### STRESZCZENIE

Celem naszego projektu inżynierskiego było stworzenie portfela kryptowaluty Bitcoin o modularnej architekturze, który nie jest obecny aktualnie na rynku. Projekt obejmuje zarówno oprogramowanie jak i sprzęt, gdyż postawiliśmy sobie za cel stworzenie działającego produktu, który zainstalujemy na popularnym w ostatnim czasie Raspberry Pi Zero W. Jest to urządzenie typu SoC (System on a Chip), czyli bardzo mały komputer o ograniczonych możliwościach, mieszczący się na jednej płytce drukowanej. Głównym założeniem architektury było stworzenie uniwersalnego interfejsu dla modułów, które tworzą użytkownicy. Każdy moduł odpowiedzialny jest za dostarczenie funkcjonalności szyfrowania i odszyfrowania fragmentu klucza, który jest mu przekazywany. W zależności od implementacji samego modułu, może on być bardzo skomplikowany lub mniej wyszukany. Moduły traktowane są jako odrębne programy, które jedynie wywiązują się z kontraktu, którym jest implementacja w dostarczonym frameworku. Funkcjonalności portfela udostępniane są w postaci przeglądarkowej aplikacji, która wykorzystuje nowoczesne technologie takie jak języki Java i Kotlin, w którym zaimplementowaliśmy bardzo lekki serwer HTTP; komunikację przy pomocy HTTP i Websocket, oraz framework React stworzony w języku JavaScript, który aktualnie wiedzie prym w rozwiązaniach Frontendowych.

Zaproponowaliśmy i zaimplementowaliśmy kilka przykładowych modułów, wpisujących się w naszą architekturę. Niektóre są bardzo proste, podczas gdy inne mogłyby być rozwinięte do miana pełnoprawnych projektów. Ze względu na bardzo małe ograniczenia stawiane podczas implementacji modułu w naszej architekturze, ich twórcy mogą konstruować je praktycznie w dowolnych technologiach. Taką możliwość przedstawiliśmy w jednym z naszych modułów, który stosuje odmienny stos technologiczny.

Ważnym z punktu widzenia całego projektu, jest podejście do organizacji pracy i zarządzania całym projektem. Zmierzyliśmy się z problemem rozproszonej pracy zdalnej, którą należy synchronizować pomiędzy trzema osobami. Na początku staraliśmy się wpisać w kanon gotowych metodyk i zbiorów narzędzi, zalecanych na różnych przedmiotach. Okazało się, że takie podejście nie było wystarczająco efektywne i musieliśmy je zupełnie porzucić na korzyść sprawdzonych metod, wykorzystywanych w większości nowych projektów open source, czyli podejście inspirowane metodykami iteracyjnymi, oparte o rozwinięte narzędzie kontroli wersji GitHub.

Ostatecznie, utworzyliśmy działający prototyp portfela. Najważniejsze z początkowych założeń zostały osiągnięte, a cały projekt jest gotowy do dalszego rozwoju.

Podział prac w tworzeniu dokumentacji został zawarty w każdym rozdziale, poprzez określenie autora pod tytułem rozdziału. Dokładny spis autorstwa rozdziałów:

Patryk Milewski - 1, 1.1, 4, 4.1, 4.3, 4.4, 4.5, 4.6, 5.4, 5.6, 6, 6.1, 6.2, 7, 7.1, 7.2, 7.4, 7.5  
Stanisław Barański - 1.3, 2, 2.1, 2.2, 2.3, 2.4, 2.5, 4.2, 5.2, 5.7

Piotr Pabis - 1, 1.1, 1.2, 3, 3.1, 3.2, 5, 5.1, 5.3, 5.5, 5.9, 7.3

**Słowa kluczowe:** portfel kryptowalut, Raspberry Pi, modularność, Java, Kotlin, JVM, React, Bitcoin, kryptografia, open source, GitHub.

**Dziedzina nauki i techniki, zgodnie z wymogami OECD:** Nauka o komputerach, informatyka i bioinformatyka.Sprzęt komputerowy i architektura komputerów. Elektrotechnika i elektronika.

