O'REILLY®

Ethereum

dla zaawansowanych

TWORZENIE INTELIGENTNYCH KONTRAKTÓW
I APLIKACJI ZDECENTRALIZOWANYCH

Andreas M. Antonopoulos
Dr. Gavin Wood

Helion

Tytuł oryginału: Mastering Ethereum: Building Smart Contracts and Dapps

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-283-5574-3

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/ethzaa
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 2.

Podstawy Ethereum

W tym rozdziale rozpoczniemy poznawanie Ethereum. Dowiesz się, jak używać portfeli, jak tworzyć transakcje, a także jak uruchomić prosty inteligentny kontrakt.

Jednostki waluty ether

Jednostka waluty w Ethereum to *ether*. Inne stosowane nazwy to ETH, symbol Ξ (jest to grecka litera ksi; przypomina ona stylizowaną wielką literę E) lub, rzadziej, ♦. Oto przykłady: 1 ether, 1 ETH, Ξ1 lub ♦1.

Symbolowi Ξ odpowiada kod Unicode U+039E, a symbolowi ♦ — kod Unicode U+2666.

Ether dzieli się na mniejsze jednostki, aż do najmniejszej, którą jest *wei*. Jeden ether to trylion wei (10^{18}, czyli 1 000 000 000 000 000 000). Możesz usłyszeć, jak niektóre osoby nazywają walutę „Ethereum", jest to jednak częsty błąd nowicjuszy. Ethereum to system, natomiast walutą jest ether.

W Ethereum liczba etherów jest reprezentowana wewnętrznie zawsze jako liczba całkowita bez znaku reprezentująca wei. Gdy przesyłasz jeden ether, w transakcji jako wartość kodowany jest 1000000000000000000 wei.

Poszczególne jednostki waluty ether mają *nazwę naukową* zgodną z międzynarodowym układem jednostek miar *SI*, a także nazwę potoczną, honorującą liczne wybitne umysły ze świata informatyki i kryptografii.

W tabeli 2.1 wymieniono różne jednostki wraz z nazwami potocznymi i naukowymi. Aby zachować zgodność z wewnętrzną reprezentacją kwot, w tabeli wszystkie wartości są podane w wei (pierwszy wiersz), a ether przedstawiono jako 10^{18} wei (siódmy wiersz).

Tabela 2.1. Jednostki waluty ether i ich nazwy

Wartość (w wei)	Wykładnik	Nazwa potoczna	Nazwa w układzie SI
1		wei	Wei
1000	10^3	Babbage	Kilowei lub femtoether
1 000 000	10^6	Lovelace	Megawei lub picoether
1 000 000 000	10^9	Shannon	Gigawei lub nanoether
1 000 000 000 000	10^{12}	Szabo	Microether lub micro
1 000 000 000 000 000	10^{15}	Finney	Milliether lub milli
1 000 000 000 000 000 000	*10^{18}*	*Ether*	*Ether*
1 000 000 000 000 000 000 000	10^{21}	Grand	Kiloether
1 000 000 000 000 000 000 000 000	10^{24}		Megaether

Wybieranie portfela Ethereum

Pojęcie „portfel” może oznaczać wiele rzeczy, przy czym wszystkie te znaczenia są powiązane i w praktyce sprowadzają się do tego samego. Tu „portfel” oznacza aplikację pomagającą w zarządzaniu kontem Ethereum. Krótko mówiąc: portfel Ethereum to brama do systemu Ethereum. Zawiera klucze i pozwala tworzyć oraz rozsyłać transakcje w imieniu właściciela. Wybieranie portfela Ethereum bywa trudne, ponieważ dostępnych jest wiele różnych narzędzi o zróżnicowanych funkcjach i projektach. Niektóre portfele są lepsze dla początkujących, inne dla ekspertów. Sama platforma Ethereum wciąż jest usprawniana, a „najlepsze” portfele to zwykle te, które dostosowują się do zmian w platformie.

Nie martw się jednak. Jeśli po wybraniu portfela okaże się, że nie w pełni Ci on odpowiada lub że chcesz wypróbować inne narzędzie, możesz stosunkowo łatwo go zmienić. Wystarczy przeprowadzić transakcję przekazującą środki ze starego portfela do nowego lub wyeksportować klucze prywatne i zaimportować je w nowym portfelu.

W tej książce w przykładach używane będą trzy różne typy portfeli: mobilny, desktopowy i internetowy. Wybraliśmy te trzy rodzaje, ponieważ reprezentują one zróżnicowany poziom złożoności i funkcji. Nie wybraliśmy ich z powodu wysokiej jakości lub solidnych zabezpieczeń. Są one po prostu dobrymi narzędziami do testów i demonstracji rozwiązań.

Warto pamiętać, że aby portfel działał, musi mieć dostęp do kluczy prywatnych. Dlatego niezbędne jest, by pobierać i stosować wyłącznie portfele z zaufanych źródeł. Na szczęście zwykle jest tak, że im bardziej popularny jest dany portfel, tym większe można mieć do niego zaufanie. Jednak dobrą praktyką jest unikanie „wkładania wszystkich jajek do jednego koszyka”. Lepiej jest rozdzielić konto Ethereum między kilka portfeli.

Oto kilka dobrych portfeli na początek:

MetaMask
: MetaMask to portfel będący rozszerzeniem przeglądarki. Działa on w przeglądarkach takich jak Chrome, Firefox, Opera i Brave Browser. Jest łatwy w użyciu i wygodny do wykonywania testów, ponieważ potrafi łączyć się z różnymi węzłami Ethereum i testowymi łańcuchami bloków. MetaMask jest portfelem internetowym.

Jaxx
: Jaxx to portfel obsługujący wiele walut i platform. Działa w różnych systemach operacyjnych, takich jak Android, iOS, Windows, macOS i Linux. Często okazuje się dobrym wyborem dla początkujących, ponieważ został zaprojektowany z myślą o prostocie i łatwości użytkowania. Jaxx może być portfelem mobilnym i desktopowym (w zależności od tego, gdzie został zainstalowany).

MyEtherWallet (MEW)
: MyEtherWallet to internetowy portfel działający w przeglądarce. Udostępnia wiele zaawansowanych funkcji, które poznasz w licznych przykładach z tej książki. MyEtherWallet jest portfelem internetowym.

Emerald Wallet
: Emerald Wallet zaprojektowano na potrzeby łańcucha bloków Ethereum Classic, jednak obsługuje także inne łańcuchy bloków oparte na Ethereum. Jest to aplikacja desktopowa o otwartym dostępie do kodu źródłowego. Działa w systemach Windows, macOS i Linux. Emerald Wallet może uruchamiać pełny węzeł lub łączyć się ze zdalnym węzłem publicznym i pracować w trybie „lekkim". Dostępne jest też powiązane narzędzie, które pozwala wykonywać wszystkie operacje w wierszu poleceń.

Zaczniemy od zainstalowania portfela MetaMask na komputerze stacjonarnym. Najpierw jednak pokrótce omówimy kontrolowanie kluczy i zarządzanie nimi.

Kontrola i odpowiedzialność

Otwarte łańcuchy bloków takie jak Ethereum są ważne, ponieważ działają jako system **zdecentralizowany**. Wynika z tego wiele rzeczy, a jednym z bardzo ważnych aspektów jest to, że każdy użytkownik Ethereum może (i powinien) kontrolować własne klucze prywatne, ponieważ decydują one o dostępie do środków i inteligentnych kontraktów. Mechanizm dający dostęp do środków i inteligentnych kontraktów czasem nazywany jest „kontem" lub „portfelem". Te pojęcia są w tym kontekście dość skomplikowane; szczegółowo opiszemy je dalej. Na razie zapamiętaj podstawową zasadę, zgodnie z którą jeden klucz prywatny równa się jednemu „kontu". Niektórzy użytkownicy rezygnują z kontroli nad własnymi kluczami prywatnymi i korzystają z usług niezależnego nadzorcy, np. giełdy internetowej. Z tej książki dowiesz się, jak przejąć kontrolę nad własnymi kluczami prywatnymi i jak nimi zarządzać.

Z kontrolą związana jest duża odpowiedzialność. Jeśli utracisz klucze prywatne, nie będziesz miał dostępu do środków i kontraktów. Nikt nie będzie mógł Ci pomóc w odzyskaniu dostępu — Twoje fundusze zostaną na zawsze zablokowane. Oto kilka wskazówek, które pomogą Ci udźwignąć tę odpowiedzialność:

- Nie eksperymentuj z bezpieczeństwem. Stosuj sprawdzone standardowe rozwiązania.
- Im ważniejsze konto (im wyższa wartość kontrolowanych środków lub istotniejszy dostępny inteligentny kontrakt), tym ściślejsze środki bezpieczeństwa należy zastosować.
- Najwyższy poziom bezpieczeństwa zapewniają urządzenia niepodłączone do sieci, jednak nie każde konto wymaga takich zabezpieczeń.

