulę był opłacalny. Ustawiając trudność zdobywania udziałów na niższym poziomie, pula monitoruje ilość pracy wykonana przez każdego górnika. Za każdym razem, gdy górnik uczestniczący w puli znajdzie hash nagłówka bloku, który jest mniejszy, niż trudność puli, musi udowodnić, że wykonał zadanie hashowania w celu znalezienia danego wyniku. Jeszcze ważniejsza jest praca wykonana w celu znalezienia udziałów oraz jej wkład wniesiony w sposób mierzalny statystycznie w ogólny wysiłek włożony w znalezienie wartości hash poniżej celu sieci bitcoin. Tysiące górników podejmując próby znalezienia niskiej wartości hash w końcu znajdą jedną, dostatecznie niską, aby spełnić cel sieci bitcoin.

Powróćmy na chwilę do porównania do gry w kości. Jeżeli uczestnicy gry rzucają kości w celu uzyskania wyniku poniżej czterech (ogólny stopień trudności sieci), pula może określić łatwiejszy cel wyliczając, ile razy pula graczy uzyskała wynik poniżej ośmiu. Kiedy pula graczy uzyska taki wynik (wartość docelowa puli udziałów), jej uczestnicy zaczynają zdobywać udziały, ale nie wygrywają gry, ponieważ nie osiągnęli celu gry (tj. wartości mniejszej od czterech). Gracze w puli osiągną cel puli o wiele częściej zdobywając udziały regularnie, nawet jeśli nie zrealizują bardziej trudnego celu. Jakim jest wygranie gry.

Od czasu do czasu, jeden z graczy puli uzyska łączny wynik poniżej czterech i wtedy pula wygrywa. Następnie, zyski mogą być rozzielone pomiędzy graczy puli w zależności od ich udziałów. Nawet, jeśli cel osiem lub mniej nie okazał się wygrywający, stanowił uczciwy sposób oceny rzutów dla puli graczy, która od czasu do czasu generuje rzut o wartości poniżej czterech.

Podobnie, pula miningowa określi poziom trudności puli, która zapewni, że pojedynczy górnik w puli będzie w stanie znaleźć wartości hashujące nagłówka bloku, które będą niższe niż poziom trudności puli na tyle często, aby zdobyć udziały. Od czasu do czasu, jedna z tych prób pozwoli wygenerować wartość hashującą nagłówka bloku, która będzie poniżej wartości docelowej sieci bitcoin sprawiając, że stanie się ona ważnym blokiem, a cała pula odnosi zwycięstwo.

Pule zarządzane

Większość pul wydobywczych jest “zarządzanych”, co oznacza, że istnieje firma lub osoba fizyczna, która zarządza serwerem puli. Właściciel serwera puli nazywany jest *operatorem puli*; to on dokonuje podziału procentowego na rzecz górników zgromadzonych w puli.

Serwer puli uruchamia wyspecjalizowane oprogramowanie oraz protokół pool-miningowy koordynujący działania górników uczestniczących w puli. Serwer ten jest połączony do jednego lub więcej pełnych węzłów i posiada bezpośredni dostęp do pełnej kopii bazy danych łańcucha blokowego. To umożliwia serwerowi puli weryfikację bloków oraz transakcji w imieniu górników zwalniając ich z obowiązku prowadzenie pełnego węzła. Dla górników uczestniczących w puli jest to ważna przesłanka, ponieważ pełny węzeł wymaga posiadania dedykowanego komputera z pamięcią nieulotną co najmniej 15 do 20 GB r(dysk) oraz co najmniej 2 GB pamięci (RAM). Ponadto, oprogramowanie działają na pełnych węzłach wymaga monitoring, opieki technicznej oraz częstej aktualizacji wersji. Wszelkie przestoje w wyniku braku działań konserwacyjnych i braku środków będą ze szkodą dla rentowności górnika. Dla wielu górników możliwość wydobywania bez konieczności posiadania pełnego węzła to kolejna korzyść wynikająca z przyłączenia się do zarządzanej puli.

Górnicy uczestniczący w puli łączą się z serwerem puli za pośrednictwem protokołu miningowego, np. Stratum (STM) lub GetBlockTemplate (GBT). Starszy standard znany pod nazwą GetWork (GWK) jest w większości przypadków przestępco i niewykorzystywany od końca 2012r., ponieważ nie wspiera wydobycia z prędkościami powyżej 4 GH/s. Obydwaj protokoły STM oraz GBT tworzą blok wzorców zawierających modele kandydujących nagłówków bloku. Serwer puli konstruuje blok kandydujący agregując transakcje, dodając transakcje coinbase (wraz z dodatkowym miejscem na wartości jednorazowe), obliczając korzeń drzewka skrótów oraz łącząc go z wartością hash poprzedniego bloku. Nagłówek bloku kandydującego jest następnie przesyłany do każdej puli górników jako wzorzec. Każdy górniczy wykorzystuje je w procesie wydobycia stosując niższy poziom trudności niż trudność sieci bitcoin i przesyła wszystkie skuteczne wyniki z powrotem do serwera puli, aby zyskać udział.

P2Pool

Zarządzane pule dają możliwość oszukania operatora puli, który może skierować działania puli w stronę podwójnych transakcji wydatkowania lub unieważnić bloki (patrz: "Ataki na consensus"). Dodatkowo, zcentralizowane serwery puli stanowią jeden punkt niepowodzenia. Jeżeli serwer puli jest uszkodzony lub jego działanie jest spowolnione wskutek ataku odmowy usługi górnicy nie mogą prowadzić pracy. W roku 2011, aby rozwiązać kwestię centralizacji, przedstawiono i wdrożono nową metodę: P2Pool – jest to pula wydobywcza typu peer-to-peer bez centralnego operatora.

P2Pool działa poprzez decentralizację funkcji serwera puli wdrażając system równoległy przypominający łańcuch blokowy nazywany *share chain* (*łańcuchem udziałów*). łańcuch udziałów jest to łańcuch blokowy działający na niższym poziomie trudności w porównaniu z łańcuchem blokowym bitcoin. łańcuch udziałów umożliwia górnikom uczestniczącym w puli współpracę w ramach puli zdecentralizowanej poprzez wydobywanie udziałów z łańcucha udziałów z prędkością jednego bloku udziałowego co 30 sekund. Każdy z bloków łańcucha rejestruje proporcjonalną nagrodę w postaci udziałów dla pracujących górników, którzy wnoszą swój wkład i przenoszą udziały do przodu z poprzedniego bloku. Kiedy jeden z bloków udziałów także osiągnie trudność docelową sieci bitcoin, jest on rozsyłany i uwzględniany w łańcuchu blokowym bitcoin nagradzając wszystkich, którzy przyczynili się do zdobycia udziałów poprzedzających, wygrywający blok udziałowy. Zasadniczo, zamiast śledzenia serwera puli zawierającego informacje o udziałach i nagrodach górników, łańcuch udziałowy umożliwia wszystkim górnikom śledzenie wszystkich udziałów przy wykorzystaniu zdecentralizowanego mechanizmu consensusu takie jak np. mechanizm consensusu bitcoin.

Wydobycie za pomocą P2Pool jest bardziej złożone niż górnictwo w oparciu o pulę, ponieważ wymaga, aby pula górników posiadała dedykowane komputery z wystarczającą ilością miejsca na dysku, pamięci oraz przepustowości łączy, aby wspierać pełny węzeł bitcoin oraz oprogramowanie węzła P2Pool. Górnicy P2Pool łączą swoje oprogramowanie wydobywcze z lokalnym węzłem P2Pool, który stymuluje funkcje serwera puli wysyłając wzorce bloków do urządzeń wydobywczych. W P2Pool, poszczególni górnicy puli konstrują swoje własne bloki kandydujące agregując transakcje podobnie jak to ma miejsce w przypadku górników solo, ale potem dokonują wydobycia wspólnie na łańcuchu udziałów. P2Pool oparte jest na podejściu hybrydowym,

które zapewnia drobniejsze (rozproszone) wypłaty w porównaniu z górnikami solo, ale bez dawani zbyt dużej kontroli operatorowi puli podobnych do puli zarządzanych.

W ostatnim czasie, liczba uczestników P2Pool znacznie wzrosła, ponieważ koncentracja wydobycia w pulach osiągnęła poziom zagrażający 51% ryzykiem ataków (patrz: "[Ataki na consensus](#)"). Dalszy rozwój protokołu P2Pool ma na celu wyeliminowanie potrzeby posiadania pełnego węzła i tym samym dalsze ułatwienie prowadzenia zdecentralizowanego górnictwa.

Ataki na consensus

Mechanizm consensus Bitcoin jest, przynajmniej teoretycznie, podatny na ataki ze strony górników (lub puli) podejmujących próby wykorzystania swojej siły hashującej w nieuczciwych lub destrukcyjnych celach. Jak mieliśmy okazję przekonać się mechanizm consensusu jest zależny od posiadania grupy górników, którzy działając z uczciwych побudek oraz we własnym interesie. Nie mniej jednak, jeżeli górnik lub grupa górników mogą osiągnąć znaczący udział w sile wydobywczej mogą oni przypuścić atak na mechanizm consensusu w celu złamania zabezpieczeń i dostępności sieci bitcoin.

Należy pamiętać, że ataki na consensus mogą jedynie mieć wpływ na consensus osiągany w przyszłości lub najwyżej na najbliższą przeszłość (dziesiątki bloków). Księgi bitcoin stają się coraz bardziej niezmienialne w miarę upływu czasu. Chociaż teoretycznie rozwidlenie można osiągnąć na dowolnej głębokości w praktyce moc obliczeniowa niezbędne do wymuszenia bardzo głębokiego rozwidlenia jest ogromna co sprawia, że stare bloki są praktycznie niezmienialne. Ataki na consensus nie mają także wpływu na bezpieczeństwo prywatnych kluczy oraz algorytm podpisu (ECDSA). Atak na consensus nie może przyczyniać się kradzieży bitcoinów, wydatkowanych bez podpisu, przekierowywania bitcoinów lub wprowadzania innych zmian do danych transakcji zawieranych w przeszłości lub rejestrów właścicieli. Ataki na consensus mogą jedynie mieć wpływ na najnowsze bloki i powodować zakłócenia typu odmowa usług (denial-of-service disruptions) w momencie tworzenia bloków w przyszłości.

Jeden ze scenariuszy ataku na mechanizm consensusu nazywany jest "atakiem 51%", W tym scenariuszu grupa górników kontrolująca większość (51%) łącznej mocy hashującej sieci spiskuje w celu dokonania ataku na bitcoin. Mając zdolność wydobycia większości bloków, atakujący górnicy mogą tworzyć celowe "rozwidlenia" w łańcuchu blokowym oraz transakcje podwójnego wydatkowania lub wykonywać ataki typu denial-of-service skierowane na wybrane transakcje lub adresy. Ataki rozwidlenia/podwójnego wydatkowania mają miejsce wtedy, gdy atakujący wcześniej potwierdził bloki jako unieważnione tworząc rozwidlenie poniżej nich i rekonwertując je na alternatywnym (przemiennym) łańcuchu. Mając wystarczającą moc, atakujący może unieważnić sześć lub więcej bloków po kolei powodując, że transakcje, które były uznane za niezmienialne (sześć potwierdzeń) stają się nieważne. Warto zauważyć, że podwójne wydatkowanie może dotyczyć jedynie własnych transakcji atakującego, dla których może on wygenerować ważny podpis. Podwójne wydatkowanie własnych transakcji przynosi korzyści, jeżeli atakujący może dokonać nieodwracalnej wymiany lub pozyskać towar bez płacenia.

Zapoznajmy się teraz z przykładem praktycznym ataku 51%. W pierwszym rozdziale, zapoznaliśmy się z transakcjami pomiędzy Alice i Bob dotyczącej zakupu filiżanki kawy. Bob, właściciel kawiarni, chce przyjąć płatność za kawę bez konieczności oczekiwania na potwierdzenie (wydobycie bloku), ponieważ ryzyko podwójnego wydatkowania sumy należnej za kawę jest niskie w porównaniu z wygodą szybkiej obsługi klienta. Jest to zjawisko zbliżone do praktyki wykorzystywanej w kawiarniach, które akceptują płatności kredytowe kartą kredytową bez podpisu na kwoty poniżej 25 dolarów, ponieważ ryzyko ograniczenia zwrotnego jest znikome, a koszt opóźnienia transakcji w związku z koniecznością uzyskania podpisu są porównywalnie wyższe. Dla porównania sprzedaż bardziej kosztownych towarów wiąże się z ryzykiem ataku podwójnego

wydatkowania, w którym kupujący wysyła konkurencyjną transakcję, która wydatkuje te same bloki wejściowe (UTXO) anulując płatności na rzecz sprzedawcy. Atak podwójnego wydatkowania może być przeprowadzony na dwa sposoby: albo zanim transakcja zostanie potwierdzona, albo kiedy atakujący wykorzystuje rozwidlenie łańcucha blokowego w celu cofnięcia danych kilku bloków. Atak 51% umożliwia atakującym podwójne wydatkowanie swoich własnych transakcji w nowym łańcuchu tym samym odwracając odpowiednie transakcje w starym łańcuchu.

W naszym przykładzie złośliwy atakujący - Mallory udaje się do galerii Carol i dokonuje zakupu pięknego tryptyku malarstwa przedstawiającego Satoshi Nakamoto, jako Prometeusza. Carol sprzedają obrazy zatytułowane "The Great Fire" (Wielki ogień) za 250 000 dolarów w przeliczeniu na bitcoiny Mallory'emu. Zamiast czekać na sześć lub więcej potwierdzeń transakcji, Carol pakuje dzieła i wręcza je Mallory'emu po zaledwie jednym potwierdzeniu. Mallory ma wspólnika - Paula, który uczestniczy w dużej puli wydobywczej i przeprowadza atak 51%, gdy tylko transakcja Mallory'ego zostanie umieszczona w bloku. Paul kieruje pulę miningową w celu odzyskania bloku tej samej wysokości jako bloku zawierającego transakcję Mallory'go zastępując płatność Mallory na rzecz Carol transakcją, która stanowi dwukrotne wydatkowanie tego samego bloku wejściowego jako płatności Mallory'ego. Podwójne wydatkowanie transakcji pochłania tą samą UTXO i wraca do portfela Mallory'ego zamiast płatności na rzecz Carol, zasadniczo umożliwiając Mallory'emu zachowanie bitcoinów. Następnie Paul przekierowuje pulę miningową w celu wydobycia z puli miningowego dodatkowego bloku, aby sprawić, że łańcuch zawierający podwójnie wydatkowaną transakcję będzie dłuższy niż oryginalny łańcuch (powodując rozwidlenie poniżej bloku zawierającego transakcję Mallory'ego). Kiedy łańcuch blokowy znajduje rozwiązanie na korzyść nowego (dłuższego) łańcucha, podwójnie wydatkowania transakcja zastępuje oryginalną płatność na rzecz Carol. Carol obecnie nie ma trzech obrazów i jednocześnie utraciła należność wyrażoną w bitcoinach. Przez cały czas uczestnicy puli górniczej Paula mogą nie mieć świadomości, że podejmowana jest próba podwójnego wydatkowania, ponieważ dokonują wydobycia posługując się automatycznymi górnikami i nie mają możliwości monitorowania każdej transakcji lub bloku.

Aby zapobiegać tego typu atakom, osoba prowadząca sprzedaż towarów o dużej wartości musi odczekać, aż pojawi się co najmniej sześć potwierdzeń zanim produkt zostanie przekazany kupującemu. Sprzedający może także użyć konta escrow z wieloma podpisami i także odczekać na kilka potwierdzeń po utworzeniu konta escrow. Im więcej potwierdzeń tym trudniej jest unieważnić transakcję atakiem 51%. W przypadku towarów o znacznej wartości płatności w bitcoinach wciąż mogą stanowić wygodny i skuteczny sposób rozliczenia, nawet, jeśli kupujący będzie musiał poczekać 24 h na dostawę, co oznacza 144 potwierdzenia.

W uzupełnieniu do ataku podwójnego wydatkowania istnieją także inne scenariusze ataków na consensus w postaci domowy usługi dla szczególnych uczestników systemu bitcoin (specyficznych adresów bitcoin). Atakujący posiadający większość mocy wydobywczej może po prostu zignorować wybrane transakcje. Jeżeli zostaną one umieszczone w bloku wydobytym przez innego górnika, atakujący może celowo rozwidlić i odzyskać ten blok, aby wykluczyć określone transakcje. En rodzinny atak może skutkować trwałą odmową usług dla danego adresu lub grupy adresów rakiem dłujo, jak atakujący kontroluje większość mocy wydobywczej.

Wbrew nazwie, scenariusz ataku 51% wcale nie wymaga posiadania 51% procent mocy hashującej. Tak naprawdę tego typu atak może być dokonany w oparciu o mniejszą moc hashującą. Próg 51% procent to po prostu poziom, na którym atak ma niemal całkowitą gwarancję sukcesu. Atak na consensus jest zasadniczo swoistą odmianą przeciągania liny dla kolejnego bloku, przy czym "silniejsza" grupa ma większe szanse na wygraną. Dysponując mniejszą mocą hashującą, prawdopodobieństwo sukcesu jest niższe, ponieważ inni górnicy kontrolują pokolenie wybranych bloków za pomocą swojej "uczciwej" siły wydobywczej. Jedno z podejść zakłada, że im większą mocą dysponuje atakujący, tym dłuższe rozwidlenie może utworzyć działające celowo i tym więcej bloków nowo wprowadzonych do łańcucha może unieważnić i tym więcej bloków może kontrolować w przyszłości. Grupy zajmujące się zabezpieczeniami wykorzystując model modelingu

statystycznego doszły do wniosku, że różne typy ataków na consensus są możliwe mając do dyspozycji zaledwie 30% mocy hashującej.