### ABSTRACT

The primary goal of our engineering project was the construction of a hardware wallet for the Bitcoin cryptocurrency, which utilises a modular architecture to secure the wallet’s secret. None of the solutions currently available on the market provide such functionality. The project comprises of both the software and hardware aspects because we want the final product to be released as ready-to-use. The hardware platform of our choice is Raspberry Pi Zero W - an affordable and popular System on a Chip, which is small in size with limited resources. One of the goals of our architecture is to provide an universal interface for modules, allowing users to implement their own authentication factors. Every module must be able to encrypt and decrypt a given payload - a part of user’s secret - needed to sign Bitcoin transactions. They can be simple or very complicated - implementation is left up to the developer to fit their needs. Modules are treated as standalone programs that must adhere to a given contract, provided as an abstract class. The end user can access our wallet through a web interface which was built using modern technologies such as Java, Kotlin in which we have implemented a lightweight HTTP server, Websockets for dynamic updates in the UI and a widely used React framework.

To make the prototype functional, we have provided some example modules. While most of them are basic, one is distinctively complex and can be developed into a standalone project. Due to the very low restrictions to implement a module for our wallet, developers can use a diverse range of technologies to create a new one. We demonstrate such possibility in one of our modules which utilises a different technological stack.

An important problem we encountered during the work on our project was the organisation and management of a team of three developers. In the beginning, we wanted to use a classic methodology that we learnt throughout our courses at Gdansk University of Technology. Unfortunately, during the implementation phase, these methods began to slow us down. Because of that, we changed our approach to non-formal methodology, present in modern open source projects, that is inspired by an iterative process and based on Pull Requests. The GitHub platform aided us to effectively cooperate on the project.

In the end, we created a working prototype of the wallet. The most important requirements assigned in the beginning have been achieved and the project has the potential to be later developed.

**Keywords:** cryptocurrency wallet, Raspberry Pi, modularity, Java, Kotlin, JVM, React, Bitcoin, cryptography, open source, GitHub.

### WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

***Raspberry Wallet***–nazwa własna naszego projektu.

***Blockchain*** –zdecentralizowana i rozproszone baza danych przechowująca transakcje w formie bloków, połączonych ze sobą algorytmami kryptograficznymi.

***GUID, UUID*** –Globally Unique IDentifier, Universally Unique IDentifier, są to generowane losowo 128 bitowe liczby, które gwarantują niepowtarzalność.

***JAR*** –archiwum zawierające skompilowane klasy języka Java, swoje metadane oraz zasoby.

***ID*** –IDentifier, identyfikator.

***Frontend***–część systemu, która jest odpowiedzialna za wyświetlanie interfejsu użytkownika.

***Backend*** –część systemu, która jest odpowiedzialna za logikę biznesową, przeważnie pod postacią niewidocznego dla użytkownika serwera.

***YAML*** –Yet Another Markup Language - język formatu danych oparty o tabulacje oraz prostą interpunkcję, w 100% kompatybilny z innymi popularnymi formatami takimi jak JSON i XML.

***JSON*** –JavaScript Object Notation - język formatu danych stylizowany na kod ECMAScript (potocznie JavaScript). Cechuje się małym rozmiarem i łatwością odczytu jako plik tekstowy.

***Kotlin*** –statycznie typowany język programowania, którego główną platformą uruchomieniową jest maszyna wirtualna Javy, pozwala również na kompilacje do JavaScriptu, a nawet do kodu natywnego. Ogromny wzrost popularności zawdzięcza interoperacyjności z kodem źródłowym Javy.

***JVM*** –Java Virtual Machine, maszyna wirtualna Javy, na której wykonywany jest kod w postaci bytecode.

***GitHub*** –serwis umożliwiający przechowywanie i publikowanie repozytoriów systemu kontroli wersji Git, oraz tworzenie i zarządzanie całymi projektami.

***Issue*** –w serwisie GitHub: publikowany przez użytkownika błąd lub pomysł, którego naprawą lub implementacją powinni zająć się członkowie zespołu projektowego. Pod każdym Issue może być prowadzona dyskusja.

***Pull Request*** –w serwisie GitHub: prośba wysłana do innych członków zespołu projektowego, o rozpatrzenie scalenia jednej gałęzi kodu z inną. Rozszerza Issue.

***Moduł*** –dynamicznie ładowany podprogram, którego zadaniem jest zaszyfrowanie oraz odszyfrowanie danych swoim unikalnym algorytmem. Zawierany jest on w pakiecie JAR, podpisanym naszym certyfikatem.