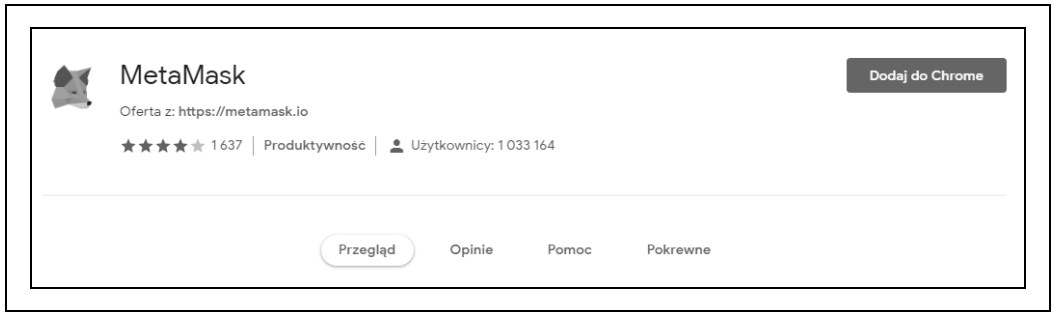
- Nigdy nie przechowuj kluczy prywatnej w czytelnej formie, zwłaszcza cyfrowo. Na szczęście większość dostępnych obecnie interfejsów użytkownika nie pozwoli Ci nawet zobaczyć kluczy prywatnych w nieprzetworzonej formie.
- Klucze prywatne można przechowywać w zaszyfrowanej postaci, takiej jak cyfrowy plik *keystore*. Ponieważ taki klucz jest zaszyfrowany, do jego odblokowania niezbędne jest hasło. Zadbaj o to, by było ono mocne (czyli długie i losowe), zarchiwizuj je i nie udostępniaj go. Jeśli nie używasz menedżera haseł, zapisz je i przechowuj w bezpiecznym, tajnym miejscu. Aby uzyskać dostęp do konta, potrzebny będzie plik *keystore* i hasło.
- Nie przechowuj haseł w dokumentach cyfrowych, na zdjęciach cyfrowych, zrzutach ekranu, dyskach sieciowych, w zaszyfrowanych plikach PDF itd. Nie eksperymentuj z bezpieczeństwem. Korzystaj z menedżera haseł lub z kartki i długopisu.
- Gdy chcesz zachować klucz jako mnemoniczną sekwencję słów, utwórz kopię fizyczną za pomocą kartki i długopisu. Nie odkładaj tego zadania „na później", ponieważ możesz zapomnieć hasło. Taką kopię fizyczną możesz zastosować do odzyskania klucza prywatnego, gdy utracisz wszystkie dane zapisane w systemie lub zapomnisz hasło. Taka kopia może jednak zostać wykorzystana przez napastników do uzyskania Twoich kluczy prywatnych, dlatego nigdy nie zapisuj jej w formie cyfrowej. Ponadto przechowuj fizyczną kopię hasła schowaną bezpiecznie w zamkniętej szufladzie lub w sejfie.
- Przed przelewem dużej kwoty (zwłaszcza na nowy adres) najpierw wykonaj małą transakcję testową, np. o wartości złotówki, i poczekaj na jej potwierdzenie.
- Gdy tworzysz nowe konto, zacznij od przesłania na nowy adres tylko niewielkiej transakcji testowej. Gdy otrzymasz środki, spróbuj odesłać je z nowego konta. Jest wiele przyczyn, dla których tworzenie konta może się nie powieść. Lepiej dowiedzieć się o takim niepowodzeniu, przelewając niewielką kwotę. Jeśli testy zakończą się sukcesem, konto zostało poprawnie utworzone.
- Publiczne eksploratory bloków to łatwe narzędzie do samodzielnego sprawdzenia, czy transakcja została zaakceptowana w sieci. Ta wygoda ma jednak negatywny wpływ na prywatność, ponieważ ujawniasz swoje adresy eksploratorom bloków, co grozi tym, że będziesz śledzony.
- Nie przesyłaj środków na żaden z adresów podanych w tej książce. Klucze prywatne do tych adresów są podane w książce i ktoś natychmiast może przejąć Twoje pieniądze.

Po omówieniu podstawowych praktyk zarządzania kluczami i dbania o bezpieczeństwo pora przejść do używania portfela MetaMask!

Rozpoczynanie pracy z portfelem MetaMask

Otwórz przeglądarkę Google Chrome i przejdź pod adres *https://chrome.google.com/webstore/category/extensions*.

Poszukaj narzędzia MetaMask i kliknij logo z liskiem. Powinieneś zobaczyć informacje podobne do tych z rysunku 2.1.



Rysunek 2.1. *Strona szczegółów rozszerzenia MetaMask Chrom*

Ważne jest, abyś się upewnił, że pobrałeś prawdziwe rozszerzenie MetaMask. Czasem napastnikom udaje się „przemycić" szkodliwe rozszerzenie przez filtry Google'a. Oto dane prawdziwego rozszerzenia:

- ma identyfikator `nkbihfbeogaeaoehlefnkodbefgpgknn` na pasku adresu;
- jest udostępniane z witryny *https://metamask.io*;
- ma ponad 1400 recenzji;
- ma ponad 1 000 000 użytkowników.

Gdy się upewnisz, że wybrałeś poprawne rozszerzenie, kliknij przycisk *Dodaj do Chrome*, aby je zainstalować.

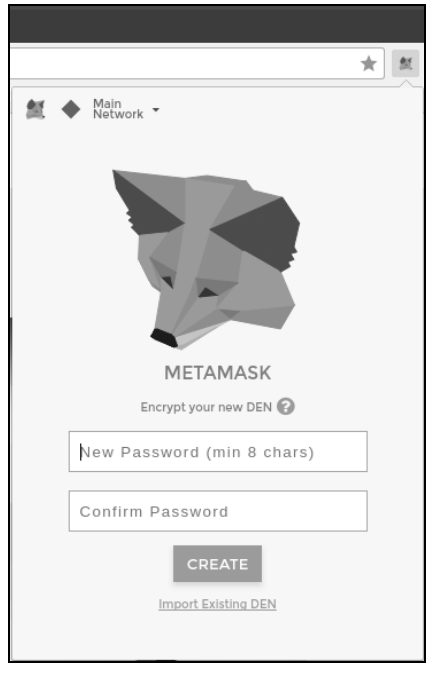
Tworzenie portfela

Po zainstalowaniu rozszerzenia MetaMask powinieneś zobaczyć nową ikonę (z głową liska) na pasku narzędzi przeglądarki. Kliknij tę ikonę, aby rozpocząć pracę. Pojawi się prośba o zaakceptowanie zasad i warunków. Następnie będziesz mógł utworzyć nowy portfel Ethereum. W tym celu wpisz hasło (zob. rysunek 2.2).

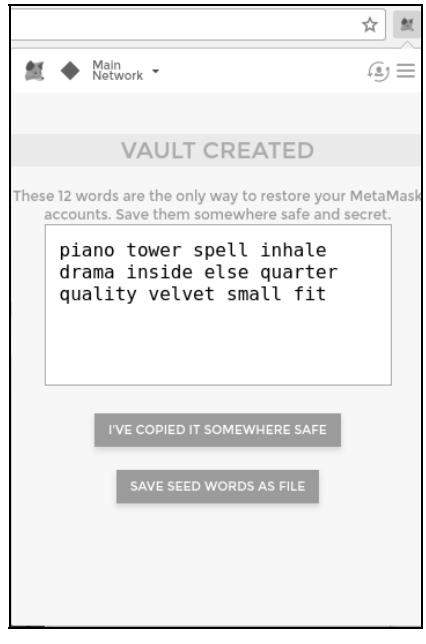
Dostęp do rozszerzenia MetaMask jest zabezpieczony hasłem, dlatego portfel nie może być stosowany przez dowolnego użytkownika przeglądarki.

Po podaniu hasła MetaMask wygeneruje portfel i wyświetli **mnemoniczną kopię zapasową**, składającą się z 12 angielskich słów (zob. rysunek 2.3). Te słowa możesz zastosować w dowolnym zgodnym portfelu, aby odzyskać dostęp do środków, jeśli coś stanie się z rozszerzeniem MetaMask na Twoim komputerze. Do odzyskania konta nie będziesz potrzebował hasła — wystarczy 12 wygenerowanych słów.

Zapisz mnemoniczną kopię (12 słów) na kartce — i to dwukrotnie. Umieść obie kartki w dwóch odrębnych bezpiecznych miejscach, np. w ognioodpornym sejfie, zamykanej szufladzie lub skrytce depozytowej. Traktuj te kartki jak kwotę w gotówce odpowiadającą wartości środków w Twoim portfelu Ethereum. Każdy, kto uzyska dostęp do tych słów, będzie mógł ukraść Twoje pieniądze.

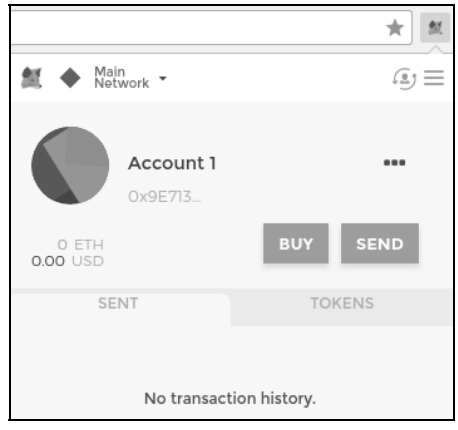


Rysunek 2.2. *Strona na hasło w rozszerzeniu MetaMask dla przeglądarki Chrome*



Rysunek 2.3. *Mnemoniczna kopia zapasowa portfela wygenerowana przez rozszerzenie MetaMask*

Po potwierdzeniu, że bezpiecznie zapisałeś mnemoniczną kopię zapasową, zobaczysz szczegółowe informacje na temat swojego konta Ethereum. Ilustruje to rysunek 2.4.



Rysunek 2.4. *Konto Ethereum w rozszerzeniu MetaMask*

Strona konta wyświetla jego nazwę (domyślnie jest to „Account 1”), adres Ethereum (w przykładzie jest to `0x9E713...`) i kolorową ikonę, która pomaga odróżnić wizualnie dane konto od innych. W górnej części strony konta możesz zobaczyć, z jakiej sieci Ethereum aktualnie korzystasz (w przykładzie jest to sieć „Main Network”).

Gratulacje! Utworzyłeś swój pierwszy portfel Ethereum.

Zmienianie sieci

Na stronie konta w rozszerzeniu MetaMask widać, że możesz wybrać jedną z kilku sieci Ethereum. MetaMask domyślnie próbuje połączyć się z siecią główną (ang. *Main Network*). Inne możliwości to publiczne sieci testowe, dowolny wybrany węzeł Ethereum lub węzły udostępniające prywatne łańcuchy bloków na Twoim komputerze (na hoście lokalnym).