Masywny wzrost łącznej mocy hashującej uczynił bitcoin niewrażliwym na ataki pojedynczego górnika – twierdzenie to jest kwestionowane. Nie ma nawet potencjalnej możliwości, aby górnik solo przejął kontrolę na choćby 1% łącznej mocy wydobywczej. Nie mniej jednak centralizacja kontroli zainicjowana przez pule wydobywcze wiąże się z ryzykiem ataków for-profit ze strony przez operatora puli wydobywczej. Operator puli zarządzanej kontroluje budowę bloków kandydujących i jednocześnie kontroluje decyduje, które transakcje mają być w niej umieszczone, to daje operatorowi puli moc wyłączenia transakcji lub wprowadzania transakcji podwójnego wydatkowania. Jeżeli tego typu nadużycia są realizowane na ograniczoną skalę i w sposób subtelny, operator puli może w przekonywujący sposób czerpać korzyści z ataków na consensus bez ujawnienia.

Nie wszyscy atakujący będą kierować się zyskiem. Jeden z potencjalnych scenariuszy ataku zakłada, że atakujący zamierza naruszyć sieć bitcoin bez żadnych korzyści w związku z tym naruszeniem. Złośliwy atak ukierunkowany na sparaliżowanie systemu bitcoin może wymagać ogromnych nakładów oraz zmiany planów, ale może być w sposób przekonywający przeprowadzony przez stabilnego i sponsorowanego ze środków publicznych atakującego. Z drugiej strony, dofinansowany atakujący może przypuścić atak na consensus bitcoin jednocześnie gromadząc sprzęt górniczy, naruszając bezpieczeństwo operatorów puli oraz atakując inne pule odmową usług. Wszystkie z wymienionych scenariuszy są teoretycznie możliwe, ale w coraz większym stopniu stają się niepraktyczne, ponieważ ogólna moc hashująca sieci bitcoin stale rośnie wykładniczo. Najnowsze rozwiązania w dziedzinie bitcoinów takie, jak wydobycie typu P2Pool mają na celu dalszą decentralizację kontroli wydobycia sprawiając, że atak na consensus bitcoin stanie się jeszcze bardziej trudny.

Niewątpliwie poważny atak na consensus może podważyć zaufanie do bitcoin w perspektywie krótkookresowej potencjalnie przyczyniając się do obniżenia ceny. Nie mniej jednak zarówno sieć bitcoin, jak i oprogramowanie są stale rozwijane, zatem ataki na consensus mogą spotkać się z natychmiastowym przeciwdziałaniem sprawiając, że stanie się bardziej trudny do zdobycia, mocniejszy, bardziej dyskretny i stabilny niż kiedykolwiek przedtem.

ROZDZIAŁ 9

Łańcuchy, waluty i aplikacje alternatywne

System *Bitcoin* to wynik 20 lat prac badawczych w dziedzinie systemów rozproszonych oraz walut, które przyczyniły się do opracowania nowych, rewolucyjnych technologii na kosmicznym poziomie w postaci mechanizmu zdecentralizowanego *consensusu* opartego na algorytmie *proof of work*. Wynalazek ten, stanowiący rdzeń systemu *bitcoin* utorował drogę do licznych innowacji w dziedzinie walut, usług finansowych, ekonomiki, systemów rozproszonych, wyborczych, ładu korporacyjnego oraz kontraktów.

W tym rozdziale będziemy zajmować się licznymi odmianami *bitcoinów* oraz łańcuchów blokowych: alternatywnymi łańcuchami, walutami oraz aplikacjami zbudowanymi od daty wprowadzenia systemu w roku 2009. Dużą część dyskusji poświęcimy alternatywnym *coinom*, nazywanymi także *alt coinami*; są to waluty cyfrowe zaimplementowane przy wykorzystaniu tych samych wzorów, jakich użyto w przypadku *bitcoinów*, ale w oparciu o całkowicie odmienny łańcuch blokowy oraz sieć.

Dla każdego typu *alt coin* wymienionego w niniejszym rozdziale, 50 lub więcej nie zostanie wspomnianych wywołując falę oburzenia po stronie ich twórców oraz fanów. Celem niniejszego rozdziału nie jest dokonanie oceny lub klasyfikacji *alt coinów*, czy też sporządzenie listy najważniejszych systemów tego typu w oparciu o ocenę subiektywną. Zamiast tego, skupimy się na przedstawieniu kilku przykładów przedstawiających rozległość i różnorodność tego swoistego ekosystemu kładąc nacisk na pierwszy model danego typu lub każde unowocześnienie lub istotną zmianę. Niektóre z najciekawszych przykładów to tak naprawdę całkowite porażki z perspektywy systemu monetarnego, co zapewne czyni je jeszcze bardziej interesującymi obiektami analizy i potwierdza raz jeszcze cel niniejszego podręcznika, którym nie jest pełnenie funkcji przewodnika inwestycyjnego.

Zważywszy, że codziennie pojawiają się na rynku coraz to nowe monety niepominiecie tych ważniejszych może okazać się zadaniem niemożliwym do realizacji – nawet tych o znaczeniu historycznym. Poziom innowacyjności sprawia, że dziedzina ta staje się niezwykle eksytyująca i czyni niniejszy rozdział niekompletnym i nieaktualnym tuż po zakończeniu jego opracowania.

Podział taksonomiczny walut i łańcuchów alternatywnych

Bitcoin to projekt typu *open source*, a jego kod stanowi podstawę wielu rodzajów oprogramowania. Najbardziej powszechnym typem oprogramowania opracowanego na bazie kodu źródłowego *bitcoin* są zdecentralizowane waluty alternatywne nazywane także *alt coinami*, które wykorzystują te same podstawowe komponenty w celu implementacji walut cyfrowych.

Istnieje kilka warstw protokołów, które są implementowane na szczeblu łańcucha *bitcoin*. Te *meta coin*, *meta łańcuchy* lub *aplikacje łańcuchów blokowych* wykorzystują łańcuch, jako platformę aplikacji lub rozszerzenie protokołu *bitcoin* poprzez dodanie kolejnych warstw protokołu. Jako przykład można wymienić *Colored Coins*, *Mastercoin*, czy *Counterparty*.

W kolejnym rozdziale zajmiemy się oceną kilku ważniejszych alt coinów, m.in. *Litecoin*, *Dogecoin*, *Freicoin*, *Primecoin*, *Peercoin*, *Darkcoin* oraz *Zerocoin*. Systemy te odgrywają istotną rolę z przyczyn historycznych, bądź dlatego, że są doskonałymi przykładami szczególnego rodzaju innowacji *alt coinowych*, a nie dlatego, że są to najcenniejsze lub “najlepsze” alt coin.

Poza alt coinami, istnieje kilka alternatywnych wdrożeń łańcuchów blokowych, które nie są prawdziwymi "monetami" i które nazywam *alt chainami* (*alt chainami/alt łańcuchami*). Te *alt chainy* zwierają algorytm *consensusu* oraz rozproszonej księgi, która stanowi platformę wykorzystywana w celu zawierania kontraktów, rejestracji nazw(isk) lub innych. *Alt chainy* wykorzystują te same zasadnicze komponenty, a niekiedy także waluty lub tokeny, jako mechanizmy płatności, ale ich zasadniczym celem nie jest służenie jako waluta. W dalszej części przyjrzymy się systemom *Namecoin*, *Ethereum* oraz *NXT*, jako przykładowym *alt chainom*.

Poza mechanizmem consensusu *proof-of-work* wykorzystywanym w systemie bitcoin istnieją alternatywne rozwiązania w postaci eksperymentalnych protokołów opartych na algorytmie *proof of resource* oraz *proof of publishing*. Omówimy rozwiązania *MaidSafe* oraz *Twister*, jako przykładowe mechanizmy consensusów.

Na koniec zajmiemy się także kilkoma konkurentami bitcoin oferującymi waluty cyfrowe i usługi cyfrowych sieci płatniczych, ale bez wykorzystywania zdecentralizowanej księgi lub mechanizmu consensusu, opartego na *proof of work*, takie jak *Ripple* i wiele innych. Te technologie nieoparte na łańcuchach blokowych wykraczają poza ramy niniejszego opracowania i nie zostały ujęte w tym rozdziale.

Platformy meta coin

Meta coin i meta chainy to warstwy oprogramowania zaimplementowane na szczycie systemu bitcoin, zawierające nakładkę *currency-inside-a-currency* lub platformy/protokołu wewnętrz systemu bitcoin. Te warstwy funkcji stanowią rozszerzenie protokołu bitcoin i uzupełniają cechy oraz możliwości systemu poprzez zakodowanie dodatkowych danych wewnętrz transakcji oraz adresów bitcoin. Pierwsza implementacja meta coinów była oparta na różnego rodzaju włamaniach, aby dodać metadane do łańcucha blokowego bitcoin wykorzystując adresy bitcoin w celu zakodowania danych lub niewykorzystanych wcześniej pól transakcji (np. pole sekwencji transakcji), aby zakodować metadane dotyczące dodanych warstw protokołu. Od chwili wprowadzenia skryptu opakowania transakcji *OP_RETURN*, meta coin były w stanie rejestrować metadane bezpośrednio w łańcuchu blokowym; większość z nich obecnie migruje w stronę ich wykorzystywania.

Colored Coiny

Colored coin to meta protokół, który nadpisuje informacje na małe kwoty wyrażone w *bitcoincach*. "*Colored coin*" (z ang. kolorowa moneta) to kwota, której zadanie zostało zmienione tak, aby wyrażać inne aktywa (środki). Wyobraźmy sobie na przykład banknot o nominale 1 dolara i umieszczenie na nim pieczęci z informacją „To jest certyfikat 1 udziału w Acme Inc.” Banknot ten spełnia dwa cele: jest wykorzystywany, jako jednostka walutowa i jednocześnie, jako świadectwo udziałowe. Ponieważ jego wartość jest większa, jeżeli jest wykorzystywany jako świadectwo udziałowe, posiadacz nie jest zainteresowany jego wykorzystywaniem w celu zakupu batoników, zatem praktycznie środek ten nie jest już użyteczny, jako środek płatniczy. *Colored coin* działają w taki sam sposób zamieniając określona, niewielką kwotę *bitcoinów* na zbywalne świadectwo, które stanowi inny środek. Zwrot “kolorowy” odnosi się do pomysłu nadawania specjalnego znaczenia poprzez dodanie odpowiedniego atrybutu - *color—it* – jest to metafora, a nie rzeczywiście przypisany kolor. *Colored coin* nie mają żadnego koloru.

Colored coin są zarządzanie za pośrednictwem wyspecjalizowanych portfeli, które rejestrują i interpretują metadane stanowiące część *colored bitcoinów*. Wykorzystując ten portfel, użytkownik dokona zamiany pewnej kwoty *bitcoinów* z walutą bezbarwnej na kolorową poprzez dodanie etykiety, która ma szczególne znaczenie. Na przykład, etykieta może odzwierciedlać świadectwa udziałowe, kupony od kapitału udziałowego, nieruchomości, towary oraz tokeny wymienialne. Użytkownicy *colored* podejmują decyzję o przypisaniu i interpretacji znaczeniu “koloru” skojarzonego z dana monetą. Aby zabarwić monety, użytkownik definiuje

skojarzone metadane, np. rodzaj emisji, możliwość podziału na mniejsze jednostki, symbol oraz opis, a także inne związane informacje. Po wprowadzeniu koloru, monety te można kupować lub sprzedawać, dzielić lub otrzymywać wypłaty dywidend. Kolorowe monety można także “odbarwiać” usuwając szczególne skojarzenia i umarzać je zgodnie z wartością nominalną danego *bitcoina*.

Aby zaprezentować sposób wykorzystania tego systemu, utworzyliśmy zbiór 20 *colored coinów* oznaczonych symbolem “MasterBTC”, które reprezentują kupony dołączone do darmowego egzemplarza tej książki w **Przykładzie 9-1**. Każdą jednostkę MasterBTC reprezentowaną przez te *colored coin'y* można teraz sprzedać lub przekazać dowolnym użytkownikom *bitcoinów* za pośrednictwem portfeli *colored-coin-capable*, a następnie dokonywać ich transferu do innych lub umorzyć u wystawcy w oparciu o darmową kopię księgi. Ten przykład podany jest [tutaj](#).

Przykład 9-1. Profil metadanych zapisanych w colored coinach w postaci kuponu darmowej kopii księgi

```
{  
  "source_addresses": [  
    "3NpZmvSPLmN2cVFw1pY7gxEA/PCVfnWfVD"  
,  
    "contract_url": "https://www.coinprism.info/asset/  
3NpZmvSPLmN2cVFw1pY7gxEA/PCVfnWfVD",  
    "name_short": "MasterBTC",  
    "name": "Free copy of \"Mastering Bitcoin\"",  
    "issuer": "Andreas M. Antonopoulos",  
    "description": "This token is redeemable for a free copy of the book \"Mastering  
Bitcoin\"",  
    "description_mime": "text/x-markdown; charset=UTF-8",  
    "type": "Other",  
    "divisibility": 0,  
    "link_to_website": false,  
    "icon_url": null,  
    "image_url": null,  
    "version": "1.0"  
}
```

Mastercoiny

Mastercoiny to warstwa protokołu znajdująca się na szcycie systemu bitcoin, która wspiera platformę w zakresie różnych aplikacji stanowiących jego rozszerzenie. System *Mastercoin* wykorzystuje walutę MST jako token do zawierania transakcji w *Mastercoinach* chociaż z założenia jego główną funkcją nie jest rola waluty. Jest to platforma, na której buduje się inne elementy takie jak waluty użytkownika, tokeny do inteligentnego zarządzania nieruchomościami, zdecentralizowana giełda towarowa oraz kontrakty. *Mastercoiny* należy traktować jako protokół warstwy aplikacji znajdujący się na szcycie warstwy transportującej transakcje finansowe bitcoin podobnie jak ma to miejsce w przypadku HTTP działającego na szcycie / wierzchu TCP.

System *Mastercoin* działa przede wszystkim w oparciu o transakcje wysłane do i ze specjalnych adresów bitcoin nazywanych adresami “exodus”(1ExoDusjGwnjZUyKkxZ4UHEf77z6A5S4P) podobnie jak HTTP wykorzystuje szczególne porty TCP (port 80) w celu rozróżnienia ruchu własnego od pozostałego traffiku TCP. Protokół *Mastercoin* stopniowo odchodzi od wykorzystywania wyspecjalizowanych adresów exodus oraz systemu *multi-signatures* w stronę wykorzystywania operatora bitcoinów OP_RETURN w procesie szyfrowania metadanych transakcyjnych.

Counterparty

Counterparty to kolejna warstwa protokołu zaimplementowana na szcycie systemu bitcoin. *Counterparty* aktywuje waluty użytkownika, tokeny zbywalne, instrumenty finansowe, zdecentralizowane giełdy aktywów i inne elementy systemu. *Counterparty* jest implementowane głównie w oparciu o operatora OP_RE TURN przy

wykorzystaniu języka skryptów *bitcoin* w celu zapisania metadanych wzbogacających transakcje *bitcoin* o dodatkowe znaczenie. *Counterparty* wykorzystuje walutę XCP jako token do zawierania transakcji *Counterparty*.

Alt Coiny

Większość *alt coinów* pochodzi z kodu źródłowego *bitcoin* nazywanego także „rozwidleniem”. Niektóre są implementowane “od podstaw” w oparciu o model łańcucha blokowego, jednakże bez wykorzystania któregokolwiek z fragmentów kodu źródłowego. *Alt coin* oraz *alt chainy* (omówione w następnej części) stanowią zarówno odrębne implementacje technologii łańcucha blokowego, przy czym oba systemy działają w oparciu o swoje własne łańcuchy blokowe. Różnica polega na tym, że *alt coin* są głównie wykorzystywane, jako waluta, podczas gdy *alt chainy* są wykorzystywane w innych celach i ich główną rolą nie jest funkcja walutowa.

Ścisłe mówiąc, pierwsze główne rozwidlenie “alt” kodu *bitcoina* nie było *alt coinem*, ale *alt chainem Namecoin*, który zostanie omówiony w kolejnej części.

Jak wynika z oświadczenia, pierwszy *alt coin*, który stanowił rozwidlenie *bitcoina* pojawił się w sierpniu 2011r.; nazywał się *IXCoin*. *IXCoin* zmodyfikował kilka parametrów *bitcoina* w szczególności przyspieszając się do stworzenia waluty poprzez zwiększenie nagrody do 96 *coinów* za blok.

We wrześniu 2011 roku, wprowadzono *Tenebrix*. *Tenebrix* to pierwsza kryptowaluta, która wykorzystywała alternatywny algorytm *proof-of-work*, a mianowicie *scrypt* – algorytm pierwotnie zaprojektowany w celu rozszerzania haseł (odporność na ataki *brute-force*). *Tenebrix* miał przyczynić się do powstania monet, które byłyby odporne na wydobycie za pomocą GPU oraz ASIC przy wykorzystaniu algorytmu wymagającego dużej ilości pamięci (*memory-intensive*). System *Tenebrix* nie odniósł sukcesu, jako waluta, ale stanowił odpowiednią podstawę dla systemu *Litecoin*, który odniósł ogromny sukces i przyczynił się do powstania setek klonów.

Litecoin, w uzupełnieniu do wykorzystywania skryptów w charakterze algorytmu *proof-of-work* ma także zaimplementowany krótszy czas generowania bloku, który docelowo ma wynosić 2,5 minuty zamiast 10 minut obecnie niezbędnych w *bitcoenie*. Powstała w ten sposób waluta jest przedstawiana jako “srebrny ekwiwalent złota *bitcoina*” i w zamyśle ma służyć, jako lekka waluta alternatywna. Z uwagi na krótki czas potwierdzenia oraz limit wyemitowanej waluty wynoszący 84 miliony, wielu zwolenników *Litecoin* jest przekonanych, że jest to waluta lepsza w przypadku transakcji detalicznych niż *bitcoin*.

