Sprawozdanie
system kryptowaluty BasicCoin

Kryptografia Stosowana 24Z
Filip Horst 311257

1 Informacje o systemie

1.1 Architektura

Pliki, z których składa się system:

- **Node.js** – najważniejszy, implementuje uruchamiane węzły, czyli całą komunikację, mechanizmy weryfikacji i walidacji poszczególnych elementów itp. Z powodu zbieżności nazw z użytym środowiskiem programistycznym, kiedy jest mowa o pliku źródłowym to zawsze jest to zaznaczone.
- ATM.js – program do obsługi lokalnego portfela, za jego pośrednictwem wysyłane są nowe transakcje
- Wallet.js – backend portfela, tutaj generowane i szyfrowane są klucze, komunikuje się z bazą danych Sqlite3. Istnieje wersja Wallet_EC dostosowana do potrzeb krzywych eliptycznych – podział na dwa pliki w celu szybkiej i łatwej zmiany
- Miner.js – funkcja kopiąca startowana jako oddzielny wątek przez Node
- ConsoleLogger.js – własna implementacja loggera – nie ma znaczenia dla samego działania waluty

Poza tym w repozytorium znajdują się skrypty testowe, a w folderze tests dokładniejsze testy, gdzie wykorzystywane są zmodyfikowane wersje plików źródłowych. Podział testów jest w skrócie przybliżony w pliku md.

1.2 Informacje ogólne

Do najbardziej istotnych informacji o budowie systemu można zaliczyć:

- Węzły w sieci komunikują się za pomocą **żądań http** z danymi w **formacie JSON**.
- Węzły dołączając do sieci podają adres docelowy, z którym tworzone jest połączenie dwukierunkowe
- Każdy węzeł ma prywatny portfel, w którym może przechowywać dowolną liczbę kont
- Klucz prywatny zawarty w portfelu jest szyfrowany szyfrem symetrycznym z podanym przez użytkownika hasłem
- **Identyfikator konta** jest skrótem klucza publicznego
- **Blok GENESIS** zawiera **transakcję COINBASE** do jednego z kont i jest wbudowany na stałe w kod programu
- COINBASE nie jest akceptowany przez żaden węzeł – jest możliwy do przekazania tylko przez wbudowany GENESIS
- **Kopacze pracują za darmo** tj. nie otrzymują żadnego procenta, czy stałej nagrody z transakcji
- Bloki są tworzone dla pojedynczych transakcji
- Lista kopaczy jest zawarta w kodzie węzła
- Przetwarzanie transakcji odbywa się w dwóch etapach: pierwszym jest transmisja broadcast samej transakcji tak, aby mogła dotrzeć do kopaczy. Nie jest ona wtedy walidowana pod kątem logicznym (czy konto x ma wystarczająco monet itp.) tylko pod kątem poprawności

struktury i mechanizmów zabezpieczających. Drugim etapem jest transmisja wykopanego bloku.

- **Walidacja transakcji** polega na sprawdzeniu czy podany hash jest poprawny, czy podpis dla otrzymanego hasha jest poprawny oraz czy adres konta wysyłającego zgadza się z przesłanych kluczem publicznym
- **Weryfikacja transakcji** (czy konto ma monety) polega na przejściu od początku najdłuższego łańcucha po wszystkich transakcjach, obliczenie aktualnych stan kont i sprawdzenie spójności po dodaniu sprawdzanej transakcji
- **Weryfikacja bloku** polega na sprawdzeniu, czy przesłany hash jest poprawny i ma odpowiednią trudność oraz czy transakcja wewnątrz jest poprawna (ponowna walidacja i weryfikacja)
- System toleruje **tymczasowe (płytkie) rozwidlenia** priorytetyzując wcześniej otrzymane bloki
- System toleruje **głębsze rozwidlenia** (np. przy podziale na dwie podsieci). Synchronizacja polega na odpytaniu sąsiadów o podanie łańcucha dla nieznanego otrzymanego bloku i przejściu po nim od początku traktując bloki jak nowo otrzymane (weryfikacja, walidacja itp.) (proces wykonywany tylko jeśli nowy łańcuch jest dłuższy niż ten znany).
- System nie akceptuje przestarzałych bloków, ale pozwala na dołączenie bloków do łańcucha, kiedy poprzedni blok jest nie dalej niż 2 kroki od najdłuższego w ścieżce. Dzięki temu sieć działa przy niskiej trudności hasha, gdzie dochodzi do wielu rozwidleń.
- System obsługuje **orphan blocks**. Jeśli wykryta zostanie gałąź, która ma długość krótszą niż najdłuższy łańcuch pomniejszony o 1 to transakcja jest wycofywana do pamięci na początek kolejki – pomaga to zachować spójność.

W razie niewystarczających danych, bądź braku zrozumienia działania sieci zalecane jest spojrzenie na plik Main: /doc/scenariusz.md na repozytorium, gdzie zawarty jest uproszczony scenariusz krok po kroku z zagnieżdżeniami prezentujący historię przetwarzania transakcji z uwzględnieniem wielu sytuacji wyjątkowych.

2 Wybrane algorytmy kryptograficzne i uzasadnienie

2.1 Biblioteka kryptograficzna

Wykorzystana została biblioteka **Crypto** wbudowana w **Node.js**. Powodem wyboru była powszechna popularność, dobre opinie i deklarowana przez twórców wysoka jakość oraz bezpieczeństwo. Jest ona pewnego rodzaju standardem dla operacji kryptograficznych w Node. Ponadto, biblioteka oferuje automatyzacje zaawansowanych metod i wymagań niektórych szyfrów. Mowa tutaj o przekształcaniu hasła w klucz oraz bezpieczna inicjalizacja wektora IV dla szyfru symetrycznego (https://nodejs.org/api/crypto.html#keyobjectexportoptions wspomina o użyciu PKCS5 , https://datatracker.ietf.org/doc/html/rfc2898#section-3 opisuje proces tworzenia kluczy w bezpieczny sposób - z użyciem przekształceń itd.) Podsumowując, wybór został oparty o powszechne opinie nt. bezpieczeństwa, deklaracje bezpiecznego, ustandaryzowanego podejścia do problemów oraz łatwość użycia.

2.2 Szyfr asymetryczny

Wybrany został algorytm **RSA** ze względu na jego aktualność i popularność w rozwiązaniach produkcyjnych powszechnie uznawanych za bezpieczne. Jedynym rywalem, który spełniał też ten warunek był algorytm ECC, jednak z powodu lepszego osobistego zrozumienia działania wybrany został RSA.

Długość klucza (modulus) została ustawiona na 4096 bitów. Celem takiego wyboru była zgodność z zaleceniami standardów bezpieczeństwa dostępnych pod adresem https://www.keylength.com/en/6/

Po badaniach implementacji (opisanych dalej), głębszej analizie problemu oraz wysłuchaniu dalszych wykładów z przedmiotu Kryptografia Stosowana, udało się dojść do wniosku, że ECC jest po prostu lepsze dlatego w nowej wersji to ten algorytm zostałby zaimplementowany. Jako długość klucza wystarczyło by wtedy przyjąć 384 lub dla absolutnej pewności 512 z tych samych powodów co przy rozmiarze modulus.

2.2.1 Test ewentualnej migracji na EC

Zgodnie z oczekiwaniami migracja z RSA na EC jest banalnie prosta w użytej bibliotece. Edycji wymagały tylko pliki AppConfig, Wallet (druga wersja nazwana jako Wallet_EC by ułatwić przełączanie obu wersji) oraz import w ATM. Dokładniej, wymaganą zmianą było przestawienie parametru algorithm z 'rsa' na 'ec' oraz dodanie parametru 'namedCurve'. Wybrana została krzywa 'secp384r1', aby zapewnić długotrwałe bezpieczeństwo. Po wprowadzeniu takich modyfikacji należało jednak utworzyć nowe konta i poprawić adresy docelowe transakcji zawarte w skryptach testowych. Po wykonaniu tych czynności sieć działała poprawnie.