Główna sieć Ethereum
: Jest to główny publiczny łańcuch bloków Ethereum — realne ethery, realna wartość i realne konsekwencje.

Sieć testowa Ropsten
: Jest to publiczny testowy łańcuch bloków i publiczna testowa sieć Ethereum. Ethery z tej sieci nie mają żadnej wartości.

Sieć testowa Kovan
: Jest to publiczny testowy łańcuch bloków i publiczna testowa sieć Ethereum. Używany jest tu protokół osiągania konsensusu Aura z modelem PoF (ang. *proof of authority*) i konsensusem federacyjnym. Ethery z tej sieci nie mają żadnej wartości. Sieć testowa Kovan jest obsługiwana tylko w kliencie Parity. Inne klienty Ethereum do sprawdzania poprawności z użyciem modelu PoF używają zaproponowanego później protokołu osiągania konsensusu Clique.

Sieć testowa Rinkeby
: Jest to publiczny testowy łańcuch bloków i publiczna testowa sieć Ethereum. Używany jest tu protokół osiągania konsensusu Clique z modelem PoF i konsensusem federacyjnym. Ethery z tej sieci nie mają żadnej wartości.

Localhost 8545
: Łączy się z węzłem działającym na tym samym komputerze, na którym pracuje przeglądarka. Ten węzeł może działać w dowolnym publicznym łańcuchu bloków (głównym lub testowym) albo w prywatnej sieci testowej.

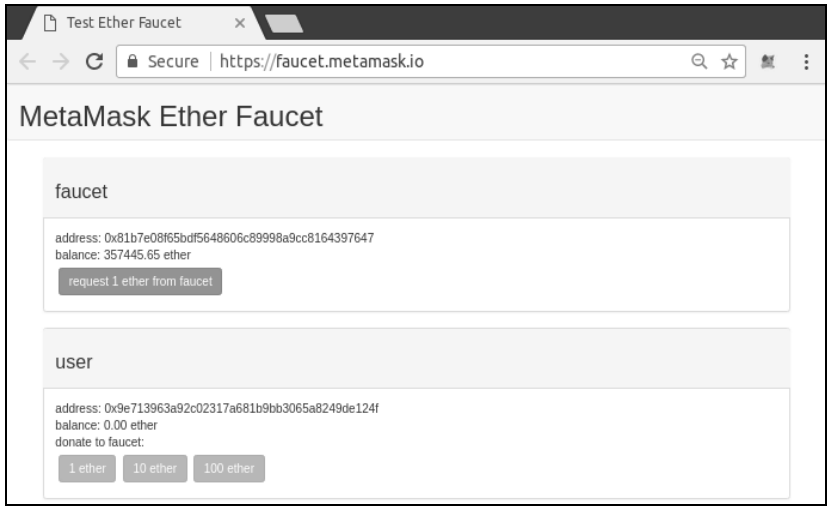
Niestandardowy klient RPC
: Umożliwia podłączenie rozszerzenia MetaMask do dowolnego węzła z interfejsem RPC zgodnym z klientem Geth. Taki węzeł może używać dowolnego publicznego lub prywatnego łańcucha bloków.

W portfelu MetaMask te same klucze prywatne i adresy Ethereum są używane we wszystkich sieciach, z jakimi portfel się łączy. Jednak w każdej z tych sieci stan konta dla adresu Ethereum będzie inny. Używane klucze mogą np. zapewniać dostęp do etherów i kontraktów w sieci Ropsten, ale już nie w sieci głównej.

Zdobywanie testowych etherów

Pierwsze zadanie polega na zapewnieniu środków w portfelu. Nie będziesz używał do tego sieci głównej, ponieważ realne ethery kosztują pieniądze, a zarządzanie nimi wymaga nieco doświadczenia. Na razie zasilisz portfel testowymi etherami.

Przełącz sieć w portfelu MetaMask na *Ropsten Test Network*. Kliknij przycisk *Buy*, a następnie wybierz opcję *Ropsten Test Faucet*. MetaMask otworzy nową stronę, widoczną na rysunku 2.5.



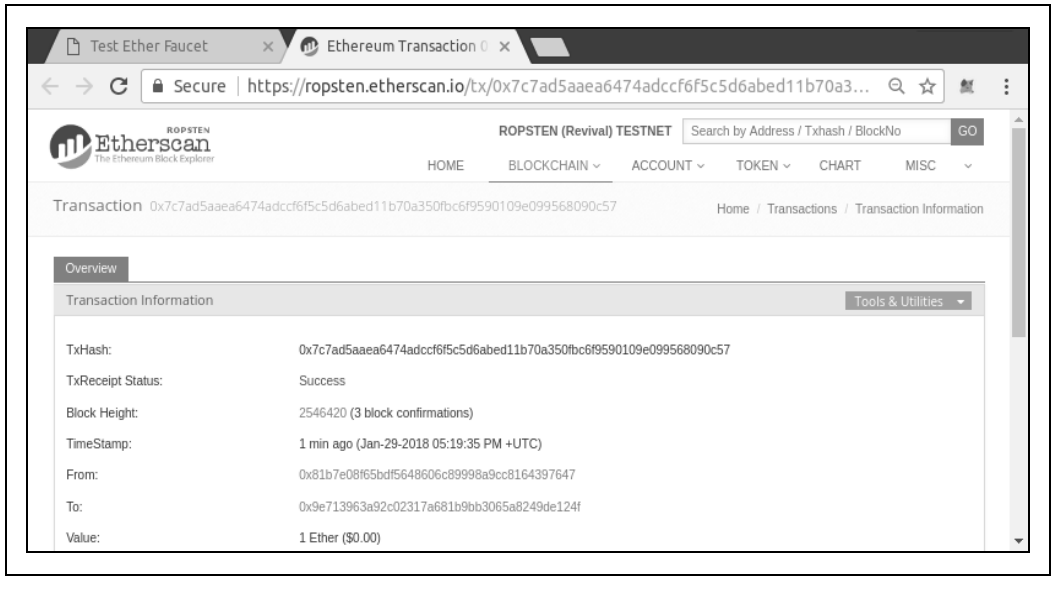
Rysunek 2.5. *Kran z testowymi etherami w sieci Ropsten*

Może zauważyłeś, że na stronie widoczny jest już adres Ethereum z Twojego portfela Ethereum. MetaMask integruje obsługujące Ethereum strony internetowe z portfelem MetaMask i wyświetla adresy Ethereum na takich stronach, co pozwala np. przesłać płatność do sklepu internetowego udostępniającego adres Ethereum. MetaMask może też podać na stronie internetowej Twój adres jako adres odbiorcy, jeśli strona tego zażąda. Na pokazanej stronie kran żąda od portfela MetaMask podania adresu, na który należy przesłać testowe ethery.

Kliknij zielony przycisk *request 1 ether from faucet*. W lewej dolnej części strony pojawi się identyfikator transakcji. Kran utworzył transakcję — płatność na Twoje konto. Oto identyfikator tej transakcji:

```
0x7c7ad5aaea6474adccf6f5c5d6abed11b70a350fbc6f9590109e099568090c57
```

Po kilku sekundach nowa transakcja zostanie wydobyta przez górników z sieci Ropsten, a w Twoim portfelu MetaMask pojawi się stan 1 ETH. Kliknij identyfikator transakcji, a przeglądarka otworzy **eksplorator bloków**, czyli witrynę, która pozwala wizualizować i przeglądać bloki, adresy oraz transakcje. MetaMask używa eksploratora bloków Etherscan (*https://etherscan.io/*). Jest to jeden z popularnych eksploratorów bloków Ethereum. Transakcja zawierająca przelew z kranu Ropsten Test Faucet jest pokazana na rysunku 2.6.



Rysunek 2.6. *Eksplorator bloków Etherscan Ropsten*

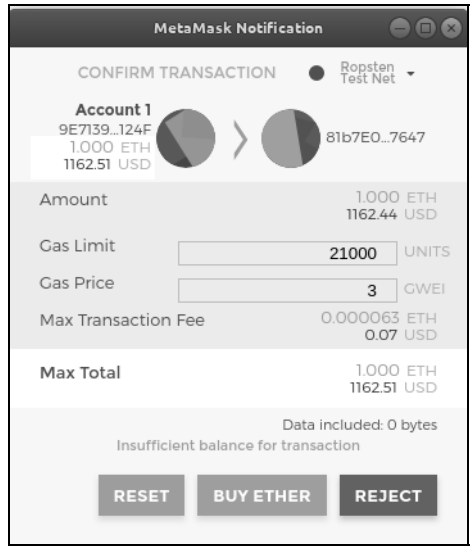
Ta transakcja została zarejestrowana w łańcuchu bloków Ropsten i może zostać wyświetlona w dowolnym momencie przez każdą osobę. Wystarczy poszukać identyfikatora transakcji lub kliknąć odsyłacz *http://bit.ly/2Q860Wk*.

Spróbuj kliknąć ten odsyłacz lub wpisz skrót transakcji w witrynie *http://ropsten.etherscan.io*, aby samemu zobaczyć dane.

Wysyłanie etherów z portfela MetaMask

Po otrzymaniu pierwszego testowego ethera z testowego kranu sieci Ropsten możesz zacząć eksperymenty z przesyłaniem etherów. Najpierw spróbuj przesłać środki z powrotem do kranu. Na stronie testowego kranu sieci Ropsten znajdziesz opcję przekazania 1 ETH do kranu. Ta opcja jest dostępna, abyś po zakończeniu testów mógł zwrócić resztę testowego ethera, co pozwoli wykorzystać te środki następnej osobie. Choć testowe ethery nie mają żadnej wartości, niektóre osoby gromadzą je, co utrudnia innym korzystanie z sieci testowej. Gromadzenie testowych etherów jest niemile widziane!

Na szczęście nie jesteśmy zbieraczami testowych etherów. Kliknij pomarańczowy przycisk *1 ether*, aby nakazać portfelowi MetaMask utworzenie transakcji przekazującej 1 ether do kranu. MetaMask przygotuje wtedy transakcję i wyświetli okno z prośbą o zatwierdzenie jej (rysunek 2.7).