Liczba *alt coinów* stale wzrosła w roku 2011 i 2012, wykorzystując systemy *bitcoin* lub *Litecoin*. Do roku 2013, na rynku istniało 20 *alt coinów* rywalizujących ze sobą o dominującą pozycję na rynku. Pod koniec 2013r., ich liczba wzrosła gwałtownie aż do 200, co sprawiło, że rok ten został nazwany “rokiem *alt coinów*”. Rozwój systemu *alt coinów* kontynuowany był przez cały rok 2014; w chwili powstawania tego materiału na rynku było dostępnych ponad 500 *alt coinów*. Ponad połowa obecnie dostępnych *alt coinów* to klony *Litecoinów*.

Tworzenie *alt coinów* jest łatwe, dlatego obecnie mamy ich już 500. Większość z nich różni się nieco od *bitcoinów* i nie stanowią ciekawego tematu do rozostrań. Wiele z nich to po prostu próba wzbogacenia ich twórców. Pośród naśladowców oraz systemów typu *pump-and-dump*, istnieje jednak kilka zauważalnych wyjątków i bardzo znaczących udoskonaleń. *Alt coin* te wykorzystują zasadniczo różne podejście i mają poważny wkład w rozwój wzorów *bitcoiniowych*. Istnieją trzy zasadnicze obszary pozwalające odróżnić *alt coin* od *bitcoinów*:

- odmienna polityka monetarna
- odmienny mechanizm *proof of work* lub *consensusu*
- szczególnie cechy takie jak wysoki poziom anonimowości

W celu uzyskania bliższych informacji, patrz: [graficzna reprezentacja działania alt coinów i alt chainów](#).

Ocena Alt Coinów

Przy tak wielu *alt coinach* dostępnych na rynku, jak rozpoznać te, które są naprawdę godne naszej uwagi? Niektóre z nich mają szerokie zastosowanie i są wykorzystywane także, jako waluta. Inne służą jako laboratoria eksperymentujące z różnymi cechami oraz modelami monetarnymi.

Wiele z nich do plany umożliwiające szybkie wzbogacenie się dla swoich twórców. Aby dokonać odpowiedniej oceny *alt coinów*, przyjrzałem się ich definicjom oraz metryce rynkowej.

Poniżej znajduje się lista wybranych pytań, które warto zadać, aby stwierdzić jak bardzo system *alt coin* różni się od *bitcoinów*:

- Czy *alt coin* stanowi zasadnicze unowocześnienie?
- Czy zmiany te są wystarczająco zaawansowane, aby odciągnąć użytkowników od systemu *bitcoin*?
- Czy *alt coin* wypełnia ciekawą niszę rynkową albo aplikację?
- Czy *alt coin* może przyciągnąć dostateczną liczbę górników, aby zabezpieczyć system przed atakami *consensusu*?

Poniżej przedstawiamy kilka kluczowych wskaźników finansowych i rynkowych, które warto wziąć pod uwagę:

- Jaka jest ogólna kapitalizacja rynkowa systemu *alt coin*?
- Ile szacunkowych użytkowników/portfeli posiada *alt coin*?
- Ile handlowców akceptuje *alt coin*?
- Ile transakcji dziennie (wolumen) jest realizowanych w *alt coinach*?
- Jaka jest dzienna wartość zawieranych transakcji?

W dalszej części rozdziału będziemy się głównie koncentrować na charakterystyce technicznej oraz potencjalne innowacyjnym *alt coinów* opisanych przez pierwszy zestaw pytań.

Alternatywne parametry monetarne: Litecoin, Dogecoin, Freicoin

Bitcoin posiada kilka parametrów monetarnych, które nadają temu systemowi wyróżniającą go spośród innych systemów cechę defuracyjnej waluty o ustalonej liczbie jednostek. System składa się z 21 milionów głównych jednostek walutowych (lub 21 kwadrylionów jednostek o mniejszych nominałach), posiada geometrycznie zmniejszający się poziom emisji oraz rytm blokowy „serca systemu” o długości 10 minut, który odpowiada za kontrolującą prędkości potwierdzeń transakcji oraz generowania waluty. Wiele *alt coinów* zmodyfikowało główne parametry, aby realizować założenia różnych polityk monetarnych. Spośród setek *alt coinów*, do najbardziej znanych należą niżej wymienione odmiany.

Litecoin

Jedna z pierwszych generacji *alt coinów* uruchomiona w 2011 roku; *Litecoiny* to druga po *bitcoinie* waluta cyfrowa, która odniosła sukces na świecie. Jej zasadnicze cechy innowacyjne to wykorzystanie *scrypt-u* jako algorytmu *proof-of-work* (odziedziczonego po *Tenebrix*) oraz szersze/lżejsze parametry waluty.

- Czas generowania bloku: 2,5 minuty
- Łączna liczba jednostek waluty: 84 miliony walut do roku 2140
- Algorytm *consensusu*: *Scrypt proof of work*
- Kapitalizacja rynkowa: 160 milionów USD w połowie 2014 r.

Dogecoin

Dogecoiny zostały wprowadzone na rynek w grudniu 2013 r. w oparciu o rozwidlenie *Litecoin*. *Dogecoiny* odgrywają istotną rolę, ponieważ polityka monetarna tego systemu zakłada szybką emisję oraz bardzo wysoki górny limit w celu zachęcania do wydawania i wykorzystywania jako napiwki (*tipping*). *Dogecoin* jest także ważnym systemem, ponieważ został uruchomiony pierwotnie, jako żart, ale zyskał ogromną popularność pośród szerokiej rzeszy aktywnych uczestników zanim jego wykorzystanie zaczęło gwałtownie wygasnąć w roku 2014.

- Czas generowania bloku: 60 sekund
- łączna liczba jednostek waluty: 100 000 000 000 (100 bilionów) Doge do 2015
- Algorytm *consensusu*: *Scrypt proof of work*
- Kapitalizacja rynkowa: 12 milionów USD w połowie 2014r.

Freicoins

Freicoiny zostały wprowadzone na rynek w lipcu 2012 r. Jest to waluta typu *demurrage currency*, co oznacza, że ma ujemną stopę procentową w odniesieniu do przechowywanej wartości. Wartość zachowana (*value stored*) we *Freicoincach* jest szacowana na 4,5% opłaty APR, aby zachęcić do konsumpcji i jednocześnie zniechęcić do przechowywania środków. System *Freicoins* wyróżnia się tym, że oparty jest na polityce monetarnej, która stanowi dokładne przeciwieństwo deflacyjnej polityki charakteryzującej system *Bitcoin*. System *Freicoins* nie odniósł sukcesu jako waluta stanowi jednak ciekawy przykład różnorodności polityk monetarnych związanych z *alt coinami*.

- Czas generowania bloku: 10 minut
- łączna liczba jednostek waluty: 100 milionów jednostek do 2140 r.
- Algorytm *consensusu*: *SHA256 proof of work*
- Kapitalizacja rynkowa: 130 000 USD w połowie 2014

Innowacje w zakresie consensusu: Peercoin, Myriad, Blackcoin, Vericoins i NXT

Mechanizm *consensusu Bitcoin* jest oparty na pracy *proof of work* wykorzystującej algorytm SHA256. Pierwsze *alt coin* wprowadziły scrypt jako alternatywny algorytm proof-of-work oraz sposób na uczynienie wydobycia bardziej przyjaznym dla jednostek CPU oraz mniej podatnym na centralizację w oparciu o ASIC. Od tej pory, innowacje w zakresie mechanizmu *consensusu* dokonywały się w zawrotnym tempie. Niektóre z *alt coinów* wykorzystywały różnego rodzaju algorytmy w postaci *scrypt*, *scrypt-N*, *Skein*, *Groestl*, *SHA3*, *X11*, *Blake* i wielu innych. Niektóre z *alt coinów* łączyły w sobie wiele różnych algorytmów dla celów uzyskania *proof of work*. W roku 2013, mieliśmy okazję obserwować wdrożenie alternatywnego w stosunku do *proof of work* rozwiązania pod nazwą *proof of stake*, które stanowi podstawę dla wielu nowoczesnych *alt coinów*.

Proof of stake jest to system, w którym obecni posiadacze waluty mogą "wykorzystywać" ją jako zabezpieczenie podlegające oprocentowaniu. Przypomina to trochę świadectwo depozytowe (SD); uczestnicy mogą zatrzymywać część posiadanej waluty oraz odnosić zyski z inwestycji w postaci nowej waluty (emitowanej w postaci wypłat odsetkowych) oraz opłat transakcyjnych.

Peercoin

System *Peercoin* wprowadzono w sierpniu 2012 r. i jest to pierwsza waluta systemu *alt coin* wykorzystująca hybrydę algorytmów *proof-of-work* oraz *proof-of-stake* w celu wyemitowania nowej waluty.

- Czas generowania bloku: 10 minut
- łączna liczba jednostek waluty: Bez limitu

- Algorytm *consensusu*: (Hybryda) *proof-of-stake* z początkowym *proof-of-work*
- Kapitalizacja rynkowa: 14 milionów USD w połowie 2014r.

Myriad

System *Myriad* został wprowadzony w lutym 2014 r. i zasługuje na uwagę przede wszystkim dlatego, że wykorzystuje pięć różnych algorytmów *proof-of-work* (*SHA256d*, *Scrypt*, *Qubit*, *Skein* oraz *Myriad-Groestl*) jednocześnie, przy czym stopień trudności zmienia się dla każdego typu algorytmu w zależności od zaangażowania górników. Celem tego rozwiązania jest uodpornienie *Myriad* na specjalizację oraz centralizację ASIC, a także podniesienie odporności systemu na ataki na *consensusu* poprzez wprowadzenie konieczności ataku na różne algorytmy wydobywcze jednocześnie.

- Czas generowania bloku: średnio 30 sekund (docelowo 2,5 minuty dla każdego algorytmu wydobywczego)
- Łączna liczba jednostek waluty: 2 miliardy do 2024r.
- Algorytm *consensusu*: wieloalgorytmowy *proof-of-work*
- Kapitalizacja rynkowa: 120 000 USD w połowie 2014

Blackcoin

Blackcoin to system wprowadzony w lutym 2014 r., który działa w oparciu o algorytm *consensusu proof-of-stake*. Zasługuje na uwagę także ze względu na wprowadzenie “multipul” stanowiących odmianę pul górniczych, które mogą przełączać się pomiędzy różnymi *alt coinami* w sposób automatyczny w zależności od odnotowywanych zysków.

- Czas generowania bloku: 1 minuta
- Łączna liczba jednostek waluty: Bez limitu
- Algorytm *consensusu*: *Proof-of-stake*
- Kapitalizacja rynkowa: 3,7 miliona USD w połowie 2014 r.

VeriCoin

System *VeriCoin* wprowadzono w maju 2014r. Wykorzystuje on algorytm *consensusu proof-of-stake* ze zmiennym oprocentowaniem, które w sposób dynamiczny dostosowuje się w zależności od sił rynkowych kierujących popytem i podażą. Stanowi on także pierwszy *alt coin* z funkcją auto-wymiany na *bitcoiny* w celu realizacji płatności portfelowych w bitcoinach.

- Czas generowania bloku: 1 minuta
- Łączna liczba jednostek waluty: Bez limitu
- Algorytm *consensusu*: *Proof-of-stake*
- Kapitalizacja rynkowa: 1,1 miliona USD w połowie 2014r.

NXT

NXT (wymawiany jako „next”) to *altcoin* z “czystym” algorytmem *proof-of-stake* w tym sensie, że nie wykorzystuje on wydobycia w oparciu o *proof-of-work*. NXT to implementacja kryptowaluty od podstaw, która nie stanowi rozwidlenia *bitcoinów* lub innych *alt coinów*. NXT zawiera wiele zaawansowanych cech takich, jak nazwa rejestru (zbliżona do *Namecoin*), zdecentralizowana wymiana aktywów (zbliżona do *Colored Coinów*), zintegrowany i bezpieczny system komunikacji (podobny do *Bitmessage*) oraz delegowanie udziałów (umożliwiające przenoszenie *proof-of-stake* na innych uczestników). Uczestnicy systemu NXT nazywają go “next-generacją” lub kryptowalutą w wersji 2.0.

- Czas generowania bloku: 1 minuta
- Łączna liczba jednostek waluty: Bez limitu
- Algorytm *consensusu*: *Proof-of-stake*
- Kapitalizacja rynkowa: 30 milionów USD w połowie 2014r.

Dwucelowa innowacja wydobywca: Primecoin, Curecoin, Gridcoin

Algorytm *bitcoin proof-of-work* ma tylko jeden cel: zabezpieczenie sieci *bitcoin*. W porównaniu z tradycyjnymi zabezpieczeniami systemu płatności, koszty wydobycia nie są tutaj bardzo wysokie. Nie mniej jednak system ten jest często krytykowany jak “marnujący zasoby”. *Next generacja alt coinów* to próba rozwiązyania tego problemu. Dwucelowe algorytmy *proof-of-work* rozwiązują szczególny problem „użytkowy”, generując jednocześnie *proof of work* zabezpieczający sieć. Ryzyko dodania zewnętrznej możliwości wykorzystania zabezpieczenia waluty polega na tym, że wprowadza ona zewnętrzne wpływy do krzywej popytu/podaży.

Primecoin

Wprowadzenie *Primecoinów* ogłoszono w lipcu 2013r. Algorytm *proof-of-work* tego systemu wyszukuje liczby pierwsze obliczając łańcuchy Cunninghama oraz zespołów bliźniaczych liczb pierwszych. Liczby pierwsze są wykorzystywane w różnych dziedzinach nauki. łańcuch blokowy *Primecoin* zawiera zidentyfikowane liczby pierwsze i tym samym zaczyna generować rekord publicznych odkrycia naukowego równolegle do zapisu w publicznej księdze transakcji.

- Czas generowania bloku: 1 minuta
- Łączna liczba jednostek waluty: Bez limitu
- Algorytm *consensusu*: *Proof of work* z odkrywaniem łańcucha liczb pierwszych
- Kapitalizacja rynkowa: 1,3 miliona USD w połowie 2014r.

Curecoin

Wprowadzenie Curecoin ogłoszono w maju 2013r. Łączy on w sobie algorytm SHA256 *proof-of-work* z badaniem fałdowania białka w projekcie *Folding@Home*. Fałdowanie białka to intensywna obliczeniowo symulacja biochemicalnych interakcji pomiędzy białkami wykorzystywana do opracowywania nowych leków pozwalających leczyć różnego rodzaju schorzenia

- Czas generowania bloku: 10 minut
- Łączna liczba jednostek waluty: Bez limitu
- Algorytm *consensusu*: *Proof of work* w połączeniu z badaniem fałdowania białek
- Kapitalizacja rynkowa: 58 000 w połowie 2014r.

Gridcoin

System *Gridcoin* wprowadzono w październiku 2013r. Stanowi on uzupełnienie algorytmu *proof of work* opartego na skrypcie oraz posiada wsparcie (dotacje) dla uczestnictwa w obliczeniach sieci otwartej BOINC. BOINC—Berkeley Open Infrastructure for Network Computing jest to otwarty protokół stosowany w celu obliczeń przez naukową sieć badawczą, która umożliwia uczestnikom dzielenie się wolnymi cyklami obliczeniowymi i przeprowadzanie różnych obliczeń dla celów naukowo-badawczych. *Gridcoin* wykorzystuje BOINC jako platformę obliczeniową wykonującą obliczenia o charakterze ogólnym, a nie do rozwiązywania szczególnych problemów naukowych takich, jak problematyka liczb pierwszych, czy fałdowanie białka.

- Czas generowania bloku: 150 sekund
- Łączna liczba jednostek waluty: Bez limitu
- Algorytm *consensusu*: *Proof-of-work* z dostępnością sieciowych zasobów obliczeniowych BOINC
- Kapitalizacja rynkowa: 122 000 w połowie 2014

Alt coin typu anonymity-focused (zapewniające anonimowość): CryptoNote, Bytecoin, Monero, Zerocash/Zerocoin, Darkcoin

System Bitcoin jest często mylnie określany jako "waluta anonimowa". W rzeczywistości jest niezwykle łatwo połączyć tożsamości z adresami bitcoin wykorzystując analitykę dużych danych, połączyć adresy ze sobą i uzyskać ogólny obraz nawyków wydatkowych. Niektóre z *alt coinów* mają na celu rozwiązywanie tej kwestii bezpośrednio koncentrując się na zachowaniu jak największej anonimowości. Pierwszą tego typu próbą jest prawdopodobnie *Zerocoin* – jest to protokół *meta-coin* pozwalający zachować anonimowość na szczeblu systemu *bitcoin*, przedstawiony w trakcie wykładu wygłoszonego na symposium poświęconym kwestiom bezpieczeństwa i prywatności w roku 2013 pn. IEEE - Symposium on Security and Privacy. *Zerocoinsy* będą wdrażane jako całkowicie oddzielne *alt coin* pod nazwą *Zerocash* – w momencie opracowywania niniejszego materiału znajdują się w fazie opracowywania. Alternatywne podejście do kwestii anonimowości zaprezentowano wraz z systemem walutowym *CryptoNote* w dokumencie opublikowanym w październiku 2013r. *CryptoNote* to technologia bazowa, która jest wykorzystywana przez różne rozwidlenia *alt coinów* omówione w dalszej części niniejszego opracowania. W uzupełnieniu do *Zerocash* oraz *CryptoNotes* istnieje kilka innych niezależnych systemów *coinowych* takich jak np. *Darkcoin*, które wykorzystują kradzione adresy lub remiksowanie transakcji w celu zapewnienia anonimowości.

Zerocoin/Zerocash

Zerocoin to podejście teoretyczne do kwestii anonimowości walut cyfrowych wprowadzone w roku 2013 przez naukowców z Uniwersytetu Johns Hopkins. *Zerocash* to kolejna implementacja *alt-coinowej* odmiany *Zerocoin* znajdująca się w fazie opracowania jeszcze nie dopuszczona do użytkowania na rynku.