2.3 Funkcja skrótu (hash)

Wybrana została funkcja **SHA-256**, ponieważ jest ona popularnym rozwiązaniem w systemach uważanych za bezpieczne oraz jest zgodna ze wspomnianymi wcześniej standardami.

2.4 Szyfr symetryczny

Wybrany został szyfr **AES** ze względu na powszechność użycia. Długość klucza **256** w celu zgodności ze standardami. Ostatnim wyborem jest **tryb CBC**, który został uznany jako lepszy spośród dwóch w zestawieniu z ECB (osobista wiedza i doświadczenia obejmowała tylko te tryby). Ponadto, jest on bezpieczny przy założeniu, że klucz oraz wektor inicjalizujący zostaną dobrane z odpowiednimi założeniami, które są realizowane przez bibliotekę zgodnie ze standardem PKCS#5 v2 (https://nodejs.org/api/crypto.html#keyobjectexportoptions).

Znalezioną najlepszą alternatywą był tryb GCM, który według wielu opinii jest uważany za po prostu lepszy od CBC. W użytej bibliotece wymagał by on jednak wykonania szyfrowania symetrycznego ręcznie wykonując po kolei wszystkie kroki bezpiecznego szyfrowania tj. generowanie klucza, wektora IV, losowanie i zapis soli itd.. Cały ten proces powinien spełniać odpowiednie standardy, a biorąc pod uwagę, że projekt dotyczy również tworzenia niebanalnej sieci kryptowaluty to łatwo było by popełnić mniejszy lub większy błąd/niedopatrzenie, które naruszyłoby bezpieczeństwo szyfrowania. Na podstawie wspomnianych wymogów oraz braku wystarczającej wiedzy teoretycznej na temat samego GCM (szczególnie na temat elementu autentykacji) oraz dostatecznego bezpieczeństwa CBC w zadaniu przechowywania plików lokalnych zdecydowano się ostatecznie użyć CBC.

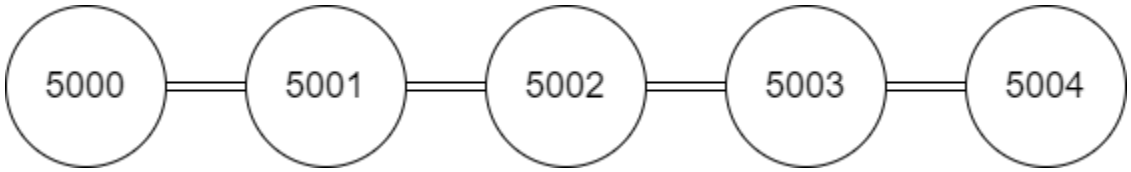
2.5 Dodatkowe uzasadnienie

W poprzednich sekcjach wiele razy wspomniana została popularność jako argument wyboru. W przypadku cyberbezpieczeństwa nie jest to oznaka lenistwa. Popularność jest przejawem tego, że metody są powszechnie uważane za bezpieczne, co jest kluczową cechą. Ponadto, szukając „nieoszlifowanych diamentów" w tym kontekście łatwo wpaść w pułapki jakimi są niedokładnie lub celowo zainfekowane biblioteki, czy algorytmy. Oczywiście upewniono się, że parametry i zastosowania użyte jako punkt odniesienia mają podobne wymagania dotyczące bezpieczeństwa, co implementowany program.

3 Analiza scenariuszy zaburzeń systemu

Sekcja zawiera analizę scenariuszy zaburzeń polegającą na testach, interpretacji wyników i ogólnych rozważaniach. Niektóre elementy mogą zostać uznane za nadmiarowe, ale jak chodzi o testy bezpieczeństwa to lepiej zbadać za dużo, niż za mało, dlatego mimo obszerności zostały załączone do sprawozdania – jeśli, któryś aspekt jest nieistotny w kontekście projektu to zawsze może zostać pominięty.

3.1 Wykorzystywana sieć



Schemat sieci testowej

Do testów wykorzystywana była sieć składające się z 5 węzłów. Została użyta do wszystkich testów, chyba że we wstępie do opisu zostało wspomniane inaczej. Dla uproszczenia opisów każdy węzeł posługuje się tylko jednym kontem (z wyjątkiem 5001, który ma konto coinbase służące do inicjalizacji stanów kont), które można utożsamiać z jego nazwą, czyli numerem portu np. 5003.

3.2 Dominacja obliczeniowa jednego węzła

Scenariusz zakłada, że jeden węzeł jest w stanie wykopać blok z dużą większą prędkością niż pozostali kopacze.

Scenariusz jest realizowany poprzez dodanie dodatkowej trudności (parametr DIFFICULTY + 2) do wymagań kopania dla każdego węzła oprócz 5001, który wciela się w rolę dominatora. Węzły wykopane przez niego są wciąż poprawne, ponieważ trudność wymagana przy weryfikacji bloku nie ulega zmianie. Kopacze w tym teście to węzły 5001, 5002, 5003, 5004.

3.2.1 Przypadek bez dominatora

Do porównania wyników wykorzystano wizualizacje oraz liczność bloków wykopanych przez różne węzły co można sprawdzić odpytując wybrany węzeł o listę bloków, grupując je według kopacza i przedstawiając liczebności (program gb.js do tego grupowania został wygenerowany przez Copilot i sprawdzony osobiście – było to możliwe ponieważ jest relatywnie prosty). Przy zwykłej pracy sieci:

Miner Counts: { '5001': 5, '5002': 5, '5003': 3, '5004': 5, undefined: 1 }

Undefined to blok GENESIS. W pozostałych przypadkach widać, że różni kopacze wykopali podobne ilości bloków. W skrypcie wysyłane jest 9 transakcji, a suma jest dużo większa – jest to skutek losowo tworzonych rozwidleń, które nie przeszkadzają w badaniu, ale trzeba o nich pamiętać.

3.2.2 Przypadek uczciwy

- Trudność dla dominatora: 2
- Trudność dla kopaczy: 2 + 2

Przypadek w którym jeden węzeł dominuje, ale jest uczciwy sprawia, że wszystkie bloki są wykopane przez niego. Wynikiem przykładowego skryptu jest liczność bloków:

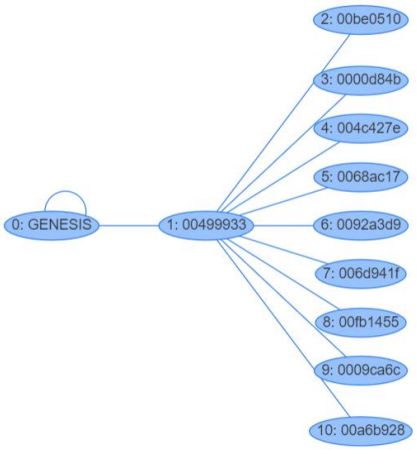
Miner Counts: { '5001': 9, undefined: 1 }

Łatwo zauważyć dominację 5001 oraz fakt, że nie dochodzi do żadnych rozwidleń. W przypadku założenia, że kopacz nie dostaje nagrody za dany blok taka sytuacja nie ma większego znaczenia,

natomiast w prawdziwej sieci sprawiło by to, że 5001 bogacił by się znacznie szybciej niż inni. Mogło by to spowodować zarówno, że pozostali zrezygnowali by z monety powodując jej upadek, ale mogło by to też zadziałać jako motywacja do zwiększenia możliwości obliczeniowych w celu rywalizacji.

3.2.3 Złośliwy węzeł dominujący – celowe rozwidlanie

Test zakładał włączenie złośliwości 5001 w postaci celowego rozwidlania łańcucha (kopie nowy blok nie dla poprzedniego bloku będącego końcem łańcucha tylko dla przedostatniego). Liczebności wykopanych bloków są takie same jak w poprzedniej podsekcji. To co jest ważne to wygląd łańcucha:



Łańcuch w sieci ze złośliwym dominatorem

Powstały w tym scenariuszu łańcuch pokazuje pełną kontrolę węzła nad siecią. Żaden inny węzeł nie ma szansy wykopać bloku, który wydłuży łańcuch, ponieważ brakuje im mocy, a węzeł dominujący celowo tego nie robi – oznacza to **całkowite zatrzymanie sieci**!