Rysunek 2.7. *Przesyłanie 1 ethera do kranu*

Ups! Prawdopodobnie zauważyłeś, że nie możesz ukończyć transakcji. MetaMask informuje, że stan konta jest niewystarczający. Na pozór jest to dziwne. Masz 1 ETH i chcesz przesłać 1 ETH. Dlaczego więc MetaMask komunikuje, że nie masz wystarczających środków?

Powodem są koszty **paliwa**. Każda transakcja w Ethereum wymaga poniesienia opłat przekazywanych górnikom za sprawdzanie poprawności transakcji. W Ethereum opłaty są ponoszone w wirtualnej walucie nazywanej paliwem. Za paliwo płaci się etherami w ramach transakcji.

Opłaty są ponoszone także w sieciach testowych. Bez opłat sieć testowa działałaby inaczej od sieci głównej, przez co nie nadawałaby się na platformę testową. Ponadto opłaty chronią sieci testowe — tak samo jak i sieć główną — przed atakami DoS i źle skonstruowanymi kontraktami (np. z pętlami nieskończonymi).

Gdy przesyłaliśmy transakcję, MetaMask wyznaczył średnią cenę paliwa z ostatnich udanych transakcji na 3 gwei (czyli gigawei). Wei to najmniejsza jednostka waluty ether, co opisano w punkcie „Jednostki waluty ether". Limit zużycia paliwa został tu ustawiony na koszt przesłania prostej transakcji i wynosi 21 000 jednostek paliwa. Tak więc maksymalny koszt w etherach wynosi 3 · 21 000 gwei = 63 000 gwei = 0,000063 ETH. Pamiętaj, że ceny paliwa fluktuują, ponieważ są ustalane przede wszystkim przez górników. W dalszych rozdziałach zobaczysz, jak zwiększyć lub zmniejszyć limit zużycia paliwa, aby w razie potrzeby zagwarantować pierwszeństwo wykonania transakcji.

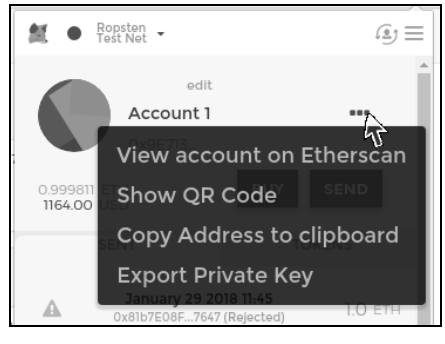
W podsumowaniu można napisać, że transakcja przelewu 1 ETH kosztuje 1,000063 ETH. MetaMask w mylący sposób zaokrągla tę wartość *w dół* do 1 ETH, gdy wyświetla podsumowanie, jednak potrzebna kwota to 1,000063 ETH, a dostępny jest tylko 1 ETH. Kliknij przycisk *Reject*, aby anulować transakcję.

Pobierz więc więcej testowych etherów! Ponownie kliknij zielony przycisk *Request 1 ether from the faucet* i odczekaj kilka sekund. Nie martw się, kran udostępnia dużo etherów przekaże Ci ich więcej, jeśli tego zażądasz.

Gdy stan konta wynosi 2 ETH, możesz ponowić próbę. Jeśli tym razem klikniesz pomarańczowy przycisk *1 ether* w celu przekazania środków, będziesz miał wystarczającą ilość środków na przeprowadzenie transakcji. Kiedy MetaMask wyświetli okno zatwierdzania płatności, kliknij przycisk *Submit*. Po tej operacji stan konta powinien wynosić 0,999937, ponieważ przesłałeś do kranu 1 ETH i poniosłeś koszty paliwa w wysokości 0,000063 ETH.

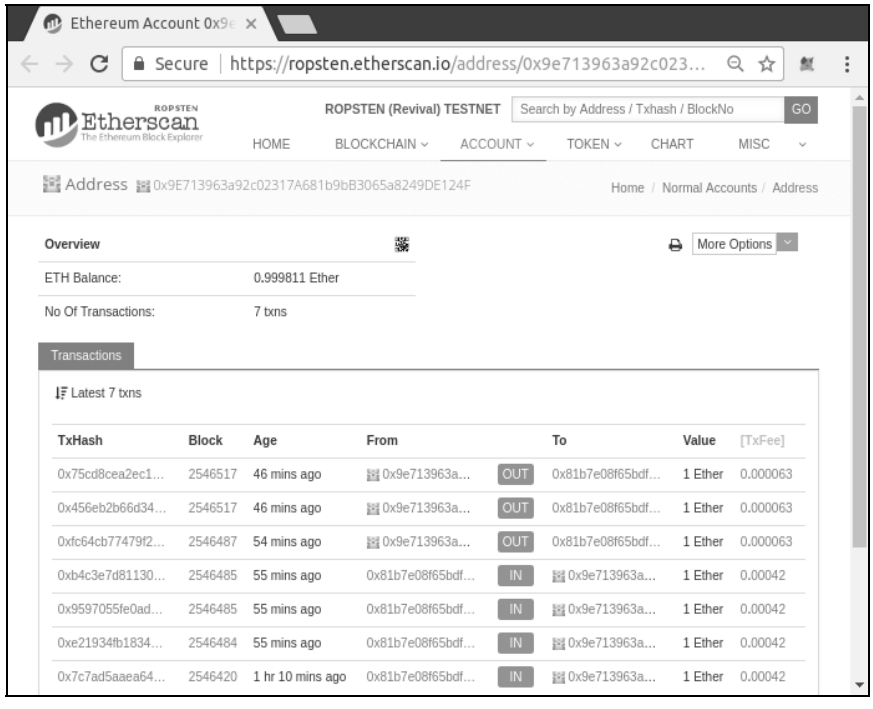
Przeglądanie historii transakcji dla adresu

Do tej pory stałeś się już ekspertem w używaniu portfela MetaMask do wysyłania i otrzymywania testowych etherów. Otrzymałeś już przynajmniej dwie płatności i wysłałeś co najmniej jedną. Wszystkie te transakcje możesz przejrzeć, używając eksploratora bloków *http://ropsten.etherscan.io*. Możesz albo skopiować adres portfela i wkleić go w polu wyszukiwania eksploratora bloków, albo otworzyć stronę za pośrednictwem portfela MetaMask. Obok ikony konta w portfelu MetaMask znajduje się przycisk z trzema kropkami. Kliknij go, aby wyświetlić menu opcji powiązanych z kontem (rysunek 2.8).



Rysunek 2.8. *Menu kontekstowe konta w portfelu MetaMask*

Wybierz opcję *View account on Etherscan*, aby otworzyć w eksploratorze bloków stronę internetową wyświetlającą historię transakcji na Twoim koncie (rysunek 2.9).



Rysunek 2.9. *Historia transakcji dla adresu w eksploratorze Etherscan*

W tym miejscu możesz zobaczyć całą historię operacji dla adresu Ethereum. Widoczne są tu wszystkie zarejestrowane w łańcuchu bloków Ropsten transakcje, w których Twój adres jest używany jako adres nadawcy lub odbiorcy. Kliknij kilka z tych transakcji, aby zobaczyć dodatkowe szczegóły.

Możesz zbadać historię transakcji dla dowolnego adresu. Przyjrzyj się historii transakcji dla adresu testowego kranu sieci Ropsten (wskazówka: jest to adres nadawcy w najstarszej płatności dla Twojego adresu). Zobaczysz wszystkie operacje przesłania testowych etherów z kranu do Ciebie i na inne adresy. Każda transakcja może prowadzić do kolejnych adresów i transakcji. Szybko zgubisz się w labiryncie powiązanych ze sobą danych. Publiczne łańcuchy bloków zawierają olbrzymią ilość informacji. Wszystkie te dane można analizować programowo, o czym przekonasz się dzięki lekturze dalszych rozdziałów.

Wprowadzenie do światowego komputera

Utworzyłeś już portfel oraz przesłałeś i otrzymałeś ethery. Do tej pory traktowaliśmy Ethereum jak kryptowalutę. Jednak Ethereum to coś o wiele więcej. Obsługa kryptowaluty jest drugorzędna względem podstawowej funkcji Ethereum, jaką jest udostępnianie zdecentralizowanego świato-

wego komputera. Ether ma służyć do opłaty za wykonywanie **inteligentnych kontraktów**, czyli programów komputerowych działających w emulowanym komputerze o nazwie *Ethereum Virtual Machine* (EVM).

Maszyna EVM jest singletonem w skali globalnej. Oznacza to, że działa tak, jakby była pojedynczym globalnym komputerem działającym wszędzie. Każdy węzeł w sieci Ethereum uruchamia lokalną kopię maszyny EVM, aby sprawdzać poprawność wykonywania kontraktu. Łańcuch bloków Ethereum rejestruje zmiany *stanu* światowego komputera, gdy ten przetwarza transakcje i inteligentne kontrakty. Te zagadnienia opisano szczegółowo w rozdziale 13.

Konta EOA i konta kontraktów

Portfel MetaMask tworzy konta typu EOA (ang. *externally owned account*). Takie konta charakteryzują się posiadaniem kluczy prywatnych. Klucze prywatne zapewniają kontrolę nad środkami i kontraktami. Prawdopodobnie domyślasz się już, że istnieje też inny typ kont. Są to **konta kontraktów**. Konto kontraktu to kod inteligentnego kontraktu, którego proste konta EOA nie mogą zawierać. Konto kontraktu nie ma też klucza prywatnego. Zamiast tego jego „właścicielem" (i zarządcą) jest logika kodu inteligentnego kontraktu — programu zapisanego w łańcuchu bloków Ethereum w momencie tworzenia konta kontraktu i wykonywanego przez maszynę EVM.