CryptoNote

Cryptonote to wzorcowa implementacja *alt coinów*, która stanowi podstawę anonimowej waluty cyfrowej (gotówka). Została wprowadzona w październiku 2013r. Jej konstrukcja zakłada możliwość tworzenia rozwidleń na różne implementacje i wyposażony jest w różnego rodzaju mechanizmy okresowego resetowania, które sprawiają, że system ten jest nieprzydatny, jako waluta jako taka. Kilka typów *alt coinów* powstało w oparciu o system *CryptoNote*, np. *Bytecoin (BCN)*, *Aeon (AEON)*, *Boolberry (BBR)*, *duckNote (DUCK)*, *Fantomcoin (FCN)*, *Monero (XMR)*, *MonetaVerde (MCN)* oraz *Quazarcoin (QCN)*. *CryptoNote* jest ważny także dlatego, że stanowi fundament kryptowaluty, a nie rozwidlenie bitcoina.

Bytecoin

Bytecoin to pierwsza pochodna *CryptoNote*, oferująca różnorodne anonimowe waluty oparte na tej technologii. *Bytecoiny* zostały wprowadzone w lipcu 2012r. Warto w tym miejscu zauważyć, że istniała wcześniejsza wersja *alt coinów* pod nazwą *Bytecoin* o symbolu BTE; kryptowaluta będąca pochodną *Bytecoin* oznaczona jest symbolem BCN. *Bytecoin* wykorzystuje algorytm *proof-of-work* pod nazwą *Cryptonight*, który wymaga dostępu do co najmniej 2 MB RAM dla każdej instancji, co sprawia, że nie nadaje się do wydobycia typu GPU lub ASIC. *Bytecoin* dziedziczy podpisy typu *ring signatures*, których nie można powiązać z innymi transakcjami oraz anonimowość odporną na analizę łańcucha blokowego z *CryptoNote*.

- Czas generowania bloku: 2 minuty
- Łączna liczba jednostek waluty: 184 miliardy BCN
- Algorytm *consensusu*: *Cryptonight proof of work*
- Kapitalizacja rynkowa: 3 miliony USD w połowie 2014r.

Monero

To kolejna postać *CryptoNote*. Charakteryzuje ją nieco spłaszczona krzywa emisji w porównaniu z *Bytecoinami*, a 80% waluty zostało wyemitowane w okresie pierwszych czterech lat. Gwarantuje ona ten sam poziom anonimowości, co system *CryptoNote*.

- Czas generowania bloku: 1 minuta
- Łączna liczba jednostek waluty: 18,4 miliona XMR
- Algorytm *consensusu*: *Cryptonight proof of work*
- Kapitalizacja rynkowa: 5 milionów USD w połowie 2014r.

Darkcoin

System *Darkcoin* zostały wprowadzone na rynek w styczniu 2014r. *Darkcoin* wprowadza walutę anonimową wykorzystującą protokół re-miksujący w odniesieniu do wszystkich transakcji nazywany *DarkSend*. System *Darkcoin* wyróżnia się także wykorzystywaniem 11 rund różnych funkcji hashujących (*blake*, *bmw*, *groestl*, *jh*, *keccak*, *skein*, *luffa*, *cubehash*, *shavite*, *simd*, czy *echo*) w celu utworzenia algorytmu *proof-of-work*.

- Czas generowania bloku: 2,5 minuty
- Łączna liczba jednostek waluty: Maksymalnie 22 miliony DRK
- Algorytm *consensusu*: *Multi-algorithm multi-round proof of work*
- Kapitalizacja rynkowa: 19 milionów USD w połowie 2014r.

Alt Chainy nie będące walutami

Alt chainy to alternatywne implementacje wzorca łańcucha blokowego, które nie są wykorzystywane jako waluta. Wiele z nich obejmuje waluty, ale są one wykorzystywane jako token pozwalający przypisanie innych elementów takich jak źródło czy kontrakt. Innymi słowy, waluta nie stanowi głównego punktu platformy; jest to cecha drugorzędna.

Namecoin

Namecoiny to pierwsze rozwidlenie kodu *bitcoinowego*. *Namecoin* stanowi zdecentralizowana platformę rejestracji wartości kluczy oraz transferu wykorzystującą łańcuch blokowy. Wspiera ona globalny rejestr nazw domen, który przypomina system rejestracji internetowych nazw domen. *Namecoin* jest obecnie wykorzystywany jako alternatywna usługa nazw domen - *domain name service (DNS)* dla domeny z rozszerzeniem *root-level .bit*. System *Namecoin* może być także wykorzystywany w celu rejestracji nazw oraz par wartości kluczy w innych nazwach: w celu przechowywania informacji takich, jak adresy poczty elektronicznej, klucze szyfrowania, certyfikaty SSL, podpisy plików, systemy głosowania, świadectwa udziałowe (giełda) oraz niezliczone ilości innych zastosowań.

System *Namecoin* składa się z waluty *Namecoin* (symbol NMC), która jest wykorzystywana w celu opłacenia transakcji oraz opłat rejestracyjnych i nazw transferowych. Obecnie, opłata za zarejestrowanie nazwy wynosi 0,01 NMC czyli w przybliżeniu 1 centa amerykańskiego. Podobnie, jak w przypadku systemu *bitcoin*, opłaty są odbierane przez górników w systemie *namecoin*.

Podstawowe parametry *Namecoinów* są takie same, jak as *bitcoinów*:

- Czas generowania bloku: 10 minut
- Łączna liczba jednostek waluty: 21 milionów NMC do 2140r.

- Algorytm *consensusu*: SHA256 proof of work
- Kapitalizacja rynkowa: \$10 milionów USD w połowie 2014r.

Nazwy domen *Namecoin* nie są ograniczone i każdy uczestnik może wykorzystać dowolną nazwę w dowolny, wybrany przez siebie sposób. Nie mniej jednak niektóre nazwy mają ogólnie przyjętą specyfikację, która sprawia, że po jej wczytaniu z łańcucha blokowego, oprogramowanie na poziomie aplikacji wie, jak ją odczytać i co następnie zrobić. W przypadku nieprawidłowej konstrukcji, każde oprogramowanie wykorzystywane do odczytu wybranych nazw wygeneruje błąd. Poniżej przedstawiamy przykłady wybranych najczęściej stosowanych nazw domen:

- d/ nazwa domen dla domen .bit
- id/ nazwa domen do przechowywania identyfikatorów osób takich, jak adresy e-mail, klucze PGP, itd.
- u/ stanowi dodatkową, charakteryzującą się lepszą strukturą specyfikację przechowywania tożsamości (na podstawie *openspecs*)

Klient *Namecoin* jest podobny do *Bitcoin Core*, ponieważ pochodzi z tego samego kodu źródłowego. W chwili instalacji, klient pobiera pełną kopię łańcucha blokowego i dopiero wtedy jest gotowy do wyszukiwania i rejestracji nazw. Zasadniczo, wykorzystuje się trzy główne polecenia:

`name_new`

Query or preregister a name

`name_firstupdate`

Register a name and make the registration public

`name_update`

Change the details or refresh a name registration

Na przykład, aby zarejestrować domenę: mastering-bitcoin.bit, stosujemy polecenie `name_new` zgodnie z poniższym:

```
$ namecoind name_new d/mastering-bitcoin
[
    "21bab5b1241c6d1a6ad70a2416b3124eb883ac38e423e5ff591d1968eb6664a",
    "a05555e0fc56c023"
]
```

Polecenie `name_new` rejestruje zgłoszenie nazwy tworząc liczbę hashującą nazwy oraz klucz losowy. Dwa strumienie zwarcane przez `name_new` to hash oraz klucz losowy (z wcześniejszego przykładu: a05555e0fc56c023), który może być wykorzystany do upublicznienia faktu rejestracji. Po zarejestrowaniu zgłoszenia w łańcuchu blokowym *Namecoin*, można go przekonwertować na rejestrację publiczną stosując polecenie `name_firstupdate` oraz podając klucz losowy:

```
$ namecoind name_firstupdate d/mastering-bitcoin a05555e0fc56c023 "{\"map":
{"www": {"ip": "1.2.3.4"} }}"
b7a2e59c0a26e5e2664948946ebeca1260985c2f616ba579e6bc7f35ec234b01
```

Przykład ten mapuje nazwę domeny `www.mastering-bitcoin.bit` na adresy IP

1.2.3.4. Zwrócony hash to identyfikator transakcji, który może być wykorzystywany w celu jej śledzenia. Można sprawdzić nazwy zarejestrowane na użytkownika stosując polecenie `name_list`:

```
$ namecoind name_list
[
    {
        "name" : "d/mastering-bitcoin",
        "value" : "{map: {www: {ip:1.2.3.4}}}",
        "address" : "NCceBXrfRUahAGrisBA1BLPWQfSrups8Geh",
        "expires_in" : 35929
    }
]
```

Rejestracje domen *Namecoin* muszą być aktualizowane co 36 000 bloków (co ok. 200 do

250 dni). Polecenie `name_update` nie łączy się z koniecznością uiszczenia jakichkolwiek opłat i dlatego odnawianie domen w systemie *Namecoin* jest darmowe. Dostawcy stron trzecich obsługują rejestracje, wznowienia automatyczne oraz aktualizację za pośrednictwem interfejsu sieciowego za niewielką opłatą. Dzięki dostawcy-stronie trzeciej można uniknąć konieczności uruchamiania klienta *Namecoin*, jednocześnie tracąc niezależną kontrolę nad zdecentralizowanym rejestrem nazw oferowanym przez *Namecoin*.

Bitmessage

Bitmessage to łańcuch *bitcoin alt*, który wykorzystuje zdecentralizowaną usługę wysyłania wiadomości zasadniczo w postaci bez serwerowego systemu poczty szyfrowanej. *Bitmessage* umożliwia użytkownikom tworzenie i wysyłanie wiadomości do siebie wzajemnie korzystając z adresów *Bitmessage*. Wiadomości działają w taki sam sposób, jak transakcje *bitcoin*, ale mają charakter przejściowy—nie są przechowywane przez okres dłuższy niż dwa dni i jeżeli w tym czasie nie zostaną dostarczone do węzła docelowego, są usuwane. Nadawcy i odbiorcy są anonimowi—nie posiadają innych identyfikatorów niż adresy *bitmessage*, ale charakteryzują się „silnym” uwierzytelnieniem, co oznacza, że wiadomości nie mogą być „sfałszowane”. Wiadomości *Bitmessage* są szyfrowane dla odbiorcy i dlatego sieć *Bitmessage* jest odporna na całościowy nadzór—pod słuchający musi włamać się do urządzenia odbiorcy, aby przejąć wiadomość.

Ethereum

Ethereum to platforma do przetwarzania i zawierania kompletnych kontraktów Turinga (ang. *Turing complete contract processing*) oparta na księdze łańcucha blokowego. Nie jest to klon *Bitcoin*a, ale całkowicie niezależna konstrukcja i implementacja. *Ethereum* posiada wbudowaną walutę nazywaną *ether*, która jest niezbędna w celu opłacenia realizacji kontraktu. łańcuch blokowy *Ethereum* rejestruje kontrakty, które są wyrażone w szczególnym języku, przypominającym kodowanie bajtami typu *Turing-complete*. Zasadniczo, kontrakt jest to program działający w każdym węźle systemu *Ethereum*. Kontrakty *Ethereum* mogą przechowywać dane wysyłać i odbierać płatności w walucie *ether*, przechowywać jednostki jej jednostki oraz wykonywać nieskończoną liczbę (dlatego *Turing-complete*) czynności obliczeniowych pełniących rolę zdecentralizowanych, autonomicznych agentów oprogramowania.

Ethereum może wdrażać bardzo złożone systemy, które niezależnie od niego mogą być wdrażane jako *alt chainy*. Na przykład, poniżej przedstawiamy kontrakt przypominający rejestracje nazwy *Namecoin* zapisany w *Ethereum* (lub nieco bardziej precyzyjnie – napisany w ogólnym języku, który można skompilować w kodzie *Ethereum*):

```
if !contract.storage[msg.data[0]]: # Is the key not yet taken?  
    # Then take it!  
    contract.storage[msg.data[0]] = msg.data[1]  
    return(1)  
else:  
    return(0) // Otherwise do nothing
```

Przyszłość kryptowalut

Przyszłość kryptowalut rysuje się w jeszcze jaśniejszym świetle niż przyszłość *bitcoinów*. System *Bitcoin* wprowadził całkowicie nową formę zdecentralizowanej organizacji oraz *consensus*, który zapoczątkował setki nieprawdopodobnych nowoczesnych rozwiązań. Unowocześnienia te będą mieć prawdopodobnie wpływ na liczne sektory gospodarki – od nauk dotyczących systemów rozproszonych poprzez finanse, ekonomię, systemy walutowe, bankowość centralną, aż po ład korporacyjny. Wiele działań podejmowanych przez człowieka, które

w przeszłości wymagały utworzenia zcentralizowanych instytucji lub organizacji, które pełniły rolę władz lub organizacji zaufania publicznego sprawujących kontrolę można obecnie całkowicie zdecentralizować. Opracowanie łańcucha blokowego oraz systemu *consensus* przyczyni się do znacznego obniżenia kosztów organizacji i koordynacji systemów działających na szeroką skalę jednocześnie eliminując konieczność koncentracji władzy, działań korupcyjnych oraz tworzenia regulacji i kontroli.

ROZDZIAŁ 10

Bezpieczeństwo systemu Bitcoin

Zabezpieczenie *bitcoinów* to spore wyzwanie, ponieważ nie są one jedynie abstrakcyjnym odniesieniem do wartości jak ma to miejsce np. w przypadku salda na koncie bankowym. System *bitcoin* to cyfrowa gotówka lub złoto. Zapewne spotkaliście się z tym zivotem, "Własność prywatna stanowi dziewięć z dziesięciu części prawa." Cóż, w przypadku *bitcoinów*, posiadanie to 100% prawa. Posiadanie kluczy do odblokowywania *bitcoinów* stanowi równoważnik posiadania gotówki lub kawałka metalu szlachetnego. Można je stracić, położyć w nieodpowiednim miejscu, ktoś może nam je ukraść lub przypadkowo możemy komuś dać niewłaściwą kwotę. W każdym z wymienionych przypadków, użytkownicy nie mają żadnej możliwości regresu, podobnie jak w przypadku zgubienia pieniędzy na ulicy.

Nie mniej jednak system *bitcoin* otwiera możliwości, których nie daje gotówka, złoto czy rachunek bankowy. Portfel *bitcoin* zawierający Wasze klucze można zapisywać i przechowywać zupełnie jak zwykły plik. Można przechowywać różne jego kopie, także na wydruku w celu posiadania kopii fizycznej – tych wszystkich czynności nie można przeprowadzać („tworzyć kopii zapasowych”) w przypadku gotówki, złota i rachunków bankowych. *Bitcoin* jest tak odmiennym systemem od wszystkich dotychczasowych systemów, że kwestia jego zabezpieczeń wymaga całkowicie nowatorskiego podejścia.

Zasady bezpieczeństwa

Podstawową zasadą systemu *bitcoin* jest decentralizacja, która ma istotne znaczenie i skutki dla systemu zabezpieczeń. Zcentralizowany model taki, jak tradycyjny bank lub sieć płatnicza, jest zależny od kontroli dostępu oraz starannego procesu weryfikacji w celu wykluczenia nieodpowiednich uczestników z systemu. Dla porównania, zdecentralizowany system taki, jak *bitcoin* przenosi odpowiedzialność na użytkowników. Ponieważ bezpieczeństwo sieci oparte jest na algorytmie *proof of work*, a nie kontroli dostępu, sieć może funkcjonować w systemie otwartym bez konieczności szyfrowania traffiku *bitcoin*.

Tradycyjne sieci płatniczej takie jak np. system kart kredytowych są otwartymi systemami, ponieważ zawierają prywatne identyfikatory użytkowników (numer karty kredytowej). Po uiszczeniu początkowej opłaty, każdy uczestnik posiadający dostęp do identyfikatora może „pobrać” środki i obciążać właściciela wiele razy. W związku z tym, sieć płatnicza musi posiadać zabezpieczenie typu end-to-end, możliwość szyfrowania, a także zapewniać, że żaden podsłuchujący, czy pośrednik nie naruszy strumienia płatności w trakcie przesyłania, czy też zapisanych (w stanie spoczynku). W sytuacji, gdy niepowołany uczestnik uzyskuje dostęp do systemu, może on naruszyć warunki bieżącej transakcji oraz tokenów płatności i wykorzystać je do utworzenia nowych transakcji. Co gorsza, uzyskanie dostępu do danych klienta powoduje narażenie tego ostatniego na ryzyko kradzieży tożsamości i w związku z tym należy podjąć działania zmierzające do zapobiegania działaniom przestępczym na kontach do których uzyskano nieuprawniony dostęp.

Bitcoin to system o całkowicie odmiennym charakterze. Transakcja *bitcoinowa* autoryzuje jedynie określone wartości dla wybranych odbiorców i nie może być sfałszowana lub zmieniona. Nie ujawnia ona żadnych poufnych (prywatnych) informacji takich jak tożsamość stron i nie może być wykorzystywana w celu autoryzacji dodatkowych płatności. W związku z tym sieć płatnicza *bitcoin* nie wymaga szyfrowania ani ochrony przed podsłuchiwaniem. W zasadzie, istnieje możliwość przesyłania transakcji *bitcoinowych* za pośrednictwem otwartego kanału publicznego takiego jak np. niezabezpieczona sieć WiFi lub Bluetooth bez uszczerbku dla bezpieczeństwa transakcji.

Zdecentralizowany model bezpieczeństwa *bitcoin* oferuje szerokie możliwości swoim uczestnikom, ale wiążą się one z koniecznością i odpowiedzialnością za zachowanie tajemnicy kluczy. W przypadku większości użytkowników warunek ten nie jest łatwy do spełnienia zwłaszcza w przypadku urządzeń obliczeniowych ogólnego przeznaczenia takich, jak połączone ze sobą smartfony czy laptopy. Chociaż zdecentralizowany model *bitcoínów* uniemożliwia masowe złamanie zabezpieczeń, jak to obserwowano w przypadku kart kredytowych, wielu użytkowników nie jest w stanie adekwatnie zabezpieczyć swoich kluczy i w efekcie doświadczają kolejno licznych kradzieży danych i włamań hackerskich.