Ryzyko całkowitego zniszczenia sieci zależy od rozmiaru i parametrów trudności dla waluty. W bardzo rozległym systemie z odpowiednio krótkim czasem kopania istnieje szansa, że węzły-sąsiedzi tego, który inicjalizuje transakcje byli by w stanie wykopać ją szybciej niż rozpropagowała by się do daleko położonego dominatora. Nie zmienia to faktu, że nawet w optymistycznym przypadku sieć i jej działanie było by **zaburzone**.

3.2.4 Złośliwy węzeł – wysyłanie przestarzałych bloków

Test wysyłania przestarzałych bloków polegał na tym, że złośliwy node podmieniał pole prev_hash przed kopaniem na wartość hasha bloku cofniętego o N miejsc w łańcuchu. Dla krótszych łańcuchów na starcie wykonywał wszystko uczciwie.

W zależności od ustawionego cofnięcia siec reagowała inaczej. Dla cofnięć 1 oraz 2 bloki były akceptowane – jest to mechanizm pozwalający na pracę sieci przy szybkim kopaniu, który pozwala na akceptację poprawnych bloków kiedy po prostu dotarły z opóźnieniem. Dla większych cofnięć **przestarzałe bloki nie były akceptowane**.

3.2.5 Złośliwy węzeł – atak „double spending”

Węzeł dominujący posiada zdecydowaną większość zasobów obliczeniowych. Nie pozwala mu to na tworzenie niepoprawnych transakcji, ale daje możliwość wycofywania własnych wydatków poprzez celowe tworzenie rozwidleń. W takim scenariuszu atakujący może zapłacić za jakąś usługę, a następnie utworzyć rozwidlenie, które nie zawiera tej transakcji i dzięki przewadze mocy obliczeniowej mieć dużą szansę na budowanie dalszych bloków szybciej niż uczciwe węzły, które będą dążyć do wykopania tej wycofanej opłaty. Taka sytuacja pozwala oszustowi na wykorzystanie swoich

środków po raz drugi np. na otrzymanie usługi po raz drugi lub na przesłanie tych monet na swoje drugie konto żeby wywołać błąd balansu u uczciwych węzłów i zatrzymać ich próby dołączenia starszej opłaty do bloku.

Żeby wykonać test należało dodać dwa elementy logiki węzła oszusta (port 5001):

- Celowe tworzenie rozwidleń jeśli przychodząca transakcja jest wysłana z konta należącego do 5001 (adres konta „305…"). Tworzenie rozwidleń poprzez podłączanie nowego bloku do przedostatniego bloku aktualnego węzła (zamiast ostatniego)
- W logice zapisu nowego łańcucha jeśli nowy najdłuższy łańcuch posiada niechciany wydatek to nie jest on zapisywany jako najdłuższy. Skutkuje to tym, że kolejny blok jest kopany dla łańcucha bez wydatku.

Uruchomiony scenariusz zakładał:

- Dwie transakcje startowe
- Wydatek 5001
- 5 transakcji wysyłanych przez 5003 (tu mogło być cokolwiek – byle sieć żyła)

Otrzymano następujące wyniki:

Struktura łańcucha na uczciwym węźle 5003 – atak double spending z dominacją obliczeniową

Wykorzystując prosty program utworzony z pomocą Copilot (nazwany pblocks.js) można pokazać listę bloków, które zawierają transakcje, gdzie wysyłający („sender") to konto oszusta:

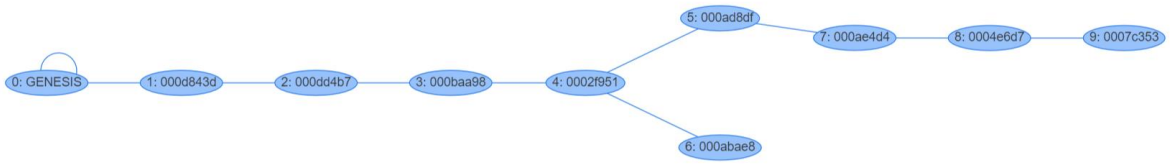
- 000b08c8... #3
- 0001f158... #6
- 000fd6ff... #9

Są to właśnie bloki z porzuconych gałęzi, czyli **atak double spending miał prawo się udać** – zależy to od zasad uznania opłaty przez wybranego usługodawcę. Jest to ważny wniosek, ponieważ oznacza, że nawet przy dominacji atak nie musi się udać – ma dopiero taką możliwość.

Według artykułu dotyczącego słabości bitcoin (https://en.bitcoin.it/wiki/Weaknesses#Attacker_has_a_lot_of_computing_power) ataki związane z przejęciem większości mocy obliczeniowej są naturą tego typu systemów, ale są praktycznie niemożliwe do wystąpienia ze względu na skrajną nieopłacalność całego procesu z perspektywy atakującego – dla dużych sieci uzyskanie przewagi mocy wymagało by ogromnej liczby potężnych komputerów, a dla małych sieci potencjalny zysk jest dla takiej osoby praktycznie niczym w porównaniu do zwykłego kopania lepszych monet.

3.2.5.1 Porównanie sytuacji bez dominacji

Po wyłączeniu mechanizmu symulującego dominację obliczeniową:



Struktura łańcucha na uczciwym węźle 5003 – atak double spending bez dominacji

, gdzie opłatą jest blok:

000baa98..., czyli #3

W tym scenariuszu atak się nie powiódł, bo wydatek został wykopany przez inny węzeł – potwierdza to, że atak może się udać tylko przy dominacji obliczeniowej.

3.2.6 Złośliwy węzeł – niedopuszczane transakcje

Biorąc pod uwagę wyniki poprzednich testów można się spodziewać, że węzeł kontrolujący większość mocy mógłby spróbować wykonać atak polegający na celowym pomijaniu wybranej transakcji. Celem takiego ataku może być konkurencyjna korporacja, której opłaty serwisowe, czynsze itp. mogą być blokowane przez atakujących w celu spowodowania bankructwa przeciwnika.

3.2.7 Propozycja rozwiązania

Co do samego przejmowania kontroli, najlepszą metodą wydaje się być znaczne **zwiększenie trudności kopania**. Dla bardzo wysokiej trudności element losowości przy poszukiwaniu będzie miał dużo większe znaczenie niż dla łatwych hashy, gdzie przeszukanie wszystkich kombinacji jest błyskawiczne.

W celu przetestowania zmienione zostały parametry trudności:

- Trudność dla dominatora: 5
- Trudność dla kopaczy: 5 + 2

Co dało wyniki:

Miner Counts: { '5001': 7, undefined: 1 } (zatrzymane wcześniej z powodu czasochłonności)

Niestety nie pomogło to w badanym problemie. Hipotezą jest to, że **trudność jest wciąż zbyt mała**, a **dysproporcja między węzłami** (trudność +2) wręcz **nierealnie duża**, ale po zmniejszeniu jej do +1 dominacja nie była zbyt zauważalna nawet dla mniejszych trudności. Żeby upewnić się, że metoda rozwiązania ma sens należałoby sprawdzić jeszcze większe trudności, bądź zastosować inną metodę spowalniania innych węzłów np. poprzez krótkie opóźnienie każdej próby wykopania nonce.

3.3 Awaria węzła przegubowego

Scenariusz to sytuacja, w której awarii ulega połączenie typu most. Prowadzi to do utworzenia dwóch podsieci, które mogą kopać zupełnie inne transakcje. Test odbył się poprzez utworzenie sieci, gdzie 5002 jest węzłem przegubowym mającym dwa mosty. Pauzowanie go, czyli wyłączanie odpowiedzi na jakikolwiek broadcast symulowało awarię i tworzyło dwie podsieci: 5000 + 5001 oraz 5003 + 5004. W testach wykorzystywane są endpointy leave_network i pause – więcej o nich w sekcji o nieautoryzowanej kontroli.