Kontrakty, podobnie jak konta EOA, mają adresy. Kontrakty, też tak jak konta EOA, mogą wysyłać i przyjmować ethery. Jednak gdy docelowym adresem transakcji jest adres kontraktu, powoduje to *uruchomienie* kontraktu w maszynie EVM z użyciem określonej transakcji i jej danych jako danych wejściowych. Obok etherów transakcje mogą też obejmować *dane* określające, którą funkcję z kontraktu należy uruchomić i jakie parametry do niej przekazać. W ten sposób transakcje mogą *wywoływać* funkcje w kontraktach.

Warto zauważyć, że ponieważ konto kontraktu nie ma klucza prywatnego, nie może *zainicjować* transakcji. Tylko konta EOA mogą inicjować transakcje. Kontrakty mogą jednak *reagować* na transakcje, wywołując inne kontrakty. Powstają w ten sposób złożone ścieżki wykonywania kodu. Jednym z typowych zastosowań tej techniki jest wysyłanie przez kontrakt EOA żądania transakcji do portfela z obsługą inteligentnych kontraktów i wielopodpisu, aby przesłać ethery na inny adres. Typowy wzorzec programowania aplikacji DApp polega na tym, że kontrakt A kieruje wywołanie do kontraktu B, aby zachować wspólny stan wśród użytkowników kontraktu A.

W kilku następnych punktach zobaczysz, jak napisać pierwszy kontrakt. Następnie dowiesz się, jak utworzyć, zasilić i zastosować ten kontrakt za pomocą portfela MetaMask i testowych etherów w sieci testowej Ropsten.

Prosty kontrakt — kran z testowymi etherami

Ethereum obsługuje wiele różnych wysokopoziomowych języków. Wszystkie te języki można stosować do pisania kontraktów i tworzenia kodu bajtowego dla maszyny EVM. O wybranych najważniejszych i najciekawszych językach przeczytasz w punkcie „Wprowadzenie do wysokopoziomowych języków dostępnych w Ethereum". Do programowania inteligentnych kontraktów zdecydowanie

najczęściej używany jest jeden z wysokopoziomowych języków — Solidity. Został on opracowany przez dr. Gavina Wooda, współautora tej książki, i stał się najczęściej używanym językiem w Ethereum (i nie tylko tu). Do napisania pierwszego kontraktu posłużysz się właśnie tym językiem.

W pierwszym przykładzie (listing 2.1) napiszesz kontrakt kontrolujący **kran**. Używałeś już kranu, aby otrzymać testowe ethery w sieci Ropsten. Kran to stosunkowo prosty mechanizm — przekazuje ethery na podany adres i może być okresowo uzupełniany środkami. Kran możesz zaimplementować jako portfel kontrolowany przez człowieka lub serwer WWW.

Listing 2.1. *Plik Faucet.sol — napisany w Solidity kontrakt działający jak kran*

```
1  // Pierwszy kontrakt działa jak kran.
2  contract Faucet {
3
4    // Przekazuje ethery na dowolny wskazany adres.
5    function withdraw(uint withdraw_amount) public {
6
7      // Ograniczanie żądanej kwoty.
8      require(withdraw_amount <= 100000000000000000);
9
10     // Przekazywanie środków na wskazany adres.
11     msg.sender.transfer(withdraw_amount);
12   }
13
14   // Przyjmuje dowolną przesyłaną kwotę.
15   function () public payable {}
16
17 }
```

Wszystkie przykładowe programy do tej książki znajdziesz w podkatalogu *code* w repozytorium GitHub powiązanym z tą pozycją (*https://github.com/ethereumbook/ethereumbook/*). Kontrakt *Faucet.sol* znajdziesz w katalogu *code/Solidity/Faucet.sol*. Spolszczona wersja kodu jest dostępna w witrynie wydawnictwa Helion.

Jest to bardzo prosty kontrakt — prawie tak prosty, jak to możliwe. Ma on pewne *wady*, aby zilustrować niezalecane praktyki i luki bezpieczeństwa. W dalszych punktach będziesz się uczył, analizując błędy z tego programu. Na razie przyjrzyj się wiersz po wierszu temu, co ten kontrakt robi i jak działa. Szybko zauważysz, że liczne elementy języka Solidity są podobne jak w innych językach programowania, takich jak JavaScript, Java i C++.

Pierwszy wiersz to komentarz:

```
// Pierwszy kontrakt działa jak kran.
```

Komentarze są przeznaczone dla ludzi i nie znajdują się w wykonywalnym kodzie bajtowym dla maszyny EVM. Zwykle umieszcza się je w wierszu przed objaśnianym kodem (a czasem w tym samym wierszu). Komentarze rozpoczynają się od dwóch ukośników — //. Wszystko od pierwszego ukośnika to końca wiersza jest traktowane jak pusty wiersz i pomijane.

W następnym wierszu rozpoczyna się właściwy kontrakt:

```
contract Faucet {
```

W tym wierszu zadeklarowany jest kontrakt (`contract`); przypomina on klasy (deklaracje `class`) z innych języków obiektowych. Definicja kontraktu obejmuje wszystkie wiersze między nawiasami klamrowymi {}, definiującymi **zasięg** (podobną funkcję nawiasy klamrowe pełnią także w wielu innych językach programowania).

Dalej zadeklarowana jest pierwsza funkcja kontraktu `Faucet`:

```
function withdraw(uint withdraw_amount) public {
```

Ta funkcja nosi nazwę `withdraw` i przyjmuje jeden argument, `withdraw_amount`, typu `uint` (jest to liczba całkowita bez znaku). Omawiana funkcja jest zadeklarowana jako publiczna, co oznacza, że może być wywoływana przez inne kontrakty. Dalej, w nawiasach klamrowych, znajduje się definicja tej funkcji. Pierwszy fragment funkcji `withdraw` określa limit wypłat:

```
require(withdraw_amount <= 100000000000000000);
```

Używana jest tu wbudowana funkcja Solidity `require`, aby sprawdzać warunek, zgodnie z którym parametr `withdraw_amount` ma wartość nie większą niż 100 000 000 000 000 000 wei (wei to podstawowa jednostka waluty ether; zob. tabelę 2.1), czyli 0,1 ethera. Jeśli funkcja `withdraw` zostanie wywołana z parametrem `withdraw_amount` przekraczającym podaną wartość, funkcja `require` zatrzyma wykonywanie kontraktu i zgłosi *wyjątek*. Warto zauważyć, że w Solidity instrukcje muszą kończyć się średnikiem.

Ta część kontraktu określa logikę działania kranu. Ten kod kontroluje przekazywanie środków przez kontrakt, ponieważ nakłada limit na wypłaty. Jest to bardzo prosty mechanizm, jednak daje przedsmak możliwości programowalnych łańcuchów bloków w zakresie tworzenia zdecentralizowanego oprogramowania do kontroli pieniędzy.

Dalej znajduje się kod wypłacający środki:

```
msg.sender.transfer(withdraw_amount);
```

Dzieje się tu kilka ciekawych rzeczy. Obiekt `msg` to jeden z obiektów dostępnych we wszystkich kontraktach. Reprezentuje on transakcję, która uruchomiła wykonywanie danego kontraktu. Atrybut `sender` to adres nadawcy transakcji. Funkcja `transfer` to wbudowana funkcja przekazująca ethery z bieżącego kontraktu na adres nadawcy. Przeczytajmy tę instrukcję od końca: oznacza ona przesyłanie (`transfer`) środków do nadawcy (`sender`) komunikatu (`msg`), który uruchomił wykonywanie kontraktu. Funkcja `transfer` przyjmuje jeden argument — kwotę. Jako ten argument przekazywana jest wartość parametru `withdraw_amount` z zadeklarowanej kilka wierszy wcześniej funkcji `withdraw`.

Kolejny wiersz to zamykający nawias klamrowy. Oznacza on koniec definicji funkcji `withdraw`.

Dalej zadeklarowana jest kolejna funkcja:

```
function () public payable {}
```

Jest to funkcja **rezerwowa** lub **domyślna**. Jest ona wywoływana, jeśli w transakcji, która uruchomiła kontrakt, albo nie podano nazwy żadnej funkcji zadeklarowanej w kontrakcie, albo w ogóle nie wywołano żadnej funkcji, albo w ogóle nie umieszczono danych. Kontrakty mogą mieć jedną funkcję domyślną (bez nazwy) i zwykle służy ona do przyjmowania etherów. To dlatego została zdefiniowana jako publiczna i z obsługą płatności (specyfikator `payable`), co oznacza, że kontrakt

może pobierać ethery. Funkcja ta nie robi nic oprócz przyjmowania etherów. Wskazuje na to pusta definicja w nawiasach klamrowych. Jeśli utworzysz transakcję, która przesyła ethery na adres kontraktu tak, jakby był to portfel, ta funkcja obsłuży tę operację.

Pod funkcją domyślną znajduje się ostatni zamykający nawias klamrowy. Kończy on definicję kontraktu `Faucet`. To wszystko!

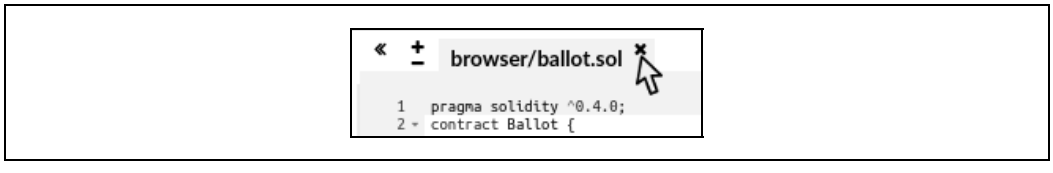
Kompilowanie kontraktu Faucet

Po utworzeniu pierwszego przykładowego kontraktu należy posłużyć się kompilatorem języka Solidity, aby przekształcić kod w języku Solidity na kod bajtowy maszyny EVM, aby można go wykonywać w maszynie EVM w łańcuchu bloków.