Bezpieczny rozwój systemów bitcoinowych

Najważniejszą zasadą, którą muszą kierować się twórcy systemu *bitcoin* jest decentralizacja. Większość autorów znane są zcentralizowane modele bezpieczeństwa i mogą ulegać pokusie wykorzystania ich w aplikacjach *bitcoinowych* z katastrofalnymi skutkami.

Bezpieczeństwo systemu zasadza się na zdecentralizowanej kontroli nad kluczami oraz na niezależnej walidacji transakcji przez górników. W przypadku konieczności zastosowania dźwigni bezpieczeństwa systemu *Bitcoin*, należy zapewnić jej zgodność z modelem bezpieczeństwa *Bitcoin*. Mówiąc prościej: nie należy pozbawiać uczestników kontroli nad kluczami ani usuwać transakcji z łańcuchów blokowych.

Na przykład, wiele wczesnych transakcji wymiany polegało na koncentracji wszystkich funduszy użytkowników w jednym "gorącym" portfelu, a klucze były przechowywane na jednym serwerze. Takie rozwiązanie pozbawia użytkowników możliwości zarządzania i skupia kontrolę nad kluczami w jednym systemie narażając go na włamania i tragiczne konsekwencje dla jego użytkowników.

Kolejnym powszechnie występującym błędem jest „usuwanie” transakcji z łańcucha blokowego w błędny przekonaniu, że pozwoli to obniżyć koszty transakcji i przyśpieszyć proces ich przetwarzania. System “poza łańcuchem blokowym” będzie rejestrować transakcje wewnętrznej, zcentralizowanej księdze i jedynie czasami synchronizować je z łańcuchem blokowym *bitcoinów*. I znowu, praktyka ta stanowi substytut dla zdecentralizowanych zabezpieczeń *bitcoínów* poprzez stosowanie autorskiego i zcentralizowanego podejścia. Umieszczanie transakcji poza łańcuchem blokowym zwiększa zagrożenie sfałszowania niewłaściwie zabezpieczonych ksiąg centralnych, wyprowadzania środków oraz zmniejszania się rezerw w niezauważalny sposób.

Należy dobrze rozważyć wszelkie opcje zanim usuniecie środki ze zdecentralizowanego systemu zabezpieczeń *Bitcoin*, chyba, że macie możliwość zainwestowania w zabezpieczenia systemu, różne poziomy kontroli dostępu oraz systemy audytowe (jak to ma miejsce w przypadku tradycyjnych banków). Nawet, jeśli dysponujecie środkami oraz posiadacie odpowiednią dyscyplinę do wdrażania silnych modeli bezpieczeństwa, pamiętajcie, że taka konstrukcja stanowi jedynie replikę kruchego modelu tradycyjnych sieci finansowych, w których plagą jest kradzież tożsamości, łamanie zabezpieczeń, czy defraudacja środków. Aby wykorzystać unikalny zdecentralizowany model zabezpieczeń *Bitcoin*, należy unikać pokusy stworzenia zcentralizowanej architektury, która może wydawać się dobrze znana, ale w ostatecznym efekcie przyczynić się do osłabienia poziomu bezpieczeństwa systemu *Bitcoin*.

Root of Trust

Tradycyjna architektura bezpieczeństwa jest oparta na koncepcji tzw. *root of trust*, która stanowi zaufany rdzeń leżący u podstaw bezpieczeństwa systemu ogólnego lub aplikacji. Architektura bezpieczeństwa jest budowana wokół *root of trust* jako seria okręgów rozchodzących się koncentrycznie, przypominających kolejne warstwy cebuli i rozchodzących się na zewnątrz od środka. Każda warstwa powstaje na posiadającej większe zaufanie warstwie wewnętrznej wykorzystując funkcje kontroli dostępu, podpisy cyfrowe, szyfrowanie oraz inne

prymitywne metody zabezpieczeń. Wraz z coraz większym stopniem złożoności systemów, rośnie prawdopodobieństwo wystąpienia w nich błędów, co będzie sprawiać, że staną się one podatne na taki ukierunkowane na złamanie zabezpieczeń. W efekcie, im bardziej złożony staje się system, tym jest trudniejszy do zabezpieczenia. Koncepcja *root of trust* gwarantuje, że większość zaufania koncentruje się w najmniej złożonej części systemu, która jednocześnie jest najmniej podatna na ataki, a złożone oprogramowanie jest budowane warstwowo wokół tego elementu systemu. Wymieniona architektura bezpieczeństwa jest powielana w różnej skali; pierwszym elementem jest stworzenie *root of trust* w urządzeniach pojedynczego systemu, a następnie jego rozszerzenie za pośrednictwem systemu operacyjnego na kolejne, wyższe usługi systemowe i ostatecznie na różne serwery rozproszone warstwowo w układzie koncentrycznym o zmniejszającym się stopniu zaufania (bezpieczeństwa).

Architektura zabezpieczeń systemu *bitcoin* ma odmienny charakter. W Bitcoin, system *consensusu* pozwala zbudować zaufaną księgę publiczną, która jest całkowicie zdecentralizowana. Odpowiednio zweryfikowane (walidacja) łańcuchy blokowe wykorzystują blok *genesis* jako *root of trust* w procesie budowy łańcucha zaufania, aż do bieżącego bloku. Systemy *bitcoinowe* mogą i powinny wykorzystywać łańcuchy blokowe, jako swoje *root of trust*. Projektując złożoną aplikację *bitcoin* składającą się z usług dostępnych w różnych systemach, należy dokładnie zapoznać się z architekturą bezpieczeństwa i ustalić miejsce, w którym koncentruje się poziom zaufania. Ostatecznie, jedynym elementem, któremu można zaufać jest poddany pełnej walidacji łańcuch blokowy. Jeżeli Wasza aplikacja w sposób wyraźny lub pośrednio powierza zaufanie elementowi innemu, niż łańcuch blokowy, powinno to stanowić powód do niepokoju, ponieważ stanowi to osłabienie odporności systemu. Dobrym sposobem oceny bezpieczeństwa architektury Waszej aplikacji jest ocean poszczególnych komponentów oraz hipotetycznego scenariusza zakładającego całkowite złamanie zabezpieczenia danego komponentu oraz i przejęcie kontroli przez złośliwego aktora. Należy przeprowadzić analizę każdego elementu aplikacji i ocenić jego wpływ jego naruszenia na ogólny poziom bezpieczeństwa. Jeżeli Wasza aplikacja nie jest zabezpieczona, w przypadku naruszenia jednego z komponentów oznacza to, że zaufanie zostało ulokowane w nieodpowiednim miejscu. Aplikacja *bitcoin* bez słabych elementów może być podatna jedynie na naruszenie przez mechanizmu *consensusu bitcoin*, co oznacza, że jego *root of trust* jest oparty na najsilniejszej części architektury bezpieczeństwa systemu *bitcoin*.

Liczne przykłady naruszenia systemu wymiany *bitcoinów* wydają się podważać tę tezę, ponieważ ich architektura bezpieczeństwa oraz konstrukcja nie sprawdzają się nawet w przypadku najbardziej ogólnej kontroli.

Wymienione zcentralizowane implementacje ulokowały zaufanie w licznych komponentach usytuowanych poza łańcuchami blokowymi *bitcoinów* takimi, jak portfele, zcentralizowane bazy ksiąg, podatne klucze szyfrowania i inne, podobne systemy.

Najlepsze praktyki użytkowników w zakresie bezpieczeństwa systemu

Od tysięcy lat ludzie stosują różnego rodzaju systemy zabezpieczeń fizycznych. Dla porównania, nasze doświadczenie w dziedzinie zabezpieczeń cyfrowych liczy sobie zaledwie 50 lat. Współczesne systemy operacyjne ogólnego przeznaczenia nie są zbyt bezpieczne ani szczególnie dobrze dostosowane dla celów przechowywania walut cyfrowych. Nasze systemy są stale narażane na zagrożenia zewnętrzne w związku ze stałym połączeniem z Internetem. Działają na nich nieustannie tysiące elementów oprogramowania opracowywanych przez tysiące autorów, często dysponujących nieograniczonym dostępem do plików użytkowników. Jedno złośliwe oprogramowanie pośród wielu tysięcy innych zainstalowanych na Waszym komputerze może wyłączyć klawiaturę, naruszyć zawartość plików, czy też skróści *bitcoiny* przechowywane w

aplikacjach portfelowych. Poziom opieki technicznej wymagany w celu zabezpieczenia jednostki przed wirusami i trojanami pozostaje poza zakresem umiejętności wszystkich za wyjątkiem niewielkiej grupy użytkowników komputerów.

Pomimo dziesięcioleci badań oraz postępów w dziedzinie zabezpieczeń informatycznych, zasoby cyfrowe są w dalszym ciągu zdecydowanie podatne na działania zdeterminowanych atakujących. Nawet najbardziej chronione oraz zabezpieczone systemy w organizacjach finansowych, agencjach wywiadowczych, czy jednostkach związanych z obronnością są często łamane. System *bitcoin* tworzy zasoby cyfrowe posiadające odpowiednią, związaną wartość i które mogą być obiektem kradzieży, bądź przekierowane na konta nowych właścicieli w sposób szybki i nieodwracalny. Stanowi to ogromną motywację do działania dla hackerów. Do chwili obecnej atakujący musieli dokonać konwersji danych tożsamości lub tokenów rachunków w postaci kart kredytowych czy rachunków bankowych na odpowiednie wartości po złamaniu zabezpieczeń. Pomimo trudności wynikających z konieczności ochrony i oczyszczania informacji finansowych obserwujemy rosnącą liczbę kradzieży. System *bitcoin* radzi sobie z tym problemem, ponieważ nie wymaga budowania ochrony czy oczyszczania danych; reprezentuje on wewnętrzną wartość przypisaną do aktywu cyfrowego.

Na szczeźle *bitcoin* także stanowi zachętę do podnoszenia bezpieczeństwa komputerów. Choć wcześniej ryzyko złamania zabezpieczeń komputerów było mgliste i nieoczywiste, *bitcoin* czyni te zagrożenia oczywistymi i wyraźnymi. Posiadanie *bitcoinów* w zasobach komputerowych ma na celu skupienie uwagi użytkownika na konieczności podnoszenia bezpieczeństwa zasobów informatycznych. Bezpośrednim skutkiem powielania i rosnącego wykorzystania *bitcoinów* jest rozwój zarówno technik hackerskich jak i postępy w dziedzinie zabezpieczeń. Mówiąc prosto – obecnie hackerzy mają przed sobą bardzo apetyczny kąsek, a użytkownicy jasno określony cel, jakim jest ochrona samych siebie.

Na przestrzenie ostatnich trzech lat, jako bezpośredni skutek stosowania *bitcoinów*, mieliśmy okazję zaobserwować ogromny postęp w dziedzinie zabezpieczeń informatycznych przejawiających się w postaci opcji szyfrowania urządzeń, przechowywania kluczy oraz portfeli sprzętowych, technologii *multi-signature*, czy cyfrowych systemów *escrow*. W kolejnych częściach omówimy najlepsze praktyki w zakresie praktycznego zapewnienia bezpieczeństwa użytkowników.

Fizyczne przechowywanie Bitcoinów

Ponieważ większość uczestników odczuwa większy komfort posiadając zabezpieczenia fizyczne niż informatyczne, bardzo skuteczną metodą ochrony *bitcoinów* jest ich zamiana na postać fizyczną. Klucze *bitcoinowe* to nic innego jak długie ciągi liczbowe. Oznacza to, że mogą być przechowywane w postaci fizycznej, np. wydruku lub jako informacja wytrawiona na monecie z metalu. Zatem zabezpieczenie kluczy staje się równie proste jak zabezpieczenie fizyczne wydruku klucza *bitcoinowego*. Wydruk zestawu kluczy *bitcoinowych* nazywamy “portfelem papierowym”; istnieje wiele różnych darmowych narzędzi, za pomocą których możemy je utworzyć. Ja osobiście przechowuję wiele własnych *bitcoinów* (99% albo więcej) w postaci portfeli papierowych szyfrowanych za pomocą BIP0038, których różne kopie przechowuję w zabezpieczonych sejfach. Przechowywanie *bitcoinów* w trybie offline nazywamy *cold storage* i jest to jeden z najskuteczniejszych sposobów bezpiecznego przechowywania cennych rzeczy. System *cold storage* polega na generowaniu kluczy w systemie offline (który nigdy nie jest połączany do Internetu) i przechowywaniu ich w trybie offline w postaci wydruku lub na nośnikach cyfrowych takich, jak urządzenia na USB.

Portfele sprzętowe

W dłuższej perspektywie czasowej, zabezpieczenia system będą coraz częściej przybierać postać urządzeń portfelowych zabezpieczonych przed włamaniem. W przeciwieństwie do smartfonów czy komputerów

stacjonarnych, urządzenia portfelowe *bitcoin* mają jedno zadanie do spełnienia: bezpieczne przechowywanie *bitcoinów*. Bez oprogramowania ogólnego przeznaczenia, które można obejść oraz mając do dyspozycji ograniczoną liczbę interfejsów, portfele hardware'owe mogą zapewnić niemal całkowite zabezpieczenie przed przypadkowymi działaniami niedoświadczonych użytkowników. Spodziewam się, że portfele hardware'owe staną się główną metodą przechowywania *bitcoinów*. System *Trezor* może stanowić tutaj dobry przykład takiego rozwiązania.

Wyważenie ryzyka

Chociaż większość użytkowników słusznie odczuwa niepokój związany z możliwością kradzieży *bitcoinów*, istnieje jeszcze większe ryzyko. Pliki danych bardzo często „giną”. Jeżeli zawierają jednocześnie *bitcoiny*, strata staje się jeszcze bardziej bolesna. Podejmując działania na rzecz zabezpieczenia swoich portfeli *bitcoinowych*, użytkownicy muszą pamiętać o tym, aby nie posuwać się zbyt daleko i ostatecznie nie stracić cennych *bitcoinów*. Latem 2010r., znany wszystkim projekt popularyzujący, którego jednym z zadań było prowadzenie szkoleń w tym zakresie odnotował stratę niemal 7 000 *bitcoinów*. Podejmując działania na rzecz zapobiegania kradzieży, właściciele wdrożyli złożony ciąg szyfrowanych backupów. W efekcie, przypadkowo zagubili klucze szyfrowania sprawiając, że wszystkie wcześniejsze działania stały się bezużyteczne i jednocześnie tracąc majątek. Niczym zagrzebywanie skarbu w piasku, zbyt mocne zabezpieczenia mogą spowodować, że nie będziemy w stanie wydobyć naszych precjozów ze schowka.

Dyweryfikacja ryzyka

Czy wolelibyście nosić swoje zasoby sieciowe w gotówce, w portfelu? Większość osób uznałaby takie zachowanie za nierozsądne, ale użytkownicy *bitcoinów* często trzymają całą walutę w jednym portfelu. Zamiast tego, użytkownicy powinny rozpraszać ryzyko umieszczając je w różnych, całkowicie różnych od siebie portfelach *bitcoinowych*. Przezorni użytkownicy będą przechowywać jedynie ułamek – zapewnie stanowiący mniej niż 5% - swoich *bitcoinów* w systemie dostępnym online lub portfelu mobilnym, jako “drobne w kieszeni”. Reszta powinna być podzielona na kilka różnych mechanizmów przechowywania takich, jak np. portfel w komputerze stacjonarnym czy w systemie offline (*cold storage*).

Systemy multi-sig i ład korporacyjny

Kiedy korporacja lub osoby fizyczna przechowuje dużą liczbę *bitcoinów*, powinna pamiętać o konieczności stosowania adresów *bitcoinowych* typu *multi-signature*. Adresy *multi-signature* zabezpieczają środki poprzez wymóg złożenia więcej niż jednego podpisu w celu dokonania płatności. Klucze podpisu powinny być przechowywane w różnych (odmiennych) lokalizacjach oraz zarządzane przez różne osoby. W środowisku korporacyjnym, na przykład, klucze powinny być generowane niezależnie i przechowywane przez różnych członków kierownictwa, aby wyeliminować ryzyko, że jedna osoba sprzeniewierzy środki firmowe. Adresy typu *multi-signature* mogą także wiązać się z redundancją w sytuacji, gdy jedna osoba posiada kilka kluczy, które są przechowywane w różnych lokalizacjach.

Survivability – przeżywalność

Jednym z istotniejszych względów bezpieczeństwa, który jest często pomijany jest dostępność, szczególnie w kontekście niesprawności lub śmierci posiadacza klucza. Użytkownicy *bitcoinów* są informowani o konieczności stosowania złożonych haseł oraz przechowywania kluczy w bezpiecznych i niedostępnych miejscach oraz nieujawniania ich nikomu. Niestety, praktyka ta sprawia, że jest niemal niemożliwe dla rodziny użytkownika

odzyskanie środków, w przypadku jego nieobecności. W większości przypadków rodziny użytkowników *bitcoinów* mogą zupełnie nie mieć świadomości o istnieniu funduszy *bitcoinowych*.

Jeżeli posiadacie dużo *bitcoinów*, powinniście rozważyć możliwość powierzenia danych dostępowych zaufanej osobie lub pełnomocnikowi. Nieco bardziej skomplikowany system dostępności można zbudować w oparciu o dostęp typu *multisignature* oraz planowanie nieruchomości za pośrednictwem prawnika specjalizującego się w "zabezpieczaniu/egzekucji zasobów cyfrowych".

Wnioski

Bitcoiny to całkowicie nowa, bezprecedensowa i złożona technologia. Z biegiem czasu będziemy opracowywać coraz to lepsze narzędzia zabezpieczeń oraz praktyki łatwiejsze w stosowaniu przez niedoświadczonych użytkowników. W chwili obecnej, użytkownicy *bitcoinów* mogą wykorzystywać wiele wskazówek zawartych w treści niniejszego opracowania oraz cieszyć się bezpiecznym i nieskrepowanym korzystaniem z możliwości oferowanych przez system *bitcoin*.

ZAŁĄCZNIK A

Operatory języków skryptów transakcyjnych, wartości stałych i symboli

Tabela A-1 zawiera informacje o operatorach wprowadzających wartości do stosu.