3.3.1 Kontrolowane odłączenie

Test polegał na przetestowaniu działania mechanizmu kontrolowanego odłączenia. Kiedy węzeł otrzyma sygnał wykonuje następujące czynności:

- Odpytuje swojego mastera (adres do którego dołączył na początku), czy może opuścić sieć – ma to na celu zapobiec sytuacji, w której sąsiedzi są przekierowywani do nowego węzła, który zaraz opuści sieć
- Po otrzymaniu zezwolenia wysyła wszystkim sąsiadom adres nowego mastera
- Węzeł się zatrzymuje

Metoda jest skuteczna i dobrze zapobiega powstawaniu nadmiarowych połączeń, ale niestety jest podatna na oszustwo. **Złośliwy węzeł master mógłby nie pozwalać na odłączenie się węzła**. Żeby temu zapobiec odłączający węzeł mógłby rozesłać wszystkim pełną listę sąsiadów, ale spowodowało by to tylko nadmiarowe połączenia. Natomiast w przypadku, gdyby opuszczający był węzłem przegubowym i po prostu zignorowałby ostrzeżenie opuszczającego mastera to **sieć została by rozcięta**. W aktualnej implementacji sieci niestety żadna wersja nie jest w 100% pewna.

Test w uczciwym scenariuszu polegał na:

- Ustawienie sieci
- Wysłanie 2 transakcji startowych
- Wyłączenie węzła 5002 (bez systemu odłączania powodowałoby dziurę w sieci)
- 7 transakcji od 5001
- 5 transakcji od 5003

Po zakończeniu skryptu blockchain był identyczny dla obu 5001 i 5003, co potwierdza, że **mechanizm zadziałał w tym scenariuszu**.

Blockchain po badaniu (identyczny dla obu podsieci)

3.3.2 Trwałe wstrzymanie przekazywania danych

Podstawowa symulacja awarii pokazała reakcje okolicznych węzłów. Poniższy zrzut ekranu z węzła 5001 pokazuje, że **utrata połączenia została wykryta**. W aktualnej implementacji węzeł nie reaguje na to ostrzeżenie, więc **sieć przestaje działać**. Dodanie kodu, który spowodowałby kontrolowane wyłączenie się węzła z sieci i zwrócenie ostrzeżenia użytkownikowi po danej liczbie prób połączenia byłby jednak relatywnie prosty do dodania.

```
16:30:35.800 /broadcast Forwarding message
-payload_hash:  00f17c522...
-target:  http://localhost:5002
16:30:35.802 Can't connect to http://localhost:5002/broadcast, retrying 3 more times
16:30:35.804 Can't connect to http://localhost:5002/broadcast, retrying 2 more times
16:30:35.806 Can't connect to http://localhost:5002/broadcast, retrying 1 more times
16:30:35.808 Can't connect to http://localhost:5002/broadcast, retrying 0 more times
16:30:35.809 /broadcast Message already received. Skipping...
```

Zrzut ekranu z logów węzła 5001

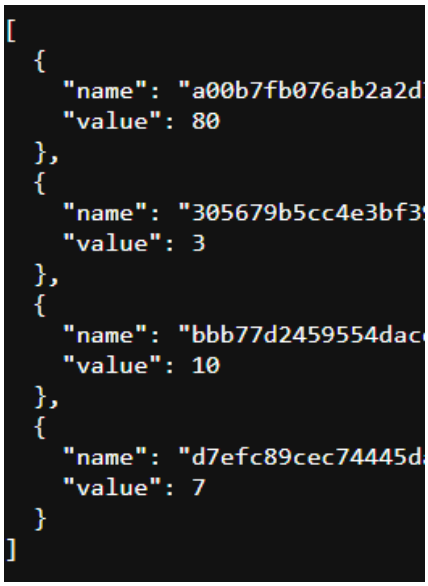
3.3.3 Tymczasowe wstrzymanie przekazywania danych

Ten scenariusz działa analogicznie do poprzedniego, z tą różnicą że węzeł przegubowy wraca po pewnym czasie. Zrealizowano to wykorzystując endpoint /pause węzła 5002. Scenariusz testu to:

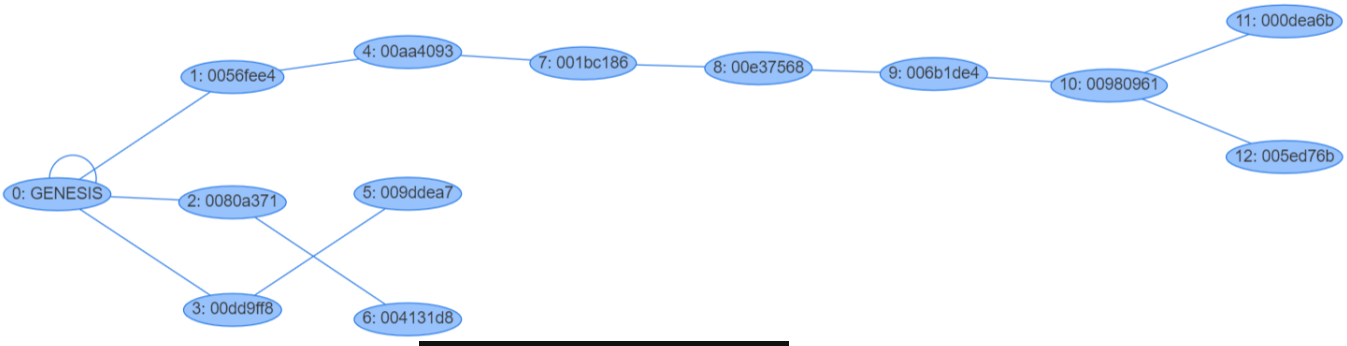
- Utworzenie sieci
- Wysłanie 2 transakcji zasilających konta 5001 oraz 5003

- Wyłączenie mostu
- Wysłanie 7 transakcji dla podsieci 1
- Wysłanie 5 transakcji dla podsieci 2
- Włączenie mostu
- Wysłanie 3 transakcji wywołujących mechanizm synchronizacji

Stan przed włączeniem mostu (do przedostatniego kroku włącznie).



Łańcuch i stany kont (podsieć 1: węzeł 5001)



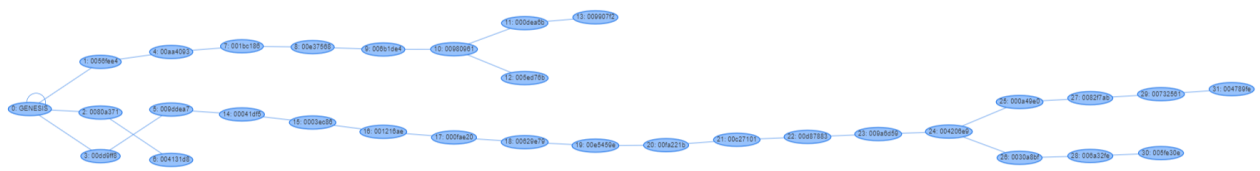

```
[
  {
    "name": "a00b7fb076ab2a2
    "value": 80
  },
  {
    "name": "305679b5cc4e3bf
    "value": 10
  },
  {
    "name": "bbb77d2459554da
    "value": 5
  },
  {
    "name": "d7efc89cec74445
    "value": 5
  }
]
```


Łańcuch i stany kont (podsieć 2: węzeł 5003)

Można zaobserwować, że zaakceptowane (najdłuższe) łańcuchy są inne. Różnią się też długością – w podsieci 1 długość to 2 (wstępne zasilanie) + 7 (transakcje w podsieci 1), natomiast dla podsieci 2 jest to 2 + 5.