Kompilator języka Solidity to niezależny plik wykonywalny dostępny w różnych platformach i środowiskach IDE. Aby uprościć pracę, użyjemy jednego z popularnych środowisk IDE — **Remix**.

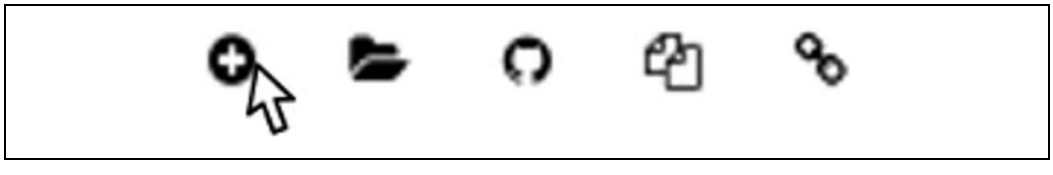
W przeglądarce Chrome (z zainstalowanym wcześniej portfelem MetaMask) przejdź do środowiska IDE Remix na stronie *https://remix.ethereum.org*.

Kiedy po raz pierwszy otworzysz środowisko Remix, widoczny będzie przykładowy kontrakt *ballot.sol*. Nie jest on potrzebny, dlatego możesz go zamknąć. Kliknij w tym celu symbol „x” w rogu zakładki (rysunek 2.10).



Rysunek 2.10. *Zamykanie zakładki z domyślnym przykładowym kodem*

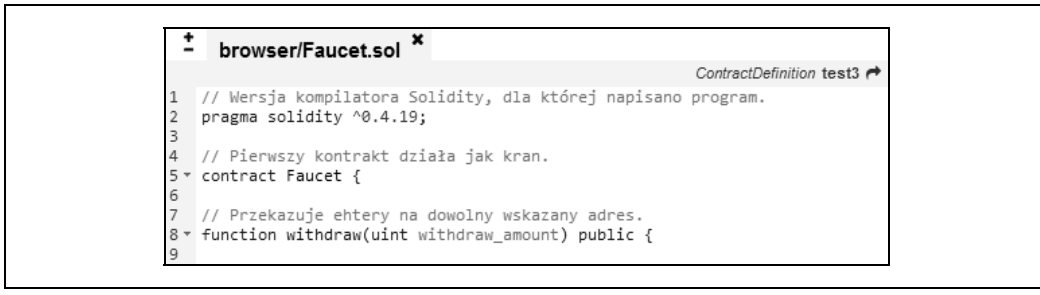
Następnie dodaj nową zakładkę. W tym celu kliknij okrągły znak plus w pasku narzędzi w lewym górnym rogu (rysunek 2.11). Nowy plik nazwij *Faucet.sol*.



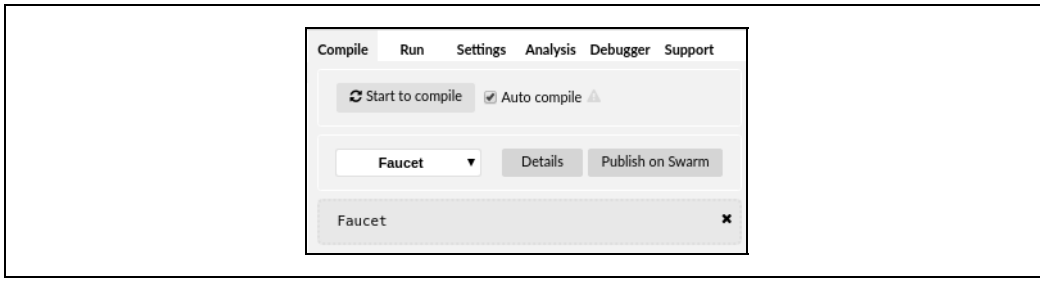
Rysunek 2.11. *Kliknij znak plus, aby otworzyć nową zakładkę*

Po otwarciu nowej zakładki skopiuj i wklej kod z przykładowego pliku *Faucet.sol* (rysunek 2.12).

Po wczytaniu kontraktu *Faucet.sol* do środowiska IDE Remix środowisko automatycznie skompiluje kod. Jeśli wszystko przebiegnie prawidłowo, po prawej stronie pod zakładką *Compile* pojawi się zielone pole z napisem *Faucet*. Jest to potwierdzenie udanej kompilacji (rysunek 2.13).



Rysunek 2.12. *Skopiuj do nowej zakładki kod przykładowego kontraktu Faucet*



Rysunek 2.13. *Remix z powodzeniem kompiluje kontrakt Faucet.sol*

Jeśli coś się nie powiodło, najbardziej prawdopodobnym problemem jest to, że środowisko IDE Remix używa kompilatora Solidity w wersji innej niż 0.4.19. Wtedy dyrektywa `pragma` sprawi, że kontrakt *Faucet.sol* się nie skompiluje. Aby zmienić wersję kompilatora, otwórz zakładkę *Settings*, wybierz wersję 0.4.19 i ponów próbę[1].

Kompilator języka Solidity skompilował plik *Faucet.sol* do postaci kodu bajtowego dla maszyny EVM. Jeśli jesteś ciekawy, ten kod bajtowy wygląda tak:

```
PUSH1 0x60 PUSH1 0x40 MSTORE CALLVALUE ISZERO PUSH2 0xF JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST PUSH1 0xE5 DUP1 PUSH2 0x1D PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN
STOP PUSH1 0x60 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT PUSH1 0x3F JUMPI
PUSH1 0x0 CALLDATALOAD PUSH29
0x100000000000000000000000000000000000000000000000000000000
SWAP1 DIV PUSH4 0xFFFFFFFF AND DUP1 PUSH4 0x2E1A7D4D EQ PUSH1 0x41 JUMPI
JUMPDEST STOP JUMPDEST CALLVALUE ISZERO PUSH1 0x4B JUMPI PUSH1 0x0 DUP1 REVERT
JUMPDEST PUSH1 0x5F PUSH1 0x4 DUP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1
SWAP2 SWAP1 POP POP PUSH1 0x61 JUMP JUMPDEST STOP JUMPDEST PUSH8
0x16345785D8A0000 DUP2 GT ISZERO ISZERO PUSH1 0x77 JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST CALLER PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND
PUSH2 0x8FC DUP3 SWAP1 DUP2 ISZERO MUL SWAP1 PUSH1 0x40 MLOAD PUSH1 0x0 PUSH1
0x40 MLOAD DUP1 DUP4 SUB DUP2 DUP6 DUP9 DUP9 CALL SWAP4 POP POP POP POP ISZERO
ISZERO PUSH1 0xB6 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP JUMP STOP LOG1 PUSH6
0x627A7A723058 KECCAK256 PUSH9 0x13D1EA839A4438EF75 GASLIMIT CALLVALUE LOG4 0x5f
PUSH24 0x7541F409787592C988A079407FB28B4AD000290000000000
```

Czy nie jesteś zadowolony, że używasz wysokopoziomowego języka takiego jak Solidity, zamiast programować bezpośrednio w kodzie bajtowym dla maszyny EVM? Ja jestem!

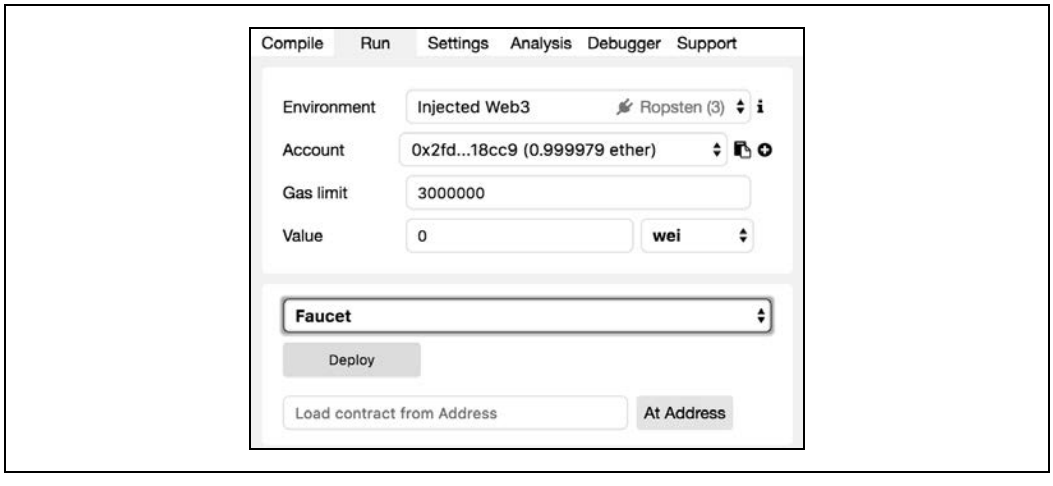
[1] Obecnie wersję kompilatora można zmienić w zakładce *Compile* — *przyp. tłum.*

Tworzenie kontraktu w łańcuchu bloków

Mamy więc kontrakt skompilowany do postaci kodu bajtowego. Teraz trzeba „zarejestrować" go w łańcuchu bloków Ethereum. Do przetestowania kontraktu posłuży sieć testowa Ropsten, dlatego trzeba przesłać kontrakt do łańcucha bloków Ropsten.

Rejestrowanie kontraktu w łańcuchu bloków wymaga utworzenia specjalnej transakcji z adresem docelowym `0x0000000000000000000000000000000000000000` (jest to **adres zerowy**). Jest to specjalny adres, informujący łańcuch bloków Ethereum, że chcesz zarejestrować kontrakt. Na szczęście środowisko IDE Remix wykona wszystkie potrzebne operacje i prześle transakcję do portfela MetaMask.

Najpierw otwórz zakładkę *Run* i wybierz opcję *Injected Web3* na liście rozwijanej *Environment*. Środowisko IDE Remix połączy się wtedy z portfelem MetaMask, a za pośrednictwem tego portfela — także z siecią testową Ropsten. Zobaczysz wtedy nazwę *Ropsten* przy polu *Environment*. W polu *Account* widoczny jest adres używanego portfela (rysunek 2.14).