Tabela A-1. Wprowadzanie wartości do stosu (Push value onto stack)

Symbol	Wartość (szesnastkowa)	Opis
OP_0 or OP_FALSE	0x00	Pusta macierz jest wprowadzana do stosu
1-75	0x01-0x4b	Wprowadź kolejnych N bajtów do stosu; N oznacza 1 do 75 bajtów
OP_PUSHDATA1	0x4c	Kolejny bajt skryptu zawiera N, wprowadź kolejne N bajtów do stosu
OP_PUSHDATA2	0x4d	Kolejne dwa bajty skryptów zawierają N, wprowadź kolejne N bajtów do stosu
OP_PUSHDATA4	0x4e	Kolejne cztery bajty skryptów zawierają N, wprowadź kolejne N bajtów do stosu
OP_1NEGATE	0x4f	Wprowadź wartość “-1” do stosu
OP_RESERVED	0x50	Stop (Halt) – transakcja nieważna, chyba, że znaleziona w niewykonanym zdaniu OP_IF
OP_1 or OP_TRUE	0x51	Wprowadź wartość “-1” do stosu
OP_2 to OP_16	0x52 to 0x60	Dla OP_N, wprowadź wartość “N” do stosu. np., OP_2 wprowadza “2”

Tabela A-2 przedstawia warunkowych operatorów kontroli przepływu.

Tabela A-2. Kontrola warunkowego przepływu

Symbol	Wartość (szesnastkowa)	Opis
OP_NOP	0x61	Bez działania
OP_VER	0x62	Stop (Halt) – transakcja nieważna, chyba, że znaleziona w niewykonanym zdaniu OP_IF
OP_IF	0x63	Wykonaj zdania, jeżeli góra stosu nie jest równa 0
OP_NOTIF	0x64	Wykonaj zdania, jeżeli góra stosu jest równa 0
OP_VERIFY	0x65	Stop (Halt) – transakcja nieważna
OP_VERNOTIF	0x66	Stop (Halt) – transakcja nieważna
OP_ELSE	0x67	Wykonaj wyłącznie, gdy poprzednie zdania nie zostały wykonane
OP_ENDIF	0x68	Zakończ blok OP_IF, OP_NOTIF, OP_ELSE
OP_VERIFY	0x69	Sprawdź szczyt stosu, zatrzymaj i unieważnij transakcję, jeżeli nie ma wartości TRUE (prawda)
OP_RETURN	0x6a	Zatrzymaj i unieważnij transakcję

Tabela A-3 pokazuje polecenia/operatory wykorzystywane do zmiany stosu.

Tabela A-3. Operacje w stosie

Symbol	Wartość (szesnastkowa)	Opis
--------	------------------------	------

OP_TOALTSTACK	0x6b	Wyrzuć szczytowy element ze stosu i przenieś na inny stos
OP_FROMALTST ACK	0x6c	Wyrzuć szczytowy element ze stosu alternatywnego przenieś na stos
OP_2DROP	0x6d	Wyrzuć dwa szczytowe elementy stosu
OP_2DUP	0x6e	Zduplikuj dwa szczytowe elementy stosu
OP_3DUP	0x6f	Zduplikuj trzy szczytowe elementy stosu
OP_2OVER	0x70	Skopiuj trzeci i czwarty element w stosie na szczyt
OP_2ROT	0x71	Przenieś piąty i szósty element w stosie na szczyt
OP_2SWAP	0x72	Zamień dwie szczytowe pary elementów w stosie
OP_IFDUP	0x73	Zduplikuj szczytowy element w stosie, jeżeli ma wartość różną od 0
OP_DEPTH	0x74	Zlicz elementy w stosie i wprowadź wynik
OP_DROP	0x75	Wyrzuć szczytowy element w stosie
OP_DUP	0x76	Zduplikuj szczytowy element w stosie
OP_NIP	0x77	Wyrzuć drugi element w stosie
OP_OVER	0x78	Skopiuj drugi element w stosie i wypchnij na szczyt
OP_PICK	0x79	Wypchnij wartość N ze szczytu, następnie skopiuj N-ty element na szczyt stosu
OP_ROLL	0x7a	Wypchnij wartość N ze szczytu, następnie skopiuj N-ty element na szczyt stosu
OP_ROT	0x7b	Obróć trzy górne elementy w stosie
OP_SWAP	0x7c	Zamień trzy górne elementy w stosie
OP_TUCK	0x7d	Skopiuj szczytowy element i wprowadź go pomiędzy szczytowy i drugi element

Tabela A-4 zawiera operatory strumieni.

Tabela A-4. Operacje splotu strumienia (String splice operations)

Symbol	Wartość (szesnastkowa)	Opis
OP_CAT	0x7e	Wyłączone (łączy dwa szczytowe elementy)
OP_SUBSTR	0x7f	Wyłączone (zwraca pod-strumień)
OP_LEFT	0x80	Wyłączone (zwraca lewy pod-strumień)
OP_RIGHT	0x81	Wyłączone (zwraca prawy pod-strumień)
OP_SIZE	0x82	Oblicz długość szczytowego elementu i wprowadź wynik

Tabela A-5 pokazuje operatory logiczne i arytmetyczne.

Tabela A-5. Arytmetyka binarna i warunki

Symbol	Wartość (szesnastkowa)	Opis
OP_INVERT	0x83	Wyłączone (obróć bit szczytowego elementu)
OP_AND	0x84	Wyłączone (logiczne ORAZ dwa szczytowe elementy)
OP_OR	0x85	Wyłączone (logiczne LUB dwa szczytowe elementy)
OP_XOR	0x86	Wyłączone (logiczne XOR dwóch szczytowych elementów)
OP_EQUAL	0x87	Wprowadź TRUE (1), jeżeli dwa szczytowe elementy są równe, wprowadź FALSE (0) w innych sytuacjach
OP_EQUALVERIFY	0x88	Podobnie jak OP_EQUAL, ale należy uruchomić OP_VERIFY potem, aby zatrzymać, jeżeli nie ma wartości TRUE
OP_RESERVED1	0x89	Stop (Halt) - nieważna transakcja, chyba że znaleziona w niewykonanym zdaniu OP_IF
OP_RESERVED2	0x8a	Stop (Halt) - nieważna transakcja, chyba że znaleziona w niewykonanym zdaniu OP_IF

Tabela A-6 przedstawia operatory numeryczne (arytmetyczne).

Tabela A-6. Operatory arytmetyczne

Symbol	Wartość (szesnastkowa)	Opis
OP_1ADD	0x8b	Dodaj 1 do elementu szczytowego
OP_1SUB	0x8c	Odejmij 1 z elementu szczytowego
OP_2MUL	0x8d	Wyłączony (pomóż szczytowy element przez 2)
OP_2DIV	0x8e	Wyłączony (podziel szczytowy element przez 2)
OP_NEGATE	0x8f	Odwróć znak szczytowego elementu
OP_ABS	0x90	Zmień znak szczytowego element na dodatni
OP_NOT	0x91	Jeżeli szczytowy element wynosi 0 lub 1 logiczną odwrócić go, w przeciwnym razie przywrócić 0
OP_NOTEQUAL	0x92	Jeżeli szczytowy element wynosi 0 przywrócić 0, w przeciwnym razie przywrócić 1
OP_ADD	0x93	Wypchnij dwa szczytowe elementy, zsumuj a następnie wypchnij wynik
OP_SUB	0x94	Wypchnij dwa szczytowe elementy, odejmij pierwszy od drugiego, wypchnij wynik
OP_MUL	0x95	Wyłączony (pomnożyć dwa szczytowe elementy)
OP_DIV	0x96	Wyłączony (podzielić drugi element przez pierwszy)
OP_MOD	0x97	Wyłączony (pozostałe podzielić drugi element przez pierwszy)
OP_LSHIFT	0x98	Wyłączony (przesunąć drugi element w lewo o liczbę bitów równą wartości pierwszego elementu)
OP_RSHIFT	0x99	Wyłączony (przesunąć drugi element w prawo o liczbę bitów równą wartości pierwszego elementu)
OP_BOOLAND	0x9a	Boolean - logiczne ORAZ dwa szczytowe elementy
OP_BOOLOR	0x9b	Boolean - logiczne LUB dwa szczytowe elementy
OP_NUMEQUAL	0x9c	Powrót TRUE, jeżeli wartości dwóch szczytowych elementów są równe
OP_NUMEQUALVERIFY	0x9d	Takie same jak NUMEQUAL, następnie OP_VERIFY aby zatrzymać, jeżeli nie ma wartości TRUE
OP_NUMNOTEQUAL	0x9e	Przywrócić TRUE, jeżeli dwa elementy nie mają równych wartości liczbowych
OP_LESSTHAN	0x9f	Przywrócić TRUE, jeżeli drugi element ma wartość mniejszą niż element szczytowy
OP_GREATERTHAN	0xa0	Przywrócić TRUE, jeżeli wartość drugiego elementu jest większa, niż szczytowego elementu
OP_LESSTHANOREQUAL	0xa1	Przywrócić TRUE jeżeli wartość drugiego elementu jest mniejsza niż lub równa wartości elementu szczytowego
OP_GREATERTHANOREQUAL	0xa2	Przywrócić TRUE, jeżeli wartość drugiego elementu jest większa lub równa wartości szczytowego elementu
OP_MIN	0xa3	Przywrócić mniejszy z dwóch szczytowych elementów
OP_MAX	0xa4	Przywrócić większą z dwóch wartości szczytowych elementów
OP_WITHIN	0xa5	Przywrócić TRUE, jeżeli wartość trzeciego elementu znajduje się pomiędzy wartością drugiego (lub jest równa) i pierwszego elementu

Tabela A-7 przedstawia operatory funkcji kryptograficznych.

Tabela A-7. Operacje kryptograficzne i hashujące

Symbol	Wartość (szesnastkowa)	Opis
OP_RIPEMD160	0xa6	Przywrócić wartość hashującą RIPEMD160 szczytowego elementu
OP_SHA1	0xa7	Przywrócić wartość hashującą SHA1 szczytowego elementu
OP_SHA256	0xa8	Przywrócić wartość hashującą SHA256 szczytowego elementu
OP_HASH160	0xa9	Przywrócić wartość hashującą RIPEMD160(SHA256(x))

		szczytowego elementu
OP_HASH256	0xaa	Przywróć wartość hashującą SHA256(SHA256(x)) szczytowego elementu
OP_CODESEPARATOR	0xab	Zaznacz początek danych weryfikowanych podpisem
OP_CHECKSIG	0xac	Wypchnij klucz publiczny i podpis oraz dokonaj walidacji podpisu dla transakcji z wartością hashującą danych, przywróć TRUE, jeżeli pasuje
OP_CHECKSIGVERIFY	0xad	Tak samo jak CHECKSIG, następnie OP_VERIFY, żeby zatrzymać, jeżeli nie ma wartości TRUE
OP_CHECKMULTISIG	0xae	Uruchomić CHECKSIG dla każdej pary podpisów i kluczy publicznych. Wszystkie wartości muszą pasować. Błąd we wstawkach implementacyjnych wyrzuca dodatkową wartość, prefiks z OP_NOP w charakterze obejścia
OP_CHECKMULTISIGVERIFY	0xaf	Tak samo jak CHECKMULTISIG, następnie OP_VERIFY, aby zatrzymać, jeżeli wartość nie jest TRUE

Tabela A-8 przedstawia symbole nie będące operatorami

Tabela A-8. Symbole nie będące operatorami

Symbol	Wartość (szesnastkowa)	Opis
OP_NOP1	0xb0-0xb9	Nic nie zmienia, zignorować
OP_NOP10		

Tabela A-9 przedstawia kody operatorów zarezerwowane do wykorzystania przez wewnętrzny parser skryptów.

Tabela A-9. Zarezerwowane kody OP wykorzystywane wewnętrznie przez parser

Symbol	Wartość (szesnastkowa)	Opis
OP_SMALLDATA	0xf9	Reprezentuje małe pola danych
OP_SMALLINTEGER	0xfa	Reprezentuje małe pola liczb całkowitych
OP_PUBKEYS	0xfb	Reprezentuje pola kluczy publicznych
OP_PUBKEYHASH	0xfd	Reprezentuje pole wartości hashującej klucza publicznego
OP_PUBKEY	0xfe	Reprezentuje pole klucza publicznego
OP_INVALIDOPCODE	0xff	Reprezentuje dowolny kod OP, który nie jest obecnie przypisany

ZAŁĄCZNIK B

Propozycje udoskonalenia systemu Bitcoin

Propozycje udoskonalenia systemu *bitcoin* to seria dokumentów zawierających informacje dla społeczności *bitcoinowej* lub opisujące nowe cechy, procesy lub środowisko systemu.

Zgodnie z BIP0001 *BIP Purpose and Guidelines (Cele i wskazówki dotyczące BIP)*, wyróżnia się trzy rodzaje BIP:

Standardowe BIP

Dokumenty te opisują zmiany mające wpływ na wszystkie implementacje takie, jak zmiana protokołu sieci, zmiana zasad walidacyjnych bloku lub transakcji lub wszelkie zmiany lub dodatki, które mają wpływ na interoperacyjność aplikacji wykorzystujących *bitcoiny*.

Informacyjne BIP

Materiały te opisują kwestie związane z projektem *bitcoinów* lub zawierają ogólne wskazówki lub informacje dla społeczności *bitcoinów*, ale nie zawierają propozycji nowych zmian lub funkcji. BIP-y informacyjne niekoniecznie reprezentują *consensus* społeczności *bitcoinowej* lub rekomendacje w związku z tym użytkownicy oraz wykonawcy systemu ignorują BIP-y informacyjne lub zalecenia w nich zawarte.

BIP-y opisujące procesy

Materiały te opisują procesy *bitcoinowe* lub zawierają propozycje zmian (lub wydarzenie) pewnych procesów. BIP procesów zbliżone są w charakterze do BIP standardowych, ale dotyczą obszarów innych niż protokoły *bitcoinowe* jako takie. Mogą one sugerować implementacje, ale nie bazę kodów *bitcoin*; bardzo często wymagają również *consensus* społeczności; w przeciwieństwie do nieformalnych BIP, stanowią więcej niż zalecenia, a użytkownicy z reguły nie mogą ich ignorować. Jako przykłady można podać procedury, wskazówki, zmiany w procesach decyzyjnych oraz zmiany narzędzi lub środowiska wykorzystywanego do opracowania systemu *Bitcoin*. Wszelkie meta-BIP-y są także uznawane za BIP-y dotyczące procesów.

Propozycje dotyczące udoskonalenia systemów *bitcoin* są rejestrowane w odpowiedniej wersji repozytorium w GitHub. Tabela B-1 zawiera migawkową prezentację BIP-ów dostępnych jesienią 2014r. Należy zwrócić się do odpowiedniego repozytorium w celu uzyskania najnowszych danych w zakresie dostępnych BIP oraz ich treści.

Tabela B-1. Migawkowy obraz BIP

Nr pozycji i BIP	Link	Tytuł	Właściciel	Typ	Status
1	https://github.com/bitcoin/bips/blob/master/bip-0001.mediawiki	BIP Purpose and Guidelines	Amir Taaki	Standardowy	Aktywny
10	https://github.com/bitcoin/bips/blob/master/bip-0010.mediawiki	Multi-Sig Transaction Distribution	Alan Reiner	Informacyjny	Wersja wstępna
11	https://github.com/bitcoin/bips/blob/master/bip-0011.mediawiki	M-of-N Standard	Gavin	Standardowy	Przyjęty

	<i>ob/</i> <i>master/bip-0011.mediawiki</i>	Transactions	Andresen		
12	https://github.com/bitcoin/bips/bl/ob/master/bip-0012.mediawiki	OP_EVAL	Gavin Andresen	Standardowy	Wycofany
13	https://github.com/bitcoin/bips/bl/ob/master/bip-0013.mediawiki	Address Format for pay-to-script-hash	Gavin Andresen	Standardowy	Wersja ostateczna
14	https://github.com/bitcoin/bips/bl/ob/master/bip-0014.mediawiki	Protocol Version and User Agent	Amir Taaki, Patrick Strateman	Standardowy	Przyjęty
15	https://github.com/bitcoin/bips/bl/ob/master/bip-0015.mediawiki	Aliases	Amir Taaki	Standardowy	Wycofany
16	https://github.com/bitcoin/bips/bl/ob/master/bip-0016.mediawiki	Pay To Script Hash	Gavin Andresen	Standardowy	Przyjęty
17	https://github.com/bitcoin/bips/bl/ob/master/bip-0017.mediawiki	OP_CHECKHASHVE RIFY (CHV)	Luke Dashjr	Wycofany	Wersja wstępna
18	https://github.com/bitcoin/bips/bl/ob/master/bip-0018.mediawikilink :	hashScriptCheck	Luke Dashjr	Standardowy	Wersja wstępna
19	https://github.com/bitcoin/bips/bl/ob/master/bip-0019.mediawiki	M-of-N Standard Transactions (Low SigOp)	Luke Dashjr	Standardowy	Wersja wstępna
20	https://github.com/bitcoin/bips/bl/ob/master/bip-0020.mediawiki	URI Scheme	Luke Dashjr	Standardowy	Zastąpiony nowym dokumentem
21	https://github.com/bitcoin/bips/bl/ob/master/bip-0021.mediawiki	URI Scheme	Nils Schneider, Matt Corallo	Standardowy	Przyjęty
22	https://github.com/bitcoin/bips/bl/ob/master/bip-0022.mediawiki	getblocktemplate - Fundamentals	Luke Dashjr	Standardowy	Przyjęty
23	https://github.com/bitcoin/bips/bl/ob/master/bip-0023.mediawiki	getblocktemplate - Pooled Mining	Luke Dashjr	Standardowy	Przyjęty
30	https://github.com/bitcoin/bips/bl/ob/master/bip-0030.mediawiki	Duplicate transactions	Pieter Wuille	Standardowy	Przyjęty
31	https://github.com/bitcoin/bips/bl/ob/master/bip-0031.mediawiki	Pong message	Mike Hearn	Standardowy	Przyjęty
32	https://github.com/bitcoin/bips/bl/ob/master/bip-0032.mediawiki	Hierarchical Deterministic Wallets	Pieter Wuille	Informacyjny	Przyjęty
33	https://github.com/bitcoin/bips/bl/ob/master/bip-0033.mediawiki	Stratized Nodes	Amir Taaki	Standardowy	Wersja wstępna
34	https://github.com/bitcoin/bips/bl/ob/master/bip-0034.mediawiki	Block v2, Height in coinbase	Gavin Andresen	Standardowy	Przyjęty
35	https://github.com/bitcoin/bips/bl/ob/master/bip-0035.mediawiki	mempool message	Jeff Garzik	Standardowy	Przyjęty
36	https://github.com/bitcoin/bips/bl	Custom Services	Stefan	Standardowy	Wersja