Po włączeniu mostu:

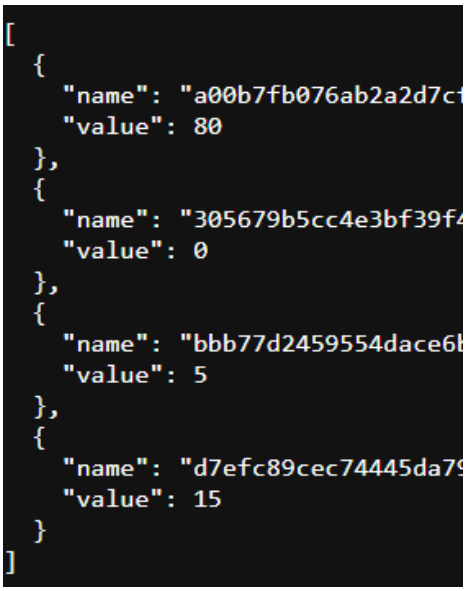
Podsieć 1 (5001) – blok końcowy 004789fe…



Podsieć 2 (5003) – blok końcowy 004789fe…

Dłuższy łańcuch podsieci wygrał i został zaakceptowany jako ten ważniejszy. Aby nie naruszać spójności sieci, wszystkie transakcje z podsieci 2 zostały wycofane i wykopane ponownie, dlatego

długość nowego wspólnego łańcucha to 17 (2 wstępne + 7 podsieć 1 + 5 podsieć 2 + 3 sychronizujące). Stany kont sprawdzone po synchronizacji również były zgodne dla wszystkich węzłów i poprawne biorąc pod uwagę skrypt rozsyłający:



Stany kont po synchronizacji (te same dla wszystkich węzłów)

Można więc powiedzieć, że sieć jest **dobrze przygotowana** na ten konkretny scenariusz.

3.3.4 Złośliwy węzeł

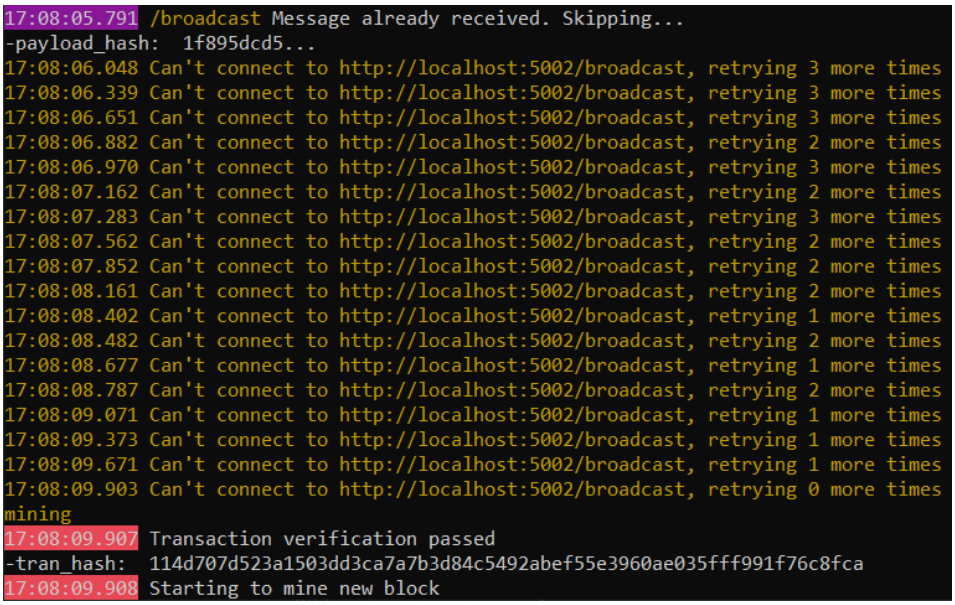
Złośliwość mogłaby polegać na wyłączeniu przekazywania wiadomości broadcast, ale w taki sposób by nie wywołać wykrycia awarii. W ramach testu odpowiedź pauzowanego węzła została zmodyfikowana tak, aby nie powodować błędów. W tym scenariuszu wszystko zadziałało tak samo, ale podsieci nigdy **nie dowiedziały się o awarii 5002**, co oznacza że jego złośliwość mogłaby spowodować **rozerwanie sieci na pomniejsze**.

3.3.5 Problem nieskończonej odpowiedzi

W przypadku, gdy węzeł nie odpowiada to istnieje ryzyko zablokowania łącza i nieskończone oczekiwanie. Problem ten został napotkany w fazie tworzenia projektu i zabezpieczony parametrem metody fetch:

signal: AbortSignal.timeout(1500)

Parametr mówi, że połączenie zostanie porzucone po 1.5s bez odpowiedzi. Ustawienie to jest jednak dość nadmiarowe i prawdopodobnie mogłoby być mniejsze.



Zrzut logów z 5001

Wadą zaimplementowanego loggera jest to, że pokazuje czas wyświetlenia wiadomości, niekoniecznie wydarzenia, a przy wielu asynchronicznych zapytaniach czasem powstaje tłok. To co jest ważne na zrzucie to to, że mimo nieudanych połączeń praca była kontynuowana, czyli **ten problem został zabezpieczony.**

3.3.6 Propozycja rozwiązań

Przy awarii łącza sytuacja jest dość prosta do rozwiązania. Awarię można rozpoznać poprzez wykrycie braku odpowiedzi na wysyłane do węzła wiadomości broadcast. Można również dla pewności okresowo wysyłać do sąsiadów prosty ping w celu diagnostycznym. W sytuacji utraty połączenia następnie należało by poczekać chwilę i spróbować ponownie, bądź od razu zwrócić użytkownikowi błąd i poprosić o ponowne dołączenie do sieci – tym razem podając inny adres docelowy połączenia.

W przypadku złośliwego węzła sytuacja się komplikuje:

Pierwszą sugestią jest tworzenie określonej liczby nadmiarowych połączeń przy dołączaniu do sieci. Dla poprawnej sieci jedno połączenie jest wystarczające jednak w celu zabezpieczenia można dołączyć np. do 2 czy 3 węzłów, które już są w sieci. Sprawi to, że powstaną połączenia zapasowe, ale kosztem zwiększonej redundancji ruchu w sieci. Liczbę nadmiarowych połączeń można by było dobrać eksperymentalnie w celu uzyskania balansu. Ważne jest też, aby nadmiarowe połączenia prowadziły do jak najbardziej odległych węzłów, aby zminimalizować ryzyko że będą one w zmowie.

3.4 Nieautoryzowana kontrola nad węzłami

3.4.1 Endpointy testowe- /pause, /leave_network i inne

Pauzowanie to mechanizm powodujący zatrzymanie odpowiedzi węzła na broadcast. Opuszczanie sieci to kontrolowany sposób wyłączania węzła. Oba te endpointy są możliwe do użycia przez każdego atakującego – jest to jednak świadome i oznaczone. Służą one ułatwieniu dostępu i skryptowania w ramach testu. W ostatecznej wersji realnego programu (nie tylko proof of concept) należało by je usunąć i ewentualnie zastąpić czymś dostępnym przez GUI bądź konsolę z poziomu użytkownika.

Program zawiera też kilka endpointów testowych do odczytu danych, które np. zwracają listę sąsiadów. Jest to oczywiście naruszenie poufności – sytuacja w tym przypadku jest analogiczna co do poprzedniej.

Podsumowując, istnieją endpointy umożliwiające **nieautoryzowany dostęp**, ale są one **wykorzystywane do testów** i **oznaczone jako niebezpieczne**.

3.4.2 Atak na kontrolowane odłączenie

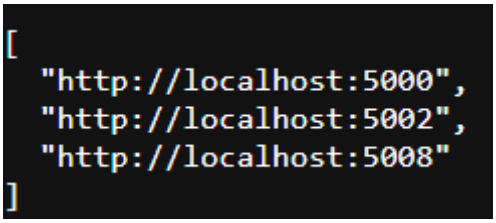
Test polegał na sprawdzeniu jakie skutki można wywołać ręcznie wysyłając zapytania na endpointy odpowiedzialne za odłączenie.