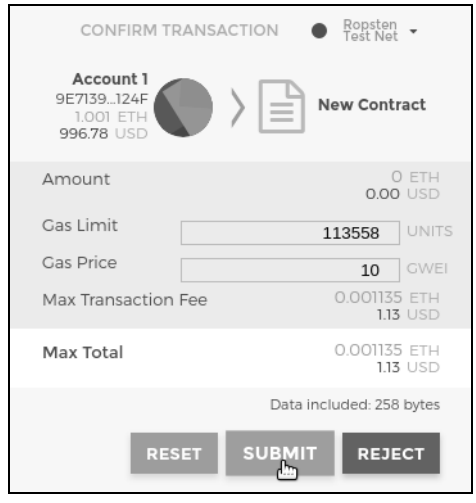
Rysunek 2.14. *Zakładka Run w środowisku IDE Remix. Wybrane jest środowisko Injected Web3*

Pod wybranymi właśnie ustawieniami uruchamiania kodu znajduje się gotowy do utworzenia kontrakt `Faucet`. Kliknij widoczny na rysunku 2.14 przycisk *Deploy*.

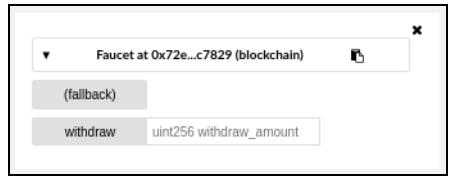
Remix utworzy wtedy specjalną transakcję tworzącą kontrakt, a MetaMask wyświetli prośbę o jej zatwierdzenie (rysunek 2.15). Warto zauważyć, że w tej transakcji tworzącej kontrakt nie są używane ethery, natomiast obejmuje ona 258 bajtów danych (jest to skompilowany kontrakt). Transakcja ta zużyje paliwo o wartości 10 gwei. Kliknij przycisk *Submit*, aby zatwierdzić transakcję.

Teraz musisz poczekać. Wydobycie kontraktu w sieci Ropsten zajmuje ok. 15 – 30 s. Środowisko Remix będzie wydawać się nieaktywne, zachowaj jednak cierpliwość.

Po utworzeniu kontrakt będzie widoczny w dolnej części zakładki *Run* (rysunek 2.16).



Rysunek 2.15. *MetaMask wyświetla transakcję tworzącą kontrakt*



Rysunek 2.16. *Kontrakt Faucet jest AKTYWNY!*

Warto zauważyć, że kontrakt `Faucet` ma teraz własny adres. Remix wyświetla go w napisie *Faucet at 0x72e…c7829* (Twój adres w postaci losowych liter i cyfr będzie inny). Niewielka ikona schowka widoczna po prawej stronie umożliwia skopiowanie adresu kontraktu do schowka. Posłużymy się tym adresem w następnym podrozdziale.

Interakcja z kontraktem

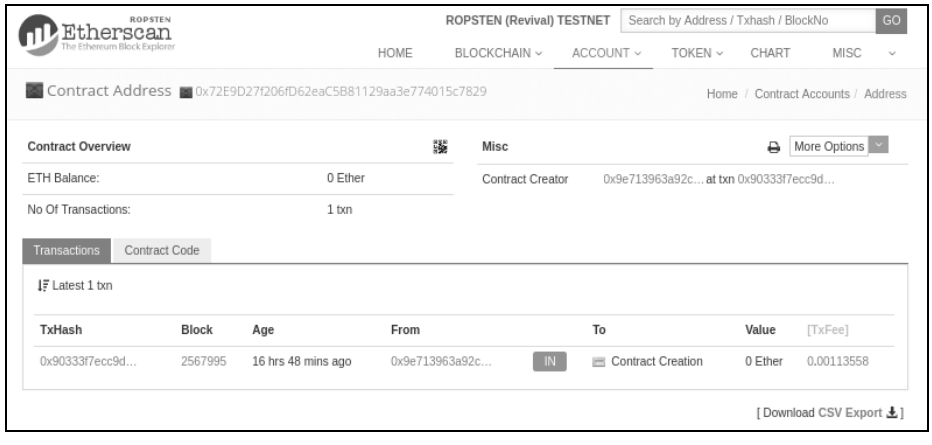
Warto podsumować przedstawione do tego miejsca informacje: kontrakty w Ethereum to programy kontrolujące pieniądze i działające w maszynie wirtualnej EVM. Są tworzone za pomocą specjalnej transakcji, która przesyła kod bajtowy rejestrowany w łańcuchu bloków. Kontrakt utworzony w łańcuchu bloków ma (podobnie jak portfele) adres Ethereum. Za każdym razem, gdy ktoś prześle transakcję na adres kontraktu, spowoduje to uruchomienie kontraktu w maszynie EVM z określoną transakcją jako danymi wejściowymi. Transakcje przesyłane na adres kontraktu mogą zawierać ethery, dane lub oba te komponenty. Jeśli transakcja zawiera ethery, są one deponowane w stanie konta. Jeżeli transakcja obejmuje dane, mogą one określać funkcję z kontraktu i wywoływać ją z podanymi argumentami.

Wyświetlanie adresu kontraktu w eksploratorze bloków

Kontrakt został już zarejestrowany w łańcuchu bloków i można się przekonać, że ma on adres Ethereum. Otwórz eksplorator bloków *https://ropsten.etherscan.io/* i sprawdź, jak wygląda kontrakt. W środowisku IDE Remix skopiuj adres kontraktu. W tym celu kliknij ikonę schowka widoczną obok nazwy kontraktu (rysunek 2.17).

Rysunek 2.17. *Kopiowanie adresu kontraktu ze środowiska Remix*

Pozostaw środowisko Remix otwarte. Wrócisz do niego później. Teraz przejdź w przeglądarce na stronę *https://ropsten.etherscan.io/* i wklej adres kontraktu w polu wyszukiwania. Powinieneś zobaczyć historię dla przypisanego do kontraktu adresu Ethereum (rysunek 2.18).



Rysunek 2.18. *Wyświetlanie adresu kontraktu Faucet w eksploratorze bloków Etherscan*

Zasilanie kontraktu

Na razie historia kontraktu obejmuje tylko jedną transakcję, tworzącą ten kontrakt. W eksploratorze widać, że kontrakt nie ma dostępnych etherów (stan jest równy zero). Wynika to z tego, że nie przesłano żadnych etherów do kontraktu w tworzącej go transakcji, choć było to możliwe.

Utworzony kran wymaga środków! Pierwszym zadaniem będzie użycie portfela MetaMask do przesłania etherów do kontraktu. Nadal powinieneś mieć dostępny w schowku adres kontraktu

(w przeciwnym razie ponownie skopiuj ten adres ze środowiska Remix). Otwórz portfel MetaMask i prześlij 1 ether na adres kontraktu w taki sam sposób, jakbyś używał dowolnego innego adresu Ethereum (rysunek 2.19).

Rysunek 2.19. *Prześlij 1 ether na adres kontraktu*

Jeśli po chwili odświeżysz stronę z eksploratorem bloków Etherscan, zobaczysz następną transakcję powiązaną z adresem kontraktu i zaktualizowany stan na poziomie 1 ethera.

Pamiętasz anonimową domyślną funkcję publiczną typu `payable` z kodu kontraktu *Faucet.sol*? Wygląda ona tak:

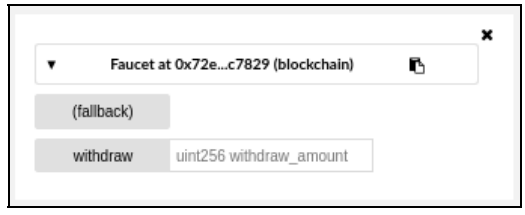
```
function () public payable {}
```

Gdy na adres kontaktu przesłałeś transakcję bez danych określających, którą funkcję należy wywołać, uruchomiona została ta domyślna funkcja. Ponieważ została ona zadeklarowana jako `payable`, przyjęła i zdeponowała 1 ether na koncie kontraktu. Transakcja spowodowała uruchomienie kontraktu w maszynie EVM i zaktualizowanie jego stanu. Właśnie zasiliłeś swój kran!

Wycofywanie środków z kontraktu

Następnie pobierzmy środki z kranu. W tym celu trzeba utworzyć transakcję, która wywołuje funkcję `withdraw` i przekazuje do niej argument `withdraw_amount`. Aby uprościć na razie rozwiązanie, użyjemy środowiska Remix do utworzenia transakcji, a portfel MetaMask wyświetli ją do zatwierdzenia.

Ponownie otwórz środowisko Remix i przyjrzyj się kontraktowi z zakładki *Run*. Powinieneś zobaczyć czerwone pole z etykietą `withdraw` i pole tekstowe z napisem `uint256 withdraw_amount` (rysunek 2.20).



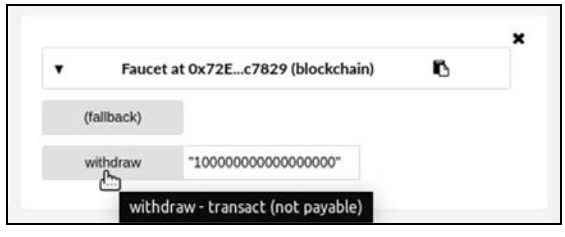
Rysunek 2.20. *Funkcja withdraw kontraktu Faucet.sol w środowisku Remix*

Jest to interfejs kontraktu w środowisku Remix. Ten interfejs umożliwia tworzenie transakcji, które wywołują funkcje zdefiniowane w kontrakcie. Możesz wpisać wartość parametru `withdraw_amount` i kliknąć przycisk *withdraw*, aby wygenerować transakcję.