	<i>ob/</i> <i>master/bip-0036.mediawiki</i>		Thomas		wstępna
37	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0037.mediawiki</i>	Bloom filtering	Mike Hearn and Matt Corallo	Standardowy	Przyjęty
38	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0038.mediawiki</i>	Passphrase-protected private key	Mike Caldwell	Standardowy	Wersja wstępna
39	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0039.mediawiki</i>	Mnemonic code for generating deterministic keys	Slush	Standardowy	Wersja wstępna
40		Stratum wire protocol	Slush	Standardowy	Przypisany numer BIP
41		Stratum mining protocol	Slush	Standardowy	Przypisany numer BIP
42	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0042.mediawiki</i>	A finite monetary supply for bitcoin	Pieter Wuille	Standardowy	Wersja wstępna
43	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0043.mediawiki</i>	Purpose Field for Deterministic Wallets	Slush	Standardowy	Wersja wstępna
44	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0044.mediawiki</i>	Multi-Account Hierarchy for Deterministic Wallets	Slush	Standardowy	Wersja wstępna
50	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0050.mediawiki</i>	March 2013 Chain Fork Post-Mortem	Gavin Andresen	Informacyjny	Wersja wstępna
60	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0060.mediawiki</i>	Fixed Length “version” Message (Relay-Transactions Field)	Amir Taaki	Standardowy	Wersja wstępna
61	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0061.mediawiki</i>	“reject” P2P message	Gavin Andresen	Standardowy	Wersja wstępna
62	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0062.mediawiki</i>	Dealing with malleability	Pieter Wuille	Standardowy	Wersja wstępna
63		Stealth Addresses	Peter Todd	Standardowy	Przypisany numer BIP
64	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0064.mediawiki</i>	getutxos message	Mike Hearn	Standardowy	Wersja wstępna
70	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0070.mediawiki</i>	Payment protocol	Gavin Andresen	Standardowy	Wersja wstępna
71	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0071.mediawiki</i>	Payment protocol MIME types	Gavin Andresen	Standardowy	Wersja wstępna
72	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0072.mediawiki</i>	Payment protocol URIs	Gavin Andresen	Standardowy	Wersja wstępna
73	<i>https://github.com/bitcoin/bips/bl ob/ master/bip-0073.mediawiki</i>	Use “Accept” header with Payment Request URLs	Stephen Pair	Standardowy	Wersja wstępna

ZAŁĄCZNIK C pycoin, ku oraz tx

Biblioteka Python - [pycoin](#), pierwotnie napisana i prowadzona przez Richarda Kissa, jest biblioteką opartą na systemie Python, który wspiera proces zmiany kluczy *bitcoin* oraz transakcji, a nawet wspiera język skryptów w wystarczającym stopniu, aby należycie obsługiwać transakcje niestandardowe.

Biblioteka *pycoin* wspiera zarówno system *Python 2 (2.7.x)*, jak i *Python 3* (po 3.3 oraz zawiera użyteczne narzędzia wierszy poleceń, *ku* oraz *tx*.

Key Utility (KU)

Narzędzie wiersza poleceń *ku* ("key utility") to wielofunkcyjny scyzoryk stosowany przez armię szwajcarską do zmiany kluczy. Wspiera ono klucze BIP32, WIF oraz adresy (*bitcoinowe* i *alt coinowe*). Poniżej przedstawiono kilka przykładów.

Utwórz klucz BIP32 w oparciu o domyślne źródła entropii GPG oraz */dev/random*:

```
$ ku create
input : create
network : Bitcoin
wallet key : xprv9s21ZrQH143K3LU5ctPZtBnb9kTjA5Su9DdWHXJemiJBsY7VqXUG7hipgdWaU
m2hnzdvxJf5KJo9vjP2nABX65c5sF3wSV8oXcbpehtJi
public version : xpub661MyMwAqRbcFpYYiuvZpKjKhnJDZYAlwSY76JvvD7FH4fsG3Nqiov2CfxzxY8
DGcpfT56AMFeo8M8KPkFMflUtvwjwb6WPv8rY65L2q8Hz
tree depth : 0
fingerprint : 9d9c6092
parent f'print : 00000000
child index : 0
chain code :
80574fb260edaa4905bc86c9a47d30c697c50047ed466c0d4a5167f6821e8f3c
private key : yes
secret exponent :
112471538590155650688604752840386134637231974546906847202389294096567806844862
hex :
f8a8a28b28a916e1043cc0aca52033a18a13cab1638d544006469bc171fddfbe
wif : L5Z54xi6qJuSQT42JHA44mfPVZGjyb4XBRWfxAzUWwRGx1kV4sP
uncompressed : 5KhoEavGNH4GHKoy2Ptu4KfdNp4r56L5B5un8FP6RZnbsz5NmB
public pair x :
76460638240546478364843397478278468101877117767873462127021560368290114016034
public pair y :
5980789657469774102040120298272207730921291736633247737077406753676825777701
x as hex :
a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
y as hex :
843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcfd625
y parity : odd
key pair as sec :
03a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
uncompressed :
04a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcfd625
```

```
hash160 : 9d9c609247174ae323acfc96c852753fe3c8819d
uncompressed : 8870d869800c9b91ce1eb460f4c60540f87c15d7
Bitcoin address : 1FNNRQ5fSv1wBi5gyfVBs2rkNheMGt86sp
uncompressed : 1DSS5isnH4FsVaLVjeVXewVSpfqktdiQAM
```

Utwórz klucz BIP32 na podstawie hasła:



Hasło wykorzystane w tym przykładzie jest zbyt proste, aby je odgadnąć.
\$ ku P:foo

```
input : P:foo
network : Bitcoin
wallet key :
xprv9s21ZrQH143K31AgNK5pyVW23gHnkBq2wh5aEk6g1s496M8ZMjxcnCKZKgb5j
    ZoY5eSJMj2Vbyvi2hbmQnCuHBujZ2WXGTux1X2k9Krdtq
public version : xpub661MyMwAqrRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ81tgH6u4DLQWb-
    QayvtS
    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 : 5JVnZASvXDoKYJdw8SwuLHxUxaWvn9mDea6k1vRPCX7KLJWWa7W
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 : 1MwkRkogzBRMehBntgcq2ajhXCXStJXHT
```

Pobrać informacje jako JSON:

```
$ ku P:foo - P - j
{
    "y_parity": "even",
    "public_pair_y_hex":
```

```

"826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
  "private_key": "no",
  "parent_fingerprint": "00000000",
  "tree_depth": "0",
  "network": "Bitcoin",
  "btc_address_uncompressed": "1MwlkRkogzBRMehBntgcq2aJhXCXStJXHt",
  "key_pair_as_sec_uncompressed": "",
  "public_pair_x_hex": "b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f826d8b4d3010a
ea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
  "public_pair_y_hex": "b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f",
  "wallet_key": "xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWb
QayvtSVYFvXz2vPPpbXE1qj0UFidhjFj82pVShWu9cuWmb2zy",
  "chain_code": "5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc",
  "child_index": "0",
  "hash160_uncompressed": "e5bd3a7e6cb62b4c820e51200fb1c148d79e67da",
  "btc_address": "19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii",
  "fingerprint": "5d353a2e",
  "hash160": "5d353a2ecdb262477172852d57a3f11de0c19286",
  "input": "P:foo",
  "public_pair_x": "81821982719381104061777349269130419024493616650993589394553404347774393168191",
  "public_pair_y": "58994218069605424278320703250689780154785099509277691723126325051200459038290",
  "key_pair_as_sec": "02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f"
}

```

Klucz publiczny BIP32:

```

$ ku - w - P P:foo
xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWbQayvtSVYFvXz2vPPpbXE1qj0UFidhjFj82pVShWu9cuW
mb2zy

```

Wygenerować pod-klucz:

```

$ ku - w - s3/2 P:foo
xprv9wTErTSkjVJa1v4cUTIMFLWMe5eu8ErbQcs9xajnsUzCBT7ykHAwdxvG3g3f6BFk7ms5hHBvmbdutNmyg6iogWKxx6mefEw4M8E
roLgkj

```

Wzmocniony pod-klucz:

```

$ ku - w - s3/2H P:foo
xprv9wTErTSu5AWGkDeUPmqBcbZWX1xq85ZNX9iQRQW9DXwygFp7iRGJo79dsVctcsCHsnZ3XU3DhsuaGZbDh8iDkBN45k67UKsJUXM1J
fRCdn1

```

WIF:

```

$ ku - WP:foo
L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH

```

Adres:

```

$ ku - a P:foo
19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii

```

Wygenerować zestaw pod-kluczy:

```

$ ku P:foo - s 0/0 -5 - w
xprv9xWkBDFyBXmZjBG9EiXBpy67KK72fphUp9utJokEBFtjsjiuKUUDF5V3TU8U8cDzytqYn -
Sekc8bYuJS8G3bhXxKB89Ggn2dzLcojsuEdRK
xprv9xWkBDFyBXmZnZKf3bAGifK593gT7WJPnYAmvc77gUQVej5QHckc5Adtwxa28ACmANi9XhCrRvtFqQdUxt8rUgFz3souMiDdWxDZ
nQzx
xprv9xWkBDFyBXmZqdXA8y4SwqfBdy71gSW9sjx9JpCiJEiBwSMQyRxan6srXUPBtj3PTxQFkJZAiwoUpmvtrxKZu4zfsnr3pqyy2vthpkwu
oVq
xprv9xWkBDFyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuWY3zuqxYGpHfv9cnGj5P7e8EskpzKL1Y8Gk9aX6QbryA5r
aK73p
xprv9xWkBDFyBXmZv2q3N66hhZ8DAcEnQDrXML1J62krJAcf7Xb1HJwuW2MIQtCofY2jtFXdiEY8UsRNJfqK6DAdyZXdMvtaLHyWQx3FS4
A9zw
xprv9xWkBDFyBXmZw4jEYXUHYc9ftT25k9irP87n2RqfJ5bqbjKdT84Mm7Wtc2xmzFuKg7Yf7XFHKkSsaYKWKjbR54bnyAD9GzjUYbAYTt
N4ruo

```

Wygenerować odpowiednie adresy:

```
$ ku P:foo - s 0/0 -5 - a
1MrjE78H1R1rqdFrmkjdhnPUDlCIALbv3x
1AnYyVEcuqeoVzH96zj1eYKwdWfwtew2pxu
1GXr1kZfxE1FcK6ZRD5sqqqs5YfvuzA1Lb
116AXZc4bDVQrqmcinz4aaPdrYqvuiBEK
1Cz2rTlJRM6pMnxPNrRKp9ZSvRtj5dDUML
1WstdwPnU6HEUPme1DQayN9nm6j7nDVEN
```

Wygenerować odpowiednie WIF-y:

```
$ ku P:foo - s 0/0 -5 - W
L5a4iE5k9gcJKGqX3FWmxzBYQc29PvZ6pgBaePLVqT5YByEnBomx
Kyjgne6GZwPGB6G6kJEhoPbmyjMP7D5d3zRbHVjwcq4iQXD9QqKQ
L4B3ygQxK6zH2NQGxLDee2H9v4Lwg14cLJW7QwWPzCtKhdmMaQz
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmIDMrqemY8UF
L2oD6A4TUyqPF8QG4vhUFSgwCyuvvFZ3v8SKHYFDwkM765Nrfd
KzChTbc3kZFxUSJ3Kt54cxsogeFAD9CCM4zGB22si8nfKcThQn8C
```

Sprawdzić, czy działają wybierając strumień BIP32 (odpowiadający pod-kluczowi 0/3):

```
$ ku - W
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKVMAaaEosNrFBKgj2TqWuIWY3zuqxYGpHfv9cnGj5P7e8EskpzKL1Y8Gk9aX6QbryA5r
aK73p
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmIDMrqemY8UF
$ ku - a
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKVMAaaEosNrFBKgj2TqWuIWY3zuqxYGpHfv9cnGj5P7e8EskpzKL1Y8Gk9aX6QbryA5r
aK73p
116AXZc4bDVQrqmcinz4aaPdrYqvuiBEK
```

Tak, wygląda znajomo.

Z sekretnego wykładnika:

```
$ ku 1
```

```
input : 1
network : Bitcoin
secret exponent : 1
hex : 1
wif : KwDIBf89QgGbjEhKnhXJuH7LrciVrZi3qYjgd9M7rFU73sVHndWn
uncompressed : 5HpHagT65TZG1PH3CSu63k8DbpvD8s5ip4nEB3kEsreAnchuDf
public pair x :
55066263022277343669578718895168534326250603453777594175500187360389116729240
public pair y :
32670510020758816978083085130507043184471273380659243275938904335757337482424
x as hex :
79be667ef9dcbbac55a06295ce870b07029bfcd2dce28d959f2815b16f81798
y as hex :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity : even
key pair as sec :
0279be667ef9dcbbac55a06295ce870b07029bfcd2dce28d959f2815b16f81798
uncompressed :
0479be667ef9dcbbac55a06295ce870b07029bfcd2dce28d959f2815b16f81798
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160 : 751e76e8199196d454941c45d1b3a323f1433bd6
uncompressed : 91b24bf9f5288532960ac687abb035127b1d28a5
Bitcoin address : 1BgGZ9tcN4rm9KBzDn7Kp1Qz87SZ26SAMH
uncompressed : 1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZm
```

Wersja Litecoin:

```
$ ku - nL 1
```

```
input : 1
network : Litecoin
secret exponent : 1
hex : 1
wif : T33ydQRKp4FCW5LCLLUB7deioUMoveiwekdwUwyfRDeGZm76aUjV
```

```

uncompressed : 6u823ozcyt2rjPH8Z2ErsSXJB5PPQwK7VVIwwN4mxLBFrao69XQ
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 : LVuDpNCSSj6pQ719Pv6d6sUkLKoqDEVUnJ
uncompressed : LYWKqJhtPeGyBAw7WC8R3F7ovxtzAiubdM

```

Dogecoin WIF:

```

$ ku - nD - W1
QNcdLw8fHkixm6NNyN6nVwxKek4u7qrioRbQmjxac5TVoTtZuot

```

Z pary kluczy publicznych (na Testnet):

```

$ ku - nT
55066263022277343669578718895168534326250603453777594175500187360389116729240,even

```

```

input :
55066263022277343669578718895168534326250603453777594175500187360389116729240
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 : mtoKs9V381UAhUia3d7Vb9GNak8Qvmcsme

```

Z hash160:

```

$ ku 751e76e8199196d454941c45d1b3a323f1433bd6

```

```

input : 751e76e8199196d454941c45d1b3a323f1433bd6
network : Bitcoin
hash160 : 751e76e8199196d454941c45d1b3a323f1433bd6
Bitcoin address : 1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAMH

```

Jako adres Dogecoin:

```

$ ku - nD 751e76e8199196d454941c45d1b3a323f1433bd6

```

```

input : 751e76e8199196d454941c45d1b3a323f1433bd6

```

```
network : Dogecoin
hash160 : 751e76e8199196d454941c45d1b3a323f1433bd6
Dogecoin address : DFpN6QqFfUm3gKNaxN6tNcab1FArL9cZLE
```

Transaction Utility (TX)

Narzędzie wiersza poleceń tx pokazuje transakcje w formacie czytelnym dla ludzi, pobrać transakcje bazowe z pamięci *cache pycoin* lub z usług sieciowych ([łańcuch blokowy.info](#), [blockr.io](#), oraz [biteeasy.com](#), które są obecnie wspierane), połączyć transakcje, dodać lub usunąć bloki wejściowe i umieścić podpis pod transakcją.

Poniżej przedstawiono kilka przykładów.