3.4.2.1 DELETE /neighbors oraz PUT /neighbors

Poprzez wywołanie Powershell:

Invoke-WebRequest -Uri http://localhost:5001/neighbors -Method Delete -Body "[`"http://localhost:5008`"]" -ContentType "application/json"

Udało się doprowadzić do dodania niestniejącego węzła do listy sąsiadów:



Sąsiedzi 5001 po teście

Oznacza to, że endpoint umożliwia **tworzenie nadmiarowych i fałszywych połączeń** złośliwym aktorom. Metoda PUT pozwala na analogiczne działanie, ale trochę innym wywołaniem.

3.4.2.2 Inne

Ważnym faktem opartym o analizę endpointów związanych z kontrolowanym odłączeniem jest to, że **nie da się doprowadzić do usunięcia** żadnego węzła z listy sąsiadów. Jeśli węzeł faktycznie opuszcza sieć to będzie wywoływać awarie połączenia – inny mechanizm usuwający nieaktywne węzły spośród sąsiadów zajął by się tym problemem. **Pod kątem usuwania dostępu innym implementacja jest bezpieczna.**

3.5 Nieuczciwy węzeł

3.5.1 Akceptacja wszystkich transakcji

Test polegał na wysłaniu trzech niepoprawnych transakcji, a następnie jednej prawdziwej. W teście oszustem był 5001.

Bloki oszusta po teście (blok 4 odpowiada legalnej transakcji, 1-3 niepoprawnym)

Bloki uczciwego kopacza po teście (blok 1 odpowiada legalnej transakcji)

Można zauważyć, że uczciwi kopacze dzięki weryfikacji bloków i transakcji nie dopuszczają nawet poprawnie wykopanych niepoprawnych transakcji. Ciekawym zjawiskiem jest to, że oszust nie przyjął poprawnego bloku wykopanego przez kogoś innego – jest tak, ponieważ węzły nie akceptują przestarzałych bloków. Można uznać, że **sieć jest odporna na tak prosty scenariusz złośliwości**.

3.5.2 Podmiana wartości w transakcji

Podmiana odbywała się na etapie przetwarzania transakcji: po jej zaakceptowaniu złośliwy węzeł podwajał wartość przesyłanej waluty i podmieniał hash transakcji na nowy.

W teście prawdziwa transakcja była rozsyłana z węzła 5001.

```
18:40:07.342 Transaction verification failed
-reason:  Transaction signature invalid
-tran_hash:  aec40b4896407cee53b19d3e0f180ecf9c00856b11b6820764da4159335ce2f6
```

Zrzut logów z węzła 5002

Jak widać weryfikacja transakcji nie powiodła się i transakcja nie została rozesłana. Błędem jest nieprawidłowość podpisu – oczywiście oszust nie może podpisać transakcji bez dostępu do czyjegoś klucza prywatnego, a jeśli ma do niego dostęp to może robić co chce bez „zabawy" w podwajanie i rehashowanie.

Ze względu na fakt, że sprawdzany jest hash całego zestawu danych i jego podpis to wiadomo, że **system jest odporny na jakiekolwiek podmiany danych transakcji.**

3.5.3 Podmiana wartości bloku

Proces analogiczny do podmiany w transakcji, ale ze względu na umiejscowienie przetwarzania bloków lepiej było przeprowadzać testy dla węzła 5000 jako kopacza (zamiast 5001, 5004 jako kopacz bez zmian). Sprawdzone zostały scenariusze:

- zmiany danych
- zmiana danych i rehash
- zmiana danych i wstawienie łańcucha znaków z odpowiednią ilością zer jako hasha

Żaden z nich się nie powiódł. Mimo niedużej liczby testów pokazują one **bezpieczeństwo systemu w tym kontekście**. Wyjątkiem zarówno w tym, jak i w przypadku transakcji są kolizje funkcji skrótu – więcej rozważań o tym zostało zawartych w dalszych sekcjach.

3.6 Ataki na portfel jako bazę danych

3.6.1 Atak na łańcuch przechowujący klucz prywatny w bazie

Według dokumentacji użytej biblioteki (https://nodejs.org/api/crypto.html#cryptocreateprivatekeykey) przy podaniu parametrów cipher oraz passphrase do klucza prywatnego przy generacji sprawia to, że przy konwersji na łańcuch jest on szyfrowany zgodnie ze standardem PKCS 5 (v2) (https://datatracker.ietf.org/doc/html/rfc2898) co oznacza m.in użycie soli, wieloiteracyjnej funkcji wyprowadzenia klucza na bazie hasła oraz wyprowadzenia zarówno klucza ostatecznego i wektora IV dla AES z wyniku wspomnianej funkcji. Ze względu na skomplikowany charakter przetwarzania testy są bardzo trudne w realizacji. Można jednak przeprowadzić test polegający na sprawdzeniu, czy klucz prywatny w bazie zgodnie z założeniem jest przechowywany w sposób zaszyfrowany:

Database string
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIJrTBXBgkqhkiG9w0BBQ0wSjApBgkqhkiG9w0BBQwwHAQIAmUq7p8SbYICAggA
MAwGCCqGSIb3DQIJBQAwHQYJYIZIAWUDBAEqBBCXNUoB1XEr0hqIhPYHbTtXBIIJ
Unencrypted key
-----BEGIN PRIVATE KEY-----
MIIJQwIBADANBgkqhkiG9w0BAQEFAASCCS0wggkpAgEAAoICAQCn2FsERpww+iiD
gH9DiIWr8njFVSvXypT/tvxKz5UMADL/mgepVlt3aL7hfS4I4o0LQ0iXuGDC1Dlk
Encrypted key
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIJrTBXBgkqhkiG9w0BBQ0wSjApBgkqhkiG9w0BBQwwHAQIYa0zvfZmPpQCAggA
MAwGCCqGSIb3DQIJBQAwHQYJYIZIAWUDBAEqBBBQNBRnh1ugkdXRBPU4Hc9hBIIJ

Przycięty zrzut z konsoli dla różnych metod wyświetlania klucza prywatnego

Z testu wynika, że łańcuch przechowywany w bazie jest zgodny z szyfrowaną wersją uzyskaną z pomocą metody export. Niestety dokładniejsze testy takie jak np. sprawdzenie kodu źródłowego i odtworzenie go poszczególnymi funkcjami w celu porównania wyniku nie były możliwe, ponieważ biblioteka korzysta z implementacji C++ dla bardziej wymagających obliczeniowo elementów takich jak szyfrowanie, a ze względu na brak wiedzy na temat tego języka analiza tak trudnego tematu nie jest możliwa.

Źródło pliku odpowiadającego za export kluczy:
https://github.com/nodejs/node/blob/main/lib/internal/crypto/keys.js

Informacja o braku dostępu do funkcji C++, z których korzysta powyżej załączony keys.js:
https://github.com/nodejs/help/issues/3079?spm=a2c6h.13046898.publish-article.12.7f356ffaNSXXIp

Przy tych testach jedyne co można powiedzieć to, że **klucz jest przechowywany poprawnie** więc **integracja z sqlite działa**, ale z samym bezpieczeństwem klucza jedyną możliwością jest **zaufanie twórcom biblioteki**.

3.6.2 Utrata danych logowania

Jeśli użytkownik straci dane logowania to jedyne do czego może uzyskać dostęp to identyfikator konta, klucz publiczny oraz zaszyfrowany klucz prywatny. Zaszyfrowana postać klucza skutkuje tym, że tak naprawdę jedyną możliwością odzyskania środków jest BruteForce, bądź znalezienie kolizji identyfikatora. Ze względu na użyte funkcje szyfrowania i skrótu jest to statystycznie niemożliwe zakładając, że biblioteka kryptograficzna działa.