***BTC*** –oznaczenie waluty Bitcoin.

***BIP*** –Bitcoin Improvement Proposal, czyli propozycja udoskonalenia protokołu Bitcoin.

***Oracle Java JRE 8*** –implementacja maszyny wirtualnej Java od firmy Oracle.

***ISC DHCP Server*** –serwer DHCP organizacji Internet Systems Consortium.

### 

### SPIS TREŚCI

[STRESZCZENIE 1](#_Toc532063340)

[ABSTRACT 2](#_Toc532063341)

[WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW 3](#_Toc532063342)

[SPIS TREŚCI 4](#_Toc532063343)

[1. WSTĘP I CEL PRACY 6](#_Toc532063344)

[1.1. Motywacja 6](#_Toc532063345)

[1.2. Cel Pracy 7](#_Toc532063346)

[1.3. Rozwiązania konkurencji 7](#_Toc532063347)

[1.3.1. Ledger Nano S 8](#_Toc532063348)

[1.3.2. Portfel online 9](#_Toc532063349)

[1.3.3. Portfel w postaci programu 10](#_Toc532063350)

[1.3.4. Portfel papierowy 10](#_Toc532063351)

[2. Bitcoin 12](#_Toc532063352)

[2.1. Transakcje 13](#_Toc532063353)

[2.2 Opłaty 16](#_Toc532063354)

[2.3 Węzły w sieci Bitcoin 16](#_Toc532063355)

[2.4 Simplified Payment Verification 17](#_Toc532063356)

[*2.5. Kopia zapasowa portfela - BIP39* 18](#_Toc532063357)

[3. Wykorzystane technologie 20](#_Toc532063358)

[3.1. Urządzenie Raspberry Pi Zero W 21](#_Toc532063359)

[3.1.1. Pomiary prędkości i optymalizacja 22](#_Toc532063360)

[3.2. Bezpieczeństwo urządzenia Raspberry Pi Zero 23](#_Toc532063361)

[3.2.1. Użytkownik *wallet* 24](#_Toc532063362)

[3.2.2. Filtr sieciowy IPTables 24](#_Toc532063363)

[3.2.3. Weryfikacja podpisu modułu 24](#_Toc532063364)

[3.2.4. Możliwości podniesienia bezpieczeństwa 25](#_Toc532063365)

[4. Architektura i moduły 26](#_Toc532063366)

[4.1. Schemat architektury oraz analiza działania 26](#_Toc532063367)

[4.2. Ochrona sekretu przez portfel 28](#_Toc532063368)

[4.3. Dzielenie sekretu - algorytm Shamira 31](#_Toc532063369)

[4.4. Konfiguracja portfela oraz modułów 32](#_Toc532063370)

[4.5. Moduły zaimplementowane w naszej architekturze 33](#_Toc532063371)

[4.5.1. Przełącznik uwierzytelniający transakcję 33](#_Toc532063372)

[4.5.2. Uwierzytelniająca sieć serwerów 34](#_Toc532063373)

[4.5.4 Odblokowanie portfela za pomocą kodu PIN 39](#_Toc532063374)

[4.6. Moduły odrzucone na etapie analizy i projektowania portfela 39](#_Toc532063375)

[4.6.1. Weryfikacja za pomocą telefonu 39](#_Toc532063376)

[4.6.2. Kody QR przechowujące część klucza 40](#_Toc532063377)

[4.6.3. Wykrywanie obecności urządzenia Bluetooth 41](#_Toc532063378)

[4.6.4. Wirtualne moduły 43](#_Toc532063379)

[5. Wnioski z procesu implementacji 44](#_Toc532063380)

[5.1. Brak zegara czasu rzeczywistego w Raspberry Pi 44](#_Toc532063381)

[5.2. Niepełna kompatybilność biblioteki BitcoinJ 45](#_Toc532063382)

[5.3. Nieaktualna dokumentacja biblioteki Pi4J 45](#_Toc532063383)

[5.4. Określenie finalnego interfejsu komunikacji z modułami 46](#_Toc532063384)

[5.5. Osiągnięcie wysokiej entropii generatora liczb pseudolosowych w systemie Linux 47](#_Toc532063385)