Najpierw należy ustalić wartość parametru `withdraw_amount`. Chcemy pobrać 0,1 ethera, co jest maksymalną kwotą dopuszczalną w kontrakcie. Pamiętaj, że wszystkie wartości w Ethereum są wewnętrznie zapisywane w jednostkach wei. Funkcja `withdraw` także oczekuje, że parametr `withdraw_amount` będzie liczbą wei. Pożądana kwota to 0,1 ethera, co oznacza 100 000 000 000 000 000 wei (jedynka i 17 zer).

Z powodu ograniczeń JavaScriptu Remix nie potrafi przetwarzać liczb tak dużych jak 10^{17}. Dlatego należy umieścić tę wartość w cudzysłowie, aby Remix pobrał ją jako łańcuch znaków i przetwarzał jako wartość typu `BigNumber`. Jeśli nie umieścisz tej wartości w cudzysłowie, środowisko IDE Remix nie przetworzy jej i wyświetli komunikat o błędzie kodowania argumentów „Error encoding arguments: Error: Assertion failed".

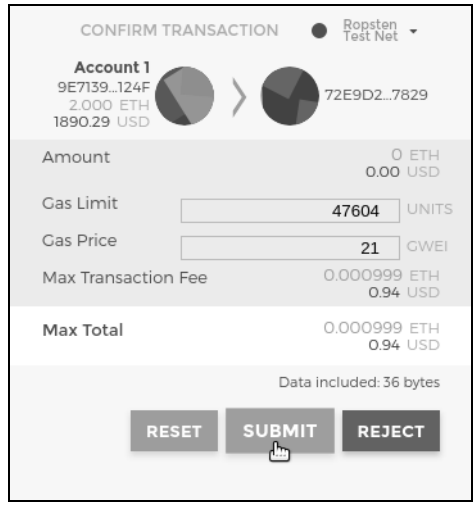
Wpisz `"100000000000000000"` (z cudzysłowem) w polu *withdraw_amount* i kliknij przycisk *withdraw* (rysunek 2.21).



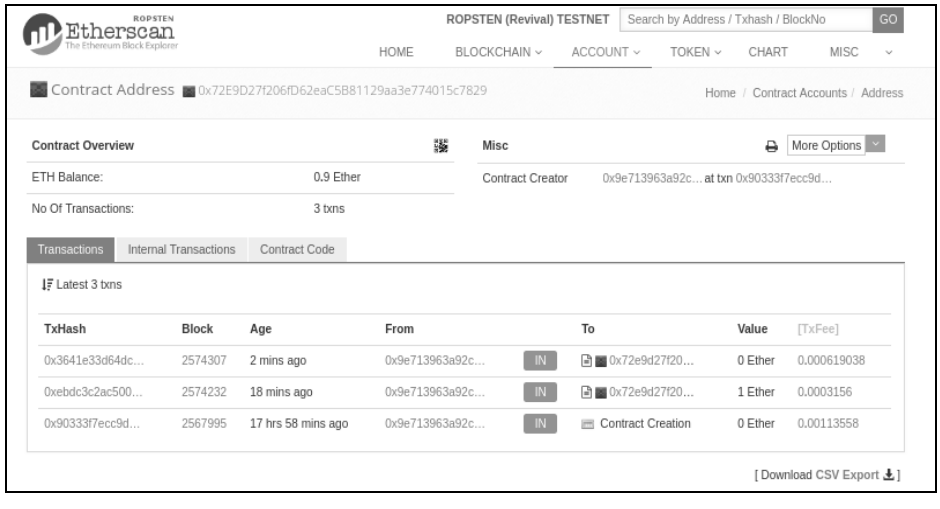
Rysunek 2.21. *Kliknij przycisk withdraw w środowisku Remix, aby utworzyć transakcję wycofywania środków*

MetaMask wyświetli okno z transakcją do zatwierdzenia. Kliknij przycisk *Submit*, aby przesłać do kontraktu żądanie wycofania środków (rysunek 2.22).

Odczekaj chwilę, a następnie odśwież stronę eksploratora bloków Etherscan, aby wyświetlić transakcję widoczną w historii adresu kontraktu `Faucet` (rysunek 2.23).



Rysunek 2.22. *Transakcja w portfelu MetaMask wywołująca funkcję withdraw*



Rysunek 2.23. *Eksplorator Etherscan wyświetla transakcję wywołującą funkcję withdraw*

Teraz widoczna jest nowa transakcja z adresem kontraktu jako adresem docelowym i wartością 0 etherów. Stan konta kontraktu się zmienił i wynosi obecnie 0,9 ethera, ponieważ zgodnie z żądaniem kontrakt przesłał 0,1 ethera. W *historii adresu kontraktu* nie widać jednak transakcji wyjściowej.

Gdzie jest transakcja wycofania środków? Na stronie historii adresu kontraktu pojawiła się nowa zakładka — *Internal Transactions*. Ponieważ transfer 0,1 ethera jest inicjowany przez kod kontraktu, wykonywana jest transakcja wewnętrzna (nazywana też **komunikatem**). Kliknij zakładkę *Internal Transactions*, aby zobaczyć jej zawartość (rysunek 2.24).

Rysunek 2.24. *Eksplorator Etherscan wyświetla wewnętrzną transakcję przekazującą ethery z kontraktu*

Wewnętrzna transakcja została przesłana przez kontrakt w przedstawionym wierszu kodu (z funkcji `withdraw` z kontraktu *Faucet.sol*):

```
msg.sender.transfer(withdraw_amount);
```

Oto podsumowanie: przesłałeś z portfela MetaMask transakcję, która zawierała instrukcje wywołania funkcji `withdraw` z argumentem `withdraw_amount` o wartości 0,1 ethera. Ta transakcja spowodowała uruchomienie kontraktu w maszynie EVM. Gdy maszyna EVM wykonywała funkcję `withdraw` kontraktu `Faucet`, najpierw wywołała funkcję `require` i upewniła się, że żądana kwota jest nie większa od maksymalnego dozwolonego poziomu 0,1 ethera. Następnie wywołana została funkcja `transfer`, aby przekazać ethery. Uruchomienie funkcji `transfer` wygenerowało wewnętrzną transakcję, która przesłała z konta kontraktu 0,1 ethera na adres portfela. Ta transakcja jest widoczna w zakładce *Internal Transactions* eksploratora Etherscan.

Podsumowanie

W tym rozdziale skonfigurowałeś portfel za pomocą rozszerzenia MetaMask i zasiliłeś go, używając kranu z sieci testowej Ropsten. Otrzymałeś ether na adres Ethereum Twojego portfela, a następnie odesłałeś ether na adres Ethereum kranu.

Później napisałeś w języku Solidity kontrakt działający jak kran. Do skompilowania kontraktu do kodu bajtowego maszyny EVM użyłeś środowiska IDE Remix. Następnie za pomocą tego środowiska przygotowałeś transakcję i dodałeś kontrakt `Faucet` do łańcucha bloków Ropsten. Utworzony kontrakt `Faucet` miał adres Ethereum, na który przesłałeś środki. W ostatnim kroku utworzyłeś transakcję, aby wywołać funkcję `withdraw`, i z powodzeniem zażądałeś 0,1 ethera. Kontrakt sprawdził żądanie i odesłał Ci 0,1 ethera za pomocą wewnętrznej transakcji.

Może wydawać się, że to niewiele, ale właśnie z powodzeniem skomunikowałeś się z oprogramowaniem, które kontroluje pieniądze w zdecentralizowanym światowym komputerze.

W rozdziale 7. napiszesz dużo więcej kodu inteligentnych kontraktów, a w rozdziale 9. poznasz najlepsze praktyki i uwagi z zakresu bezpieczeństwa.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

Z

Ź

O'REILLY®

Nowy wymiar innowacji w programowaniu!

Ethereum to platforma służąca do obsługi kryptowalut. Została zaprojektowana z zachowaniem wysokich standardów bezpieczeństwa i transparentności. Umożliwia uruchamianie zdecentralizowanych aplikacji (DApp) i inteligentnych kontraktów, które nie mają scentralizowanej kontroli ani pojedynczego punktu podatności na awarie i są zintegrowane z siecią obsługi płatności oraz operują na otwartym łańcuchu bloków. Ethereum cieszy się dużym zainteresowaniem takich organizacji jak IBM, Microsoft, NASDAQ, które coraz bardziej angażują się w korzystanie z tej platformy.

Ta książka jest praktycznym poradnikiem i encyklopedycznym źródłem wiedzy o Ethereum przeznaczonym dla programistów, którzy chcą przyswoić wiadomości dotyczące łańcuchów bloków oraz tworzenia inteligentnych kontraktów i zdecentralizowanych aplikacji — DApp. Znalazły się tu zarówno podstawowe informacje, jak i szereg zaawansowanych zagadnień związanych z Ethereum. Opisano, w jaki sposób tworzy się w nim transakcje, przedstawiono kwestie związane z kluczem publicznym, skrótami i podpisami cyfrowymi. Poszczególne zagadnienia, a także najlepsze praktyki, wzorce projektowe i antywzorce z obszaru bezpieczeństwa uzupełniono starannie opracowanymi przykładami kodu.

W tej książce między innymi:

- uruchamianie klienta Ethereum
- korzystanie z portfeli cyfrowych
- interakcje z klientami Ethereum za pomocą wywołań RPC
- tokeny reprezentujące aktywa, udziały czy głosy
- budowa zdecentralizowanych aplikacji z użyciem komponentów w modelu P2P

Andreas M. Antonopoulos jest uznanym autorytetem w dziedzinie kryptowalut i łańcuchów bloków. Jest też znany z wystąpień, w których łączy ekonomię, psychologię i teorię gier z technologią i historią. W ten sposób sprawia, że najbardziej abstrakcyjne zagadnienia stają się zrozumiałe i rzeczywiste.

Dr Gavin Wood jest programistą, technologiem, wykładowcą i projektantem gier. Jest też architektem Etherneum. Zaprojektował język kontraktów Solidity i napisał specyfikację typu Yellow Paper — pierwszą formalną specyfikację protokołu łańcucha bloków.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-5574-3

INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 79,00 zł