3.7 Błędne żądania

Celem jest zbadanie zachowania węzłów w reakcji na niepoprawne dane żądań oraz ich nielogiczne wywołania. Reakcja nie powinna powodować uzyskania przez atakującego żadnych dodatkowych informacji, a praca węzłów nie powinna być zakłócona

Endpoint	Zmiana	Obserwacje dotyczące odpowiedzi i zachowania
Broadcast	Block hash = null	OK
Broadcast	Wysłanie bloku jako transakcji	OK, ale rozesłany dalej, chociaż nikt nie wykopał z powodu błędów weryfikacji

Broadcast	Prev_hash = null i difficulty 0	OK, ale zaczął próbować synchronizować łańcuchy – nie wykonano, bo nikt nie znał bloku null
Broadcast	Brak danych wewnętrznych	OK
Broadcast	Brak danych	OK, nieprzewidziany wyjątek w konsoli
Register_neighbor	Losowe adresy	OK, ale są dodawane
Register_neighbor	Brak danych wewnętrznych	OK, ale rejestruje undefined
Register_neighbor	Brak danych	Odpowiedź nie w formacie JSON, nieprzewidziany wyjątek w konsoli
Join_network	Odpytanie pierwszego węzła INIT	Nieskończona odpowiedź, ale węzeł przetwarza bloki i działa dalej
Leave_network	Odpytanie węzła INIT	OK, ale nie w formacie json
neighbors	DELETE	OK, ale nie w formacie json
Neighbors	PUT	OK, ale przyjmuje wszystkie wartości i je zapisuje (nawet integery, puste obiekty)
Parent	Niepoprawne hashe	OK
Broadcast	Pusty sync_chain	Nie w formacie json, status 500, nieprzewidziany wyjątek w konsoli

Podsumowując zebrane dane – implementacja węzła pod kątem odporności na nietypowe/błędne żądania **nie może zostać uznana z pewnością za bezpieczną** mimo, że **nie udało się jej zepsuć**. Powodem jest wiele przesłanek **niedokładnej implementacji pod kątem sprawdzania danych** wejściowych. Dla wielu sprawdzonych scenariuszy odpowiedzi zwracane były w złym formacie, bądź jak w przypadku /neighbors dane wejściowe nie były poprawnie walidowane. Z drugiej strony, mechanizmy walidacji w ważniejszych częściach systemu naprawiały błędy popełnione przez prostsze endpointy, więc **nie ma też jasnego dowodu że system jest niebezpieczny**. Jednak zgodnie z zasadami cyberbezpieczeństwa można powiedzieć, że **brak dowodu bezpieczeństwa można interpretować jako przesłankę istnienia podatności.**

Potencjalne poprawki są oczywiste: lepsze walidowanie danych, większe przyłożenie się do przetwarzania danych i odpowiedzi nawet w mniej istotnych dla systemu endpointach – wkońcu podatność w nieistotnej części zazwyczaj jest miejscem uzyskania dostępu skąd atakujący przechodzi do tych ważniejszych.

3.8 Złamanie algorytmów kryptograficznych

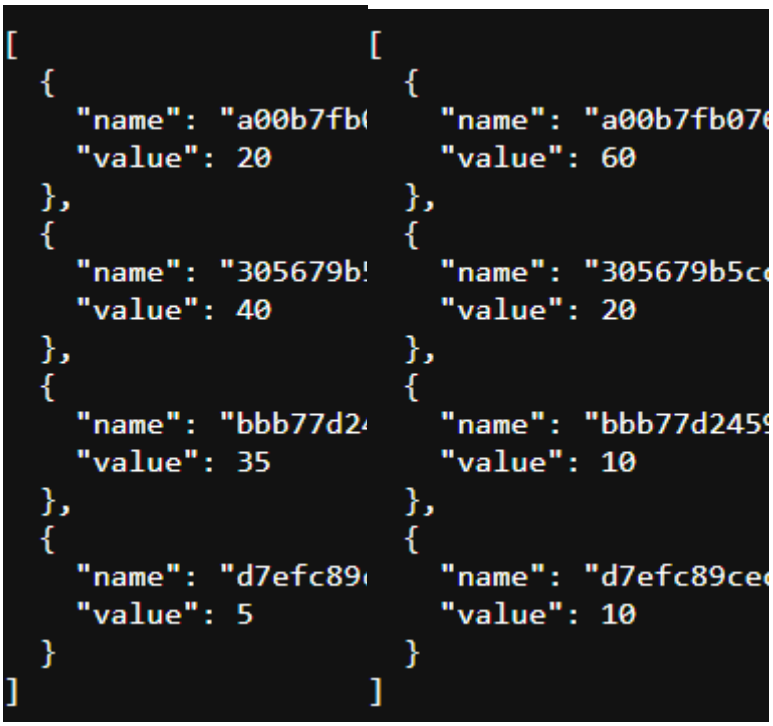
3.8.1 Złamanie hasha

Złamanie hasha można zinterpretować jako możliwość powrotu do wartości oryginalnej bądź uzyskanie możliwości wyszukiwania kolizji w krótkim czasie. W projekcie hash jest używany tylko w celach weryfikacji oraz skracania danych (id jako hash z klucza), dlatego pierwsza możliwość nie jest tutaj zagrożeniem. Skuteczny atak na hash dający możliwość łatwego tworzenia kolizji można zasymulować poprzez wyłączenie mechanizmów weryfikujących, ponieważ nie są już wiarygodne. Struktura testu:

- Kopacze na adresach 5000 i 5004

- Nieuczciwy węzeł 5001 – fałszuje zarówno otrzymane transakcje, jak i bloki poprzez podwajanie wartości transakcji
- Wysyłane po kolei transakcje:
    - 10 monet od 5001 do 5001 (adres 305…)
    - 1 moneta od 5003 do 5000 (adres d7ef…) – niepoprawna, bo 5003 nie otrzymał żadnych środków startowych od coinbase
    - 10 monet od 5001 do 5003 (adres bbb…)
    - 5 monet od 5003 do 5000 (adres d7ef) – tym razem poprawna
- W idealnych warunkach transakcje powinny zakończyć się stanem:
    - Adres 305… : 10
    - bbb… : 5
    - d7ef… : 5
    - a00b… (konto coinbase): 80

Po przeprowadzeniu testu uzyskano następujące wyniki:



Stany kont według 5003 Stany kont według 5000

Pierwszą obserwacją jest to, że fałszerstwo zostało zaakceptowane przez sieć! Jest jednak jeszcze inne interesujące zjawisko, czyli różnica w stanach kont. Jest to spowodowane tym, że transakcja o wartości 5 dla konta d7efc.. pochodzi z węzła 5003 i została wykopana przez węzeł 5004. Skutkuje to tym, że wykopany dla niej blok dociera do 5003 w niezakłóconym stanie, jednak do węzła 5000 dociera przechodząc przez fałszywy węzeł 5001, który podwaja wartość transferu.

Innym zjawiskiem są początkowe transakcje pochodzące z 5001. Oryginalnie miały one wartość 10, ale z perspektywy 5000 są warte 20 (transakcja podwojona przez fałszera), natomiast z widoku 5003 warte są aż 40 (najpierw transakcja podwojona, a później blok wykopany przez 5000 podwojony w drodze do 5003).

Trzeba się jednak zastanowić **jak wyglądało by tworzenie kolizji?** Żeby otrzymać kolizję trzeba zmodyfikować jakieś pole – w przypadku transakcji są to typ transakcji, adresy kont, wartość oraz

timestamp. Węzeł atakujący musiałby więc manipulować timestamp-em, by otrzymać kolizję, ponieważ typ, adres wychodzący i adres docelowy mają określony zbiór wartości, natomiast wartość ma stosunkowo mały przedział dozwolonych wartości spowodowany stanami kont (istnieje jednak szansa, że na wskutek procesów ekonomicznych użytkownicy przechowywali by miliardy monet, co całkowicie odwróciło by sytuację!). Można więc ten atak częściowo skontrować sprawdzaniem znaczników czasowych tak, aby nie akceptować przyszłych, bądź zbyt starych transakcji – wtedy ryzyko, że kolizja wystąpi dla pobliskiego znacznika jest bardzo niskie. Istnieje jednak możliwość, że atakujący modyfikował by adres docelowy transakcji, co mogło by umożliwić wysyłanie pieniędzy konkurenta do nieistniejących kont powodując ich utratę.