[5.6. Testy jednostkowe wymagające dostępu do API innej aplikacji 48](#_Toc532063386)

[5.7. Dust Payments 48](#_Toc532063387)

[5.8. Bezstanowość modułu fizycznego przełącznika 49](#_Toc532063388)

[6. Narzędzia i metody współpracy 50](#_Toc532063389)

[6.1. Przygotowanie wymagań i dokumentacji wstępnej 50](#_Toc532063390)

[6.2. Implementacja kodu 52](#_Toc532063391)

[7. PODSUMOWANIE 56](#_Toc532063392)

[7.1. Portfel kryptowalut 56](#_Toc532063393)

[7.2. Zastosowane technologie 56](#_Toc532063394)

[7.3. Urządzenie Raspberry Pi Zero W 57](#_Toc532063395)

[7.4. Zastosowany model współpracy 57](#_Toc532063396)

[7.5. Plany na przyszłość 58](#_Toc532063397)

[Wykaz literatury 59](#_Toc532063398)

[Dodatek A 65](#_Toc532063399)

[Przykładowa zawartość pliku *sudoers* 65](#_Toc532063400)

[Dodatek B 66](#_Toc532063401)

[Zawartość abstrakcyjnej klasy Module modułu Backend Manager 66](#_Toc532063402)

### 

### 1. WSTĘP I CEL PRACY

*Piotr Pabis*

Od dawna ludzkość stosowała różne formy zapłaty za towary i usługi. Z początku wymieniano się przedmiotami, następnie drogimi i rzadkimi przyprawami jak sól i pieprz oraz kruszcami takimi jak złoto i srebro. Przy wzroście rozwoju cywilizacji i struktur społecznych, państwo przejęło problematykę finansów, poprzez emitowanie monet i banknotów. Po latach powstawały banki oraz pojawiły się rozwiązania takie jak możliwość wysłania przelewu i czeków pomiędzy ich klientami. Na przełomie XX i XXI wieku, wprowadzono bankowości online, która bazuje na przechowywaniu w bazie danych informacji o saldzie klienta. Zrewolucjonizowało to zupełnie podejście do bankowości.

Około roku 2009, rozpowszechniła się nowa koncepcja, nawiązująca do starszych form płatności. Podczas gdy w czasach preindustrialnych każdy posiadał namacalną formę pieniądza, w latach ‘90 XX wieku to banki posiadały pieniądze swoich klientów, klienci zaś sposób uwierzytelnienia i skojarzenia swojej osoby z wpisem w bazie danych banku. Z powodu spadającego zaufania społecznego po 2000 roku do banków oraz rządów, opublikowana została praca zatytułowana *“Bitcoin: A Peer-to-Peer Electronic Cash System”* autorstwa Satoshi Nakamoto [1]. Nieznany jest autor lub autorzy tej pracy. Nazwa “Satoshi Nakamoto” jest jedynie pseudonimem. System Bitcoin nie potrzebuje zaufanej trzeciej strony jak w przypadku bankowości, a dodatkowo gwarantuje użytkownikowi absolutną własność nad swoimi środkami. Satoshi w swojej pracy nawiązał do znanych już pomysłów cyfrowych walut jak B-money [2] i HashCash [3]. Główną innowacją jaką wprowadził Bitcoin, jest algorytm Proof-of-Work. Implementuje on system porozumienia pomiędzy wszystkimi węzłami w sieci peer-to-peer, szerzej opisany w rozdziale 2. Bitcoin jest owocem dziesiątek lat badań i rozwoju w dziedzinie kryptografii oraz systemów rozproszonych.

#### 1.1. Motywacja

*Piotr Pabis*

Podobnie jak w przypadku innych form płatności, istnieją osoby niepowołane, próbujące wejść w posiadanie czyichś zasobów finansowych. W przypadku kryptowalut głównym celem ataku jest klucz prywatny. Znajomość klucza prywatnego portfela, jest tożsama posiadaniu środków na nim zgromadzonych. W wyniku tego środki zgromadzone na wirtualnych portfelach stały się jednym z głównych celów ataków. Jak raportuje Reuters, w 2017 roku suma kradzieży kryptowalut wyniosła około 1,2 miliarda dolarów [4]. Problem ten staje się coraz większy - w pierwszej połowie 2018 roku skradzionych zostało aż 1,1 mld dolarów [5], co może prognozować wzrost o 100% w porównaniu do roku poprzedniego.