Słynna transakcja z "pizzą" [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_SERVICE_PROVIDERS=BLOCKR_IO:BLOCKCHAIN_INFO:BITEASY:BLOCKEXPLORER
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
```

Hm! Nie ustawiliśmy usług sieciowych. Zróbmy to teraz:

```
$ PYCOIN_CACHE_DIR=~/pycoin_cache
$ PYCOIN_SERVICE_PROVIDERS=BLOCKR_IO:BLOCKCHAIN_INFO:BITEASY:BLOCKEXPLORER
$ export PYCOIN_CACHE_DIR PYCOIN_SERVICE_PROVIDERS
```

Nie można tego przeprowadzić automatycznie, zatem wierz poleceń nie będzie mieć "wycieku" potencjalnie poufnych informacji dotyczących interesującej Was transakcji na stronę stron trzecich. Jeśli nie ma to znaczenia, można umieścić te wiersze w Waszym *.profile*.

Spróbujmy ponownie:

```
$ tx 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
Version: 1 tx hash
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a 159 bytes
TxIn count: 1; TxOut count: 1
Lock time: 0 (valid anytime)
Input:
0: (unknown) from
1e133f7de73ac7d074e2746a3d6717dfc99ecaa8e9f9fade2cb8b0b20a5e0441:0
Output:
0: 1CZDM6oTttND6WPdt3D6bydo7DYKzd9Qik receives 10000000.00000 mBTC
Total output 10000000.00000 mBTC
including unspents in hex dump since transaction not fully signed
01000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a493046022100a7f26eda8749
31999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e9199238eb73f07c8f2
09504c84b80f03e30ed8169edd44f80ed17dd451901fffffff010010a5d4e80000001976a9147e
c1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000
```

** can't validate transaction as source transactions missing

Końcowa linia pojawia się, ponieważ, aby zweryfikować podpisy transakcji wymagane jest technicznie źródło transakcji. Dodajmy zatem -a aby wzbogacić transakcję o źródło informacji:

```
$ 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

010000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a493046022100a7f26eda8749
31999c90f87f01ff1fc76bcd058fe16137e0e63fdb6a35c2d78022100a61e9199238eb73f07c8f2
09504c84b80f03e30ed8169edd44f80ed17dd1f451901fffffff010010a5d4e80000001976a9147e
c1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000
```

all incoming transaction values validated

Przyjrzyjmy się teraz niewydatkowanym strumieniom wejściowym dla specyficznych adresów (UTXO). W bloku #1, widzimy transakcję w *coinbase* skierowaną do 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX. Wykorzystajmy *fetch_unspent*, aby znaleźć wszystkie *coiny* dla tych adresów:

```
$ fetch_unspent 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX
a3a6f902a51a2cbebede144e48a88c05e608c2cce28024041a5b9874013a1e2a/
0/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/333000
cea36d008badf5c7866894b191d3239de9582d89b6b452b596f1f1b76347f8cb/
31/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
065ef6b1463f552f675622a5d1fd2c08d6324b4402049f68e767a719e2049e8d/
86/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
a66ddd42f9f2491d3c336ce5527d45cc5c2163aaed3158f81dc054447f447a2/
0/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/
10000
ffd901679de65d4398de90cef68d2c3ef073c41f7e8dbec2fb5cd75fe71dfe7/0/76a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/100
d658ab87cc053b8dbcfd4aa2717fd23cc3edfe90ec75351fadd6a0f7993b461d/
5/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/911
36ebe0ca3237002acb12e1474a3859bde0ac84b419ec4ae373e63363ebef731c/
1/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/100000
fd87f9adecbb17f4ebb1673da76ff48ad29e64b7afa02fd0f2c14e43d220fe24/0/76a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/1337
dfdf0b375a987f17056e5e919ee6eadd87dad36c09c4016d4a03cea15e5c05e3/1/76a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/1337
cb2679bfd0a557b2dc0d8a6116822f3fcbe281ca3f3e18d3855aa7ea378fa373/0/76a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/1337
d6be34ccf6edddc3cf69842dce99fe503bf632ba2c2adb0f95c63f6706ae0c52/1/76a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/2000000
```



```
0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098/0/410496b538e85
3519c726a2c91e61ec11600ae1390813a627c66fb8be7947be63c52da7589379515d4e0a604f8141
```

781e62294721166bf621e73a82cbf2342c858eeac/5000000000

ZAŁĄCZNIK D

Polecenia dostępne w narzędziach sx

Dostępne są następujące polecenia sx :

DEPRECATED

ELECTRUM STYLE DETERMINISTIC KEYS AND ADDRESSES

genaddr Generate a Bitcoin address deterministically from a wallet seed or master public key.

genpriv Generate a private key deterministically from a seed.

genpub Generate a public key deterministically from a wallet seed or master public key.

mpk Extract a master public key from a deterministic wallet seed.

newseed Create a new deterministic wallet seed.

EXPERIMENTAL

APPS

wallet Experimental command-line wallet.

OFFLINE BLOCKCHAIN

HEADERS

showblkhead Show the details of a block header.

OFFLINE KEYS AND ADDRESSES

BASIC

addr See Bitcoin address of a public or private key.

embed-addr Generate an address used for embedding record of data into the blockchain

get-pubkey Get the pubkey of an address if available.

newkey Create a new private key.

pubkey See the public part of a private key.

validaddr Validate an address.

BRAIN STORAGE

brainwallet Make 256 bit bitcoin private key from an arbitrary passphrase.

mnemonic Make 12 word mnemonic out of 128 bit electrum or bip32 seed.

HD / BIP32

hd-priv Create a private HD key from another HD private key.

hd-pub Create an HD public key from another HD private or public key.

hd-seed Create a random new HD key.

hd-to-address Convert an HD public or private key to a Bitcoin address.

hd-to-wif Convert an HD private key to a WIF private key.

MULTISIG ADDRESSES

scripthash Create BIP 16 script hash address from raw script hex.

STEALTH

stealth-addr See a stealth address from given input.

stealth-initiate Initiate a new stealth payment.

stealth-newkey Generate new stealth keys and an address.

stealth-show-addr Show details for a stealth address.

stealth-uncover Uncover a stealth address.

stealth-uncover-secret Uncover a stealth secret.

OFFLINE TRANSACTIONS

SCRIPTING

mktx Create an unsigned tx.

rawscript Create the raw hex representation from a script.

set-input Set a transaction input.

showscript Show the details of a raw script.

showtx Show the details of a transaction.

sign-input Sign a transaction input.

unwrap Validates checksum and recovers version byte and original data from hexstring.

validsig Validate a transaction input's signature.

`wrap` Adds version byte and checksum to hexstring.

ONLINE (BITCOIN P2P)

BLOCKCHAIN UPDATES

`sendtx -node` Send transaction to a single node.

`sendtx -p2p` Send tx to bitcoin network.

ONLINE (BLOCKCHAIN.INFO)

BLOCKCHAIN QUERIES (blockchain.info)

`bci -fetch -last -height` Fetch the last block height using blockchain.info.

`bci -history` Get list of output points, values, and their spends from blockchain.info

BLOCKCHAIN UPDATES

`sendtx -bci` Send tx to blockchain.info/pushtx.

ONLINE (BLOCKEXPLORER.COM)

BLOCKCHAIN QUERIES (blockexplorer.com)

`blke - fetch -transaction` Fetches a transaction from blockexplorer.com

ONLINE (OBELISK)

BLOCKCHAIN QUERIES

`balance` Show balance of a Bitcoin address in satoshis.

`fetch -block -header` Fetch raw block header.

`fetch -last -height` Fetch the last block height.

`fetch -stealth` Fetch a stealth information using a network connection to make requests against the obelisk load balancer

backend.

`fetch -transaction` Fetch a raw transaction using a network connection to make requests against the obelisk load balancer

backend.

`fetch -transaction -index`

Fetch block height and index in block of transaction.

`get -utxo` Get enough unspent transaction outputs from a given set of addresses to pay a given number of satoshis.

`history` Get list of output points, values, and their spends for an address. grep can filter for just unspent outputs

which can

be fed into mctx.

`validtx` Validate a transaction.

BLOCKCHAIN UPDATES

`sendtx -obelisk` Send tx to obelisk server.

BLOCKCHAIN WATCHING

`monitor` Monitor an address.

`watchtx` Watch transactions from the network searching for a certain hash.

OBELISK ADMIN

`initchain` Initialize a new blockchain.

UTILITY

EC MATH

`ec -add -modp` Calculate the result of INTEGER + INTEGER.

`ec -multiply` Multiply an integer and a point together.

`ec -tweak -add` Calculate the result of POINT + INTEGER * G.

FORMAT (BASE 58)

`base58 -decode` Convert from base58 to hex.

`base58 -encode` Convert from hex to base58.

FORMAT (BASE58CHECK)

`base58check -decode` Convert from base58check to hex.

`base58check -encode` Convert from hex to base58check.

`decode -addr` Decode a address from base58check form to internal RIPEMD representation.

`encode -addr` Encode an address from internal RIPEMD representation to base58check form.

FORMAT (WIF)

```

secret - to - wif Convert a secret exponent value to Wallet Import Format.
wif - to - secret Convert a Wallet Import Format to secret exponent value.
HASHES
ripemd - hash RIPEMD hash data from STDIN.
sha256 Perform SHA256 hash of data.
MISC
qrcode Generate Bitcoin QR codes offline.
SATOSHI MATH
btc Convert Satoshis into Bitcoins.
satoshi Convert Bitcoins into Satoshis.

```

Prosimy zapoznać się z 'sx help COMMAND' w celu uzyskania bliższych informacji dotyczących poszczególnych poleceń.

Następnie, przyjrzymy się wybranym przykładom wykorzystania narzędzi sx w celu przeprowadzenia eksperymentów z kluczami i adresami. Wygenerujemy nowy klucz prywatny za przy pomocy losowej liczby wygenerowanej przez generatora systemu operacyjnego stosując polecenie newkey. Zapiszemy standardowy strumień wyjściowy do pliku *private_key*:

```

$ sx newkey > private_key
$ cat private_key
5Jgx3UAxW8AcCQCi1j7uaTaqpz2fqNR9K3r4apxdYn6fTzR1PL

```

Teraz możemy wygenerować klucz publiczny z klucza prywatnego za pomocą polecenia pubkey. Wysyłamy plik *private_key* do standardowego strumienia wejściowego i zapisujemy standardowy strumień wyjściowy polecenie w nowym pliku *public_key*:

```

$ sx pubkey < private_key > public_key
$ cat public_key
`02fca46a6006a62dfdd2dbb2149359d0d97a04f430f12a7626dd409256c12be500

```

Możemy teraz zmienić format *public_key* jako adres wykorzystujący polecenie addr command. Przekazujemy *public_key* do standardowego formatu strumienia wejściowego:

```

$ sx addr < public_key
17re1S4Q8ZHCP8Kw7xQad1Lr6XUžWUnkG

```

Wygenerowane klucze są nazywane kluczami niedeterministycznymi typu 0. Oznacza to, że każdy jest generowany przez generator liczb losowych. Narzędzia sx także wspierają klucze deterministyczne typu 2, gdzie klucz "główny" jest tworzony a następnie rozszerzany w celu utworzenia łańcucha lub drzewka pod-kluczy.

Najpierw generujemy "ziarno", które będzie wykorzystane, jako podstawa do zbudowania łańcucha kluczy zgodnego z portfelem *Electrum* oraz innymi podobnymi implementacjami. Wykorzystujemy polecenie new seed, aby uzyskać nową wartość ziarna:

```

$ sx newseed > seed
$ cat seed
eb68ee9f3df6bd4441a9feadec179ff1

```

Wartość ziarna można także wyeksportować jako narzędzie – słowo mnemoniczne, które jest czytelne dla ludzi oraz łatwe do przechowywania oraz zapisywania, niż strumień szesnastkowy wykorzystujący polecenie mnemonic:

```

$ sx mnemonic < seed > words
$ cat words
adore repeat vision worst especially veil inch woman cast recall dwell appreciate

```

Słowa mnemoniczne mogą być wykorzystywane w celu powielania ziarna znowu za pomocą polecenia mnemonic:

```

$ sx mnemonic < words

```

eb68ee9f3df6bd4441a9feadec179ff1

Mając ziarno, możemy teraz wygenerować sekwencje kluczy prywatnych i publicznych – łańcuch kluczy. Stosujemy plecenie genpriv, aby wygenerować sekwencję kluczy prywatnych z ziarna oraz polecenie addr, aby wygenerować korespondujący klucz publiczny:

```
$ sx genpriv 0 < seed  
5JzY2cPZGViPGgXZ4Syb9Y4eUGjJpVt6sR8noxrpEcqgyj7LK7i  
$ sx genpriv 0 < seed | sx addr  
1esVQV2vR9JZPhFeRaëWkAhznWq7Fi7t7  
$ sx genpriv 1 < seed  
5JdtL7ckAn3iFBFyVGIBs3A5TqziFTaB9f8NeyNo8crnE2Sw5Mz  
$ sx genpriv 1 < seed | sx addr  
1G1oTeXitk76c2fvQWhy4pryTdH1RTqSPW
```

Mając klucze deterministyczne możemy wygenerować i ponownie generować tysiące kluczy z jednego ziarna w łańcuchu deterministycznym. Technika ta jest wykorzystywana w wielu aplikacjach portfelowych w celu wygenerowania kluczy, które mogą być zapisywane i odzyskiwane za pomocą prostego słowa mnemonicznego składającego się z kilku elementów. Jest to łatwiejsze niż zapisywanie informacji w portfelu za pomocą losowo wygenerowanych kluczy zawsze, kiedy jest tworzony nowy klucz.

O autorze

Andreas M. Antonopoulos jest cenionym specjalistą w dziedzinie nowych technologii oraz przedsiębiorcą, który stał się jedną z najbardziej znanych i szanowanych postaci w świecie bitcoinów. Ten doskonały mówca, nauczyciel oraz pisarz ma zdolność przekazywania różnych tematów w sposób przystępny i łatwy do zrozumienia. Jako doradca wspiera nowopowstałe firmy w procesie identyfikacji, oceny oraz zarządzania bezpieczeństwem i ryzykiem.

Andreas wychował się dobie internetu; swoją pierwszą firmę – jedną z pierwszych BBS oraz proto-ISP, założył jako nastolatek jeszcze w Grecji. Następnie zdobywał kwalifikacje i kolejne stopnie naukowe z dziedziny informatyki, przesyłu danych oraz systemów rozproszonych na University College London (UCL) – jednego z 10 najlepszych uniwersytetu na świecie. Po przeniesieniu się do Stanów Zjednoczonych, Andreas był jednym z założycieli i zarządzał instytutem badań nowych technologii i w tym charakterze świadczył usługi konsultacyjne dla dziesiątek firm z grupy Fortune 500 i ich kierownictwa w zakresie sieci, zabezpieczeń, centrów danych oraz *cloud computing*. Jest autorem ponad 200 artykułów poświęconych zabezpieczeniom, centrom danych oraz *cloud computing*, które zostały opublikowane i dostępne w różnych kanałach informacyjnych na całym świecie. Jest także zarejestrowanym posiadaczem praw patentowych związanych z siecią i zabezpieczeniami.

W roku 1990, Andreas zaczął prowadzić wykłady poświęcone informatyce w środowiskach prywatnych, zawodowych oraz akademickich. Swoje umiejętności jako wykładowca doskonalił przed takimi widowniami, jak pięciu członków zarządu, aż po tysiące słuchaczy zgromadzonych w ogromnych salach konferencyjnych. Mając ponad 400 zaproszeń do wygłoszenia wykładów jest uważany za światowej klasy, charyzmatycznego mówcę i nauczyciela. W 2014r., został mianowany asystentem na Uniwersytecie w Nikozji – pierwszym uniwersytecie na świecie oferującym tytuły magisterskie w dziedzinie walut cyfrowych. Pełniąc te funkcje, był aktywnie zaangażowany w procesie opracowywania programów studiów oraz był współ-wykładowcą w przedmiotach związanych z Wprowadzeniem do walut cyfrowych, prowadził niezwykle popularny otwarty kurs online (massive open online course - MOOC) pod auspicjami uniwersytetu.

Jako przedsiębiorca w sektorze bitcoinów, Andreas jest założycielem wielu firm *bitcoinowych* oraz inicjatorem wielu projektów typu *community open source projects*. Świadczy także usługi doradcze dla wielu firm z sektora bitcoinów i kryptowalut. Jest także bardzo popularnym autorem artykułów oraz blogów poświęconych *bitcoynom*, stałym gospodarzem podcastów *bitcoinowych*: *Let's Talk Bitcoin* oraz często wygłasza wykłady poświęcone technologii i zabezpieczeniom na konferencjach organizowanych na całym świecie.

Informacja od wydawcy

Insekt umieszczony na okładce należy do gatunku mrówek tnących liście (*Atta colombica*). Mrówki te (nazwa nie jest właściwa nazwa gatunku) to tropikalne mrówki hodujące grzyby zamieszkujące obszary Ameryki Południowej i środkowej, Meksyku oraz na południu Stanów Zjednoczonych. Oprócz ludzi, mrówki te tworzą największe i najbardziej złożone społeczności na świecie. Nazwa pochodzi od sposobu w jaki przeżuwają liście, które stanowią podstawę do uprawy ich ogródków grzybowych.

Skrzydlate mrówki zarówno okazy męskie jak i żeńskie biorą udział w masowym opuszczaniu gniazda znanym jako *revoada*, lub inaczej lotem godowym. Osobniki żeńskie łączą się z wieloma osobnikami męskimi gromadząc średnio 300 milionów plemników niezbędnych do założenia kolonii. Osobniki żeńskie także przechowują fragmenty grzybni macierzystych ogródków w kieszeni wewnętrznej policzkowej znajdującej się w jamie ustnej; następnie ich zwartość wykorzystują do założenia swoich własnych ogrodów. Po powrocie na ziemię, samica traci skrzydła i buduje podziemny matecznik dla swojej kolonii. Współczynnik sukcesu nowych królowych jest dość niski – zaledwie 2,5% założonych kolonii ma szanse na przetrwanie.

Kiedy kolonia osiąga stadium pełnego rozwoju, mrówki zaczynają się dzielić na kasty w oparciu o kryterium wielkości, z których każda pełni odmienne funkcje. Z reguły następuje podział na cztery kasty: najmniejszych (*minims*) – najmniejszych robotnic, które zajmują się potomstwem i ogródkami; małych (*minors*) – nieco większe, niż *minims*, które jednocześnie stanowią pierwszą linię obrony dla kolonii i patrolują teren bezpośrednio sąsiadujący z kolonią atakując wrogów w razie potrzeby; średnie (*mediae*) – myśliwi, którzy tną liście i przynoszą ich fragmenty do gniazda; oraz duże (*majors*) – największe mrówki robotnice, które pełnią funkcje żołnierzy i bronią gniazda przed atakiem intruzów. Najnowsze badania wykazały, że *majors* potrafią także oczyszczać główne szalki, na których zdobywają pożywienie i transportować przedmioty dużych rozmiarów do swojego gniazda.

Wiele gatunków umieszczonych na okładkach wydawnictwa O'Reilly to gatunki zagrozone; wszystkie mają istotne znaczenie dla świata. Aby dowiedzieć się więcej na temat możliwości udzielenia wsparcia można znaleźć na stronie animals.oreilly.com.

Zdjęcie na okładce pochodzi z *Insects Abroad*. Czcionka użyta na okładce to URW Typewriter oraz Guardian Sans. Czcionka użyta w tekście to Adobe Minion Pro; nagłówek - Adobe Myriad Condensed, a czcionki kodów przy wykorzystaniu Ubuntu Mono Dalton Maag.