W przypadku bloków węzeł atakujący ma dowolność modyfikacji wszystkich pól oprócz wnętrza transakcji, ponieważ kolizje może znaleźć z użyciem nonce, który nie jest sprawdzany. W kontekście podmiany wartości bądź adresu transakcji nie daje to żadnych możliwości, bo hash transakcji jest weryfikowany również oddzielnie od hasha bloku.

Podsumowując, złamanie hasha polegające na ułatwieniu wyszukiwania kolizji **niszczy bezpieczeństwo i potencjalnie spójność** całej sieci w aktualnej implementacji.

3.8.2 Złamanie RSA

Ponieważ wykorzystywany jest asymetryczny algorytm RSA to największym zagrożeniem prawdopodobnie jest uzyskanie dostępu do klucza prywatnego na bazie publicznego np. na skutek dynamicznego rozwoju komputerów kwantowych. Na taki atak nie ma zabezpieczeń – kiedy do niego dojdzie to sieć przestanie działać, ponieważ pozwoli atakującemu na wysyłanie sobie dowolnych ilości monet i podpisywanie ich kluczami innych osób. Sprawa jest tak prosta, że nie potrzeba żadnych testów.

3.8.2.1 Ryzyko ślepego podpisu

Teoretycznie istnieje ryzyko wykonania przez użytkownika ślepego podpisu, który atakujący mógłby wykorzystać do wyciągnięcia klucza prywatnego. Scenariusz to sytuacja, w której wiarygodny użytkownik chce zapłacić za coś złodziejowi (nie wie, że to złodziej – może np. płacić za jakąś prawdziwą usługę). Wtedy złodziej preparuje swój adres, by wykonać atak na ślepy podpis (ślepota polega na tym, że nikt naprawdę nie wie czy docelowy adres konta to nie jest coś innego) nawet jeśli oznaczało by to utratę paru monet – potencjalny zysk znacznie większy.

Takie ryzyko jest jednak w aktualnej implementacji stosunkowo niskie, ponieważ transakcja posiada znacznik czasowy ustalany przez wysyłającego, czyli podpisującego – oznacza to, że atakujący musiałby przewidzieć jego wartość z ogromną dokładnością. Sprawia to, że taki atak nie jest realnym zagrożeniem, ale jest wykonywalny, czyli gdyby atakujący sprzedawał tysiące usług na minutę to wkońcu mógłby trafić. W implementacji nie jest użyty żaden mechanizm zabezpieczający przed tym, więc **istnieje małe, ale ryzyko naruszenia bezpieczeństwa**!

3.8.3 Propozycje poprawek

Dodanie weryfikacji znacznika czasowego tak, aby uczciwe węzły akceptowały tylko niedawne, przeszłe transakcje bardzo mocno ograniczyło by ryzyko akceptacji fałszywej transakcji, ponieważ to właśnie timestamp jest jedynym modyfikowalnym polem transakcji. Jak już zostało wspomniane, gdyby w skutek procesów ekonomicznych doszło do sytuacji gdzie przeciętny użytkownik ma miliony monet to pole amount stało by się dodatkową opcją do modyfikacji w poszukiwaniu kolizji znacznie zwiększając podatność sieci. Oznacza to, że nawet **taka poprawka tylko zmniejsza, ale nie eliminuje negatywnych skutków** sytuacji, czyli **nie daje bezpieczeństwa**. Ponadto, taka zmiana mogła by

spowodować problemy w mechanizmie synchronizacji dłuższego łańcucha otrzymanego z innej podsieci (jak w badaniu z awarią mostu).

Zmiana RSA na ECC również była by pozytywną zmianą, ponieważ krzywe eliptyczne dają podobny stopień bezpieczeństwa przy znacznie krótszych kluczach, a dodatkowo eliminuje minimalne ryzyko ataku ślepego podpisu. Dopóki sieć jest na etapie testowania to zmiana jest banalna, bo opiera się na zmianie paru parametrów biblioteki.

Inną strategią mitygacji jest monitorowanie potencjalnych atakujących i migracja na mocniejsze algorytmy gdy dojdzie do sytuacji zwiększonego ryzyka przełamania aktualnych. W sytuacji nagłego ataku jedynym potencjalnym rozwiązaniem jest zatrzymanie sieci jak najszybciej, migracja na nowsze metody i wznowienie sieci od przeszłego stanu sprzed ataku (znalezienie go może być poważnym problemem).

Ostatnim aspektem jest potencjalne szukanie kolizji przez podmienianie adresu docelowego transakcji. Aby się przed tym zabezpieczyć można by było dodać dodatkowy element weryfikacji poprawności adresu konta. Pierwszym pomysłem jest zastosowanie bardziej ustrukturyzowanego formatu, który zależał by od samej wartości hasha, ale jednocześnie byłby możliwy do rozpoznania. Niestety nie udało się znaleźć przykładu takiej transformacji. Zmiana ograniczyła by możliwości atakujących.

3.9 Atak DOS na kopacza

Test polegał na uruchomieniu uczciwej sieci w dwoma kopaczami 5001 oraz 5004, a następnie porównaniu liczby wykopanych bloków przez każdego z nich w sytuacji bez oraz z atakiem typu DOS wycelowanym w węzeł 5004. Atak polegał na prostej pętli wywołań programu curl wysyłającego błędne żądanie POST na /broadcast. Pozostałe parametry: trudność 4, transakcje były wysyłane z węzła 5003, czyli były bliżej kopacza 5004.

Brak DOS:

- Miner Counts: { '5004': 7, undefined: 1 }
- Miner Counts: { '5004': 7, undefined: 1 }
- Miner Counts: { '5001': 1, '5004': 6, undefined: 1 }

Z DOS:

- Miner Counts: { '5001': 7, undefined: 1 }
- Miner Counts: { '5001': 5, '5004': 2, undefined: 1 }
- Miner Counts: { '5001': 3, '5004': 4, undefined: 1 }

Oczywiście biorąc pod uwagę, że testy tego typu wykonywane na jednej maszynie nie są w pełni miarodajne, wyniki i tak można uznać za wystarczająco dobrze ilustrujące zachowanie implementacji.

Można zaobserwować, że przy wykonaniu ataku DOS skuteczność kopania znacznie spada – w przypadku małych sieci może to zostać wykorzystane do zyskania przewagi obliczeniowej do wykonywania poważniejszych ataków.

3.10 Działanie w zmowie

Do przeprowadzenia testu wykorzystany został scenariusz, gdzie dodatkowo utworzone zostało połączenia od węzła 5004 do 5002, które sprawia, że złe intencje 5003 nie odcinają już węzła 5004 od sieci. Następnie symulacja zmowy odbyła się poprzez dodanie logiki w funkcji wysyłającej wiadomości

przez złośliwe węzły, która blokowała wszystkie treści przeznaczone dla 5004. Skutki testu po wysłaniu przykładowej transakcji (rozpoczętej w węźle 5001):

Łańcuch węzła 5000 (taki sam na 5000, …, 5003)

Łańcuch węzła 5004

Zaistniała sytuacja potwierdza, że atak przez zmowę jest skuteczny w implementowanej sieci. Nastąpiło całkowite odcięcie węzła 5004 od sieci na skutek niechcianej kooperacji węzłów 5002 oraz 5003. Taki atak mógłby posłużyć do odcięcia konkurenta od sieci lub w celu lepszego ukrycia sytuacji – odcięcia tylko od części bloków/transakcji np. takich które przelewają pieniądze dla odciętego węzła.

Metodą ochrony przed tym atakiem jest sama wielkość sieci – dla dużych sieci ryzyko jest mniejsze, bo naturalnie powstaje więcej połączeń co zwiększa wymagania wobec atakujących. Innym sposobem, dla małych sieci, może być celowe tworzenie nadmiarowych połączeń. W ten sposób ochrona węzła jest zwiększona kosztem nadmiarowości komunikacji.

Tak naprawdę, z powodu sekwencyjnego charakteru użytej sieci testowej wszystkie ataki opisane w sekcjach na temat przewagi obliczeniowej jednego węzła można interpretować jako przewagę grupy węzłów w sieciach z wieloma połączeniami.