Wielu użytkowników ufa internetowym portfelom oraz giełdom, trzymając tam swoje środki i nie dbając o to, że są to główne cele ataków hakerskich, ze względu na największe skupisko zaalokowanych tam pieniędzy. Inną powszechną formą trzymania kryptowalut są portfele instalowane w formie aplikacji na komputery osobiste oraz smartfony. Powszechnie wiadomo o podatności systemów operacyjnych tych urządzeń na ataki ze strony hakerów i wirusów, w szczególności gdy osoba konfigurująca taki system, nie jest ekspertem od cyberbezpieczeństwa. Z tego powodu powstała trzecia alternatywa - sprzętowo zabezpieczony portfel zaprojektowany wyłącznie do przetrzymywania kluczy prywatnych kryptowalut. Jednym z dostępnych na rynku takim urządzeniem jest Ledger Nano S, opisany szerzej w rozdziale 1.3.1. Pozwala on przetrzymywać klucz prywatny w odizolowanej, zabezpieczonej pamięci stałej oraz podpisywać transakcje. Klucz prywatny z założenia nigdy nie opuszcza urządzenia. Jednak jedynym zabezpieczeniem przed niepowołanym użyciem, jest kod PIN, a cały klucz prywatny nie jest rozproszony i znajduje się w jednym miejscu.

#### 1.2. Cel Pracy

*Piotr Pabis*

Wzorując się na urządzeniu Ledger Nano S, postanowiliśmy stworzyć prototyp fizycznego portfela kryptowaluty Bitcoin, zabezpieczającego klucz prywatny. W odróżnieniu od naszego wzorca chcieliśmy, aby nasz produkt pozwalał użytkownikowi dostosować poziom bezpieczeństwa. Kolejnym celem, który sobie postawiliśmy, jest działanie w odizolowanym środowisku oraz zapewnienie wszystkich podstawowych funkcjonalności, które oferują typowe portfele Bitcoin.

Głównymi właściwościami skalowalności rozwiązania jest możliwość określenia liczby i rodzaju składników zabezpieczenia klucza prywatnego oraz fakt, że każdy z tych sposobów może być rozszerzalny poprzez instalację modułów przez samego użytkownika.

Nie zapomnieliśmy również o łatwości użytkowania i wygodnym interfejsie użytkownika. Nasz portfel po podłączeniu do komputera za pomocą USB, staje się dostępny przez webową aplikację, którą można otworzyć w przeglądarce.

#### 1.3. Rozwiązania konkurencji

*Stanisław Barański*

Jesteśmy świadomi tego, że portfel kryptowalut nie jest nowością na rynku i istnieje wiele konkurencyjnych rozwiązań. Większość z nich bazuje na niezbyt bezpiecznych rozwiązaniach, gdzie wystarczy złamać jedno zabezpieczenie, aby wykraść środki zgromadzone w portfelu. Niektóre z nich są w postaci, które wzbudza podejrzenia już od samego początku, czyli webowa aplikacja, której kod jest zamknięty, a twórca jest anonimową osobą. Z drugiej strony mamy rozwiązania, które są bardzo dopracowane, bogate w funkcje i sprawdzone. Sami nie chcielibyśmy korzystać z portfela, o którym nic nie wiemy. Z drugiej strony kupienie sprawdzonego produktu, często wiąże się z dużymi kosztami, które zniechęcają większość potencjalnych użytkowników. Rozdział ten jest poświęcony spojrzeniu na rynek i zbadaniu aktualnej sytuacji, którą chcielibyśmy w przyszłości zmienić i dodać nasz kompromis, zarówno pomiędzy ceną, jak i pod względem bezpieczeństwa.

##### 1.3.1. Ledger Nano S



Rys. 1.1. Ledger Nano S

Ledger Nano S jest aktualnym liderem na rynku fizycznych portfeli kryptowalut. Urządzenie widocznie na zdjęciu 3.1 jest rozmiarów pendrive. Pozwala ono na przechowywanie głównego ziarna portfela oraz podpisywanie transakcji dla ponad 25 kryptowalut. Główną zaletą działania takiego rozwiązania jest całkowita izolacja głównego ziarna, oraz pochodnych od niego kluczy prywatnych. Urządzenie nigdy nie udostępnia swojego ziarna jak i kluczy prywatnych, a części fizyczne zabezpieczone są w ten sam sposób, co karta SIM - czarna skrzynka, z której niemożliwe jest wyciągnięcie przechowywanego sekretu.

Ledger pracuje na systemie operacyjnym zwanym Blockchain Open Ledger Operating System, w skrócie BOLOS. System dostarcza lekkie środowisko open-source, pozwalające deweloperom na budowanie przenośnych aplikacji, które będą działać w bezpiecznym środowisku. Aplikacjami mogą być zarówno portfele kryptowalut jak i aplikacje przechowujące klucze SSH, PGP lub menedżer haseł do kont różnych serwisów. Ledger Nano S poza przechowywaniem głównego ziarna, z którego potem generuje klucze pochodne dla każdej aplikacji, pozwala na instalowanie aplikacji, które wiedzą jak podpisać tymże kluczem transakcje dla danej kryptowaluty. Aplikacje są dodatkowo izolowane od siebie i głównego ziarna przez ARM Memory Protection Unit - MPU, który natywnie odizolowuje pamięć każdej aplikacji. Dzięki takiemu rozwiązaniu każda aplikacja działa w swoim odizolowanym środowisku, czyli tak zwanym sandboxie, bez świadomości działających obok niej innych aplikacji. MPU zapewnia interfejs do komunikacji z systemem operacyjnym. BOLOS nie udostępnia głównego ziarna urządzenia, a jedynie pozwala na pobranie pochodnej tego klucza. Jest to możliwe dzięki zastosowaniu mechanizmu hierarchicznych deterministycznych kluczy [6]. Aplikacje nie muszą pamiętać swoich kluczy prywatnych, ponieważ zawsze są w stanie wygenerować je z ziarna głównego. Oznacza to, że w przypadku utraty urządzenia, do odzyskania środków potrzebujemy jedynie ziarna głównego. Następnie korzystając z faktu, że klucze pochodne generowane są w sposób deterministyczny, wszystkie środki na różnych kryptowalutach będzie dało się odzyskać. Dlatego tak ważne jest, aby posiadać kopię zapasową swojego ziarna głównego. Może być ono wygenerowane w postaci 24 słów mnemonicznych zgodnie z postulatem BIP39 [7].

Dodatkowe zabezpieczenia zastosowane w Ledger Nano S:

1. Zabezpieczenie kodem PIN - 3 błędne próby resetują całe urządzenie.
2. Potwierdzenie transakcji fizycznym guzikiem.
3. Automatyczne blokowanie po 1 min bezczynności.

Tabela 1.1. Lista wad i zalet portfela Ledger Nano S

|  |  |
| --- | --- |
| **Zalety** | **Wady** |
| Wysokie bezpieczeństwo zapewnione w warstwie sprzętowej systemu operacyjnego oraz aplikacji | Wysoka cena 100 € |
| Ziarno główne portfela nigdy nie opuszcza urządzenia | Mała pamięć (do ok. 7 aplikacji na raz) |
| Wsparcie standardu BIP39 [7] | Brak możliwości dostosowywania i rozbudowywania o własne zabezpieczenia |
| Wymiary zbliżone do pendrive | Wszystkie zabezpieczenia są bezużyteczne, gdy niedostatecznie zabezpieczona zostanie kopia zapasowa ziarna głównego |
| Wsparcie praktycznie nieograniczonej ilości kryptowalut |  |
| Możliwość pracy bez połączenia z Internetem |  |

Nie ukrywamy, że wzorem podczas projektowania naszego rozwiązania był w znacznej mierze Ledger Nano S. Staraliśmy się zaadaptować jego zalety i zniwelować wady, dodając przy okazji własną architekturę i otwartość kodu. Jednak niektórych wad, jak np. problem kopii zapasowej ziarna, nie da się wyeliminować.

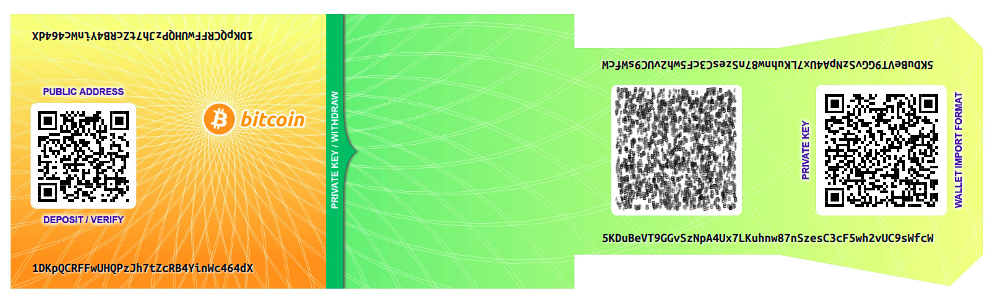
##### 1.3.2. Portfel online

Istnieją serwisy, które oferują usługę przechowywania kluczy prywatnych portfela na swoich serwerach. Jest to najprostsze i najwygodniejsze rozwiązanie ze wszystkich możliwych. Pośrednio tego typu portfelami są giełdy kryptowalut. Jednak historia pokazała, że giełdy kryptowalut dzielą się na dwa typy: te które zostały już okradzione i te które dopiero będą [8]. Nie odstrasza to mimo wszystko znacznej rzeszy użytkowników, którzy nieświadomi konsekwencji wybierają ten typ portfela głównie przez wygodę i brak konieczności zrozumienia technologii stojącej za kryptografią. Aby założyć taki portfel, wystarczy uzupełnić prosty formularz rejestracji i zapamiętać swoje dane logowania.

##### 1.3.3. Portfel w postaci programu

Kolejne rozwiązanie bardzo lubiane przez użytkowników kryptowalut. Pozwala na przechowywanie klucza prywatnego na komputerze lub smartfonie, co rodzi pewne zagrożenia, ale jest też bardzo wygodne. Jest to kompromis pomiędzy portfelami fizycznymi jak np. Ledger, a portfelami online. Portfele na smartfony często posiadają różne ciekawe mechanizmy zabezpieczeń. Niektóre z nich są zabezpieczeniami sprzętowymi, np. rozszerzenia oparte o ARM TrustZone [9], takie jak Samsung Knox [10] czy Secure Enclave [11] dla iPhone. Dodatkowo portfele programowe pozwalają na integrację z portfelami fizycznymi. Wykorzystuje to mechanizm podpisywania transakcji, dostarczony przez portfel. Aplikacja tworzy transakcję, a następnie wysyła ją do fizycznego portfela, który ją podpisuje i zwraca do aplikacji. Większość oprogramowania portfeli jest portfelami lekkimi lub SPV, czyli Simplified Payment Verification. Oznacza to, że portfel nie przechowuje całego blockchainu lokalnie, co za tym idzie nie jest w stanie sam zweryfikować transakcji i musi prosić oto pełnych klientów sieci P2P. Zostało to szerzej opisane w rozdziale (2.4). Najpopularniejsze portfele z tej kategorii to: Electrum [12], Jaxx [13], Exodus [14] i Copay [15].

##### 1.3.4. Portfel papierowy





Rys. 1.2. Papierowy portfel Bitcoin wygenerowany za pomocą strony https://bitcoinpaperwallet.com

Przedstawiony na rysunku 1.2 przykładowy portfel papierowy jest prawdopodobnie jedną z najbezpieczniejszych form przechowywania klucza prywatnego. Nie zawiera on żadnych elektronicznych elementów, dlatego kradzież na odległość jest niemożliwa. Ponadto nawet w przypadku skopiowania go, dane przechowywane na nim mogą zostać zaszyfrowane hasłem, co zabezpiecza nas przed takimi atakami.

Główną wadą tego rodzaju portfela jest niepraktyczność – za każdym razem należy przepisać lub zeskanować jego zawartość, a w przypadku użycia do tego urządzenia takiego jak smartfon, powoduje to dodatkowe zagrożenia, w postaci przechwycenia klucza prywatnego podczas przesyłu. Rozwiązanie te jest również dużo bardziej narażone na fizyczne zniszczenie lub zgubienie.

### 

### 2. Bitcoin

*Stanisław Barański*

Bitcoin jest zbiorem technologii, które tworzą środowisko zdecentralizowanego systemu rozliczeniowego. Nazwa została również użyta, jako jednostka waluty, która służy do przechowywania i przekazywania wartości pomiędzy uczestnikami sieci Bitcoin. Użytkownicy systemu komunikują się ze sobą używając protokołu Bitcoin za pośrednictwem sieci Internet. System jednak nie narzuca żadnych ograniczeń, co do warstwy transportowej, która musi zostać użyta, z tego powodu w teorii Bitcoin powinien działać również bez Internetu. Oprogramowanie używane do pracy z siecią Bitcoin, jest ogólnie dostępne w formie open-source i może zostać uruchomione na praktycznie każdym popularnym urządzeniu, takim jak komputer osobisty, smartfon, Raspberry Pi, a nawet w przeglądarce internetowej.

Użytkownik sieci może nabyć swoje Bitcoiny tak samo, jak ma to miejsce w konwencjonalnym systemie finansowym, czyli zakupić środki na specjalnej giełdzie kryptowalut za inną walutę lub otrzymać przelew od innego użytkownika sieci. Dodatkowo Bitcoin pozwala na tzw. “kopanie” nowych środków, co jest analogią do wydobywania surowców.

W przeciwieństwie do konwencjonalnych walut, Bitcoin jest walutą całkowicie wirtualną, tzn. nie istnieje jej fizyczny odpowiednik. Co więcej Bitcoiny są nierozróżnialne, oznacza to, że nie można zidentyfikować Bitcoina na podstawie np. numeru seryjnego. Bitcoin istnieje jedynie jako wartość transakcji pomiędzy dwoma użytkownikami sieci. Jego posiadanie jest możliwe dzięki kluczom, które pozwalają na wykonanie przelewu środków, które wcześniej otrzymaliśmy. Klucze najczęściej są przechowywane w cyfrowych portfelach na urządzeniach użytkowników. Posiadanie kluczy, które pozwalają na podpisanie transakcji, jest równoważne z możliwością przelewu środków i daje absolutną władzę użytkownikom.

Bitcoin jest systemem o architekturze peer-to-peer, który działa podobnie jak protokół BitTorrent [16] z tą różnicą, że jedynym plikiem, który zostaje udostępniany, jest baza danych blockchain, w której zawarte są wszystkie transakcje wykonane od początku istnienia sieci. Transakcje, które trafiają do systemu, są sprawdzane przez każdy węzeł, a następnie zostają umieszczone w blokach, które dodawane są do blockchainu co około 10min. Czas ten został ustalony w pracy Satoshi Nakamoto *“Bitcoin: A Peer-to-Peer Electronic Cash System”* [1] i pociąga za sobą bardzo ciekawe następstwa. Każdy węzeł, który zechce wziąć udział w wydobyciu, czyli kopaniu nowych bitcoinów, dostaje do rozwiązania zagadkę matematyczną nazwaną *“Proof-of-Work”,* polegającą na dodaniu do nagłówka bloku takiej wartości, która za pomocą funkcji skrótu SHA-256 [17] wygeneruje pierwsze *n* bitów o wartości 0. Dopóki funkcja SHA-256 nie zostanie skompromitowana, algorytm kopania pozostaje w zbiorze NP-trudnym. Zachęta w postaci nagrody dla osoby, który znajdzie rozwiązanie zagadki i opublikuje je w sieci, doprowadziło do rozwinięcia się ogromnego przemysłu wydobywczego, który bazuje na mocy obliczeniowej, a ustalone średnie 10 min na nowy blok powoduje zwiększenie się poziomu trudności rozwiązania zagadki, przez zwiększenie wymaganej liczby zer, na które godzi się większość uczestników sieci. Ten sposób wprowadzania nowych “monet” do obiegu eliminuje potrzebę istnienia banku centralnego. Ponadto w protokole zapisane jest, aby średnio co 4 lata nagroda za wydobycie bloku spadała o połowę, z początkową nagrodą o wartości 50 BTC [18]. Skutkiem tego Bitcoin jest walutą deflacyjną, której wydobycie zakończy się w roku 2140 [19], a maksymalna możliwa ilość monet w obiegu wynosi dokładnie 21 mln BTC, zgodnie ze stałą *CAmount MAX\_MONEY w* kodzie źródłowym Bitcoina [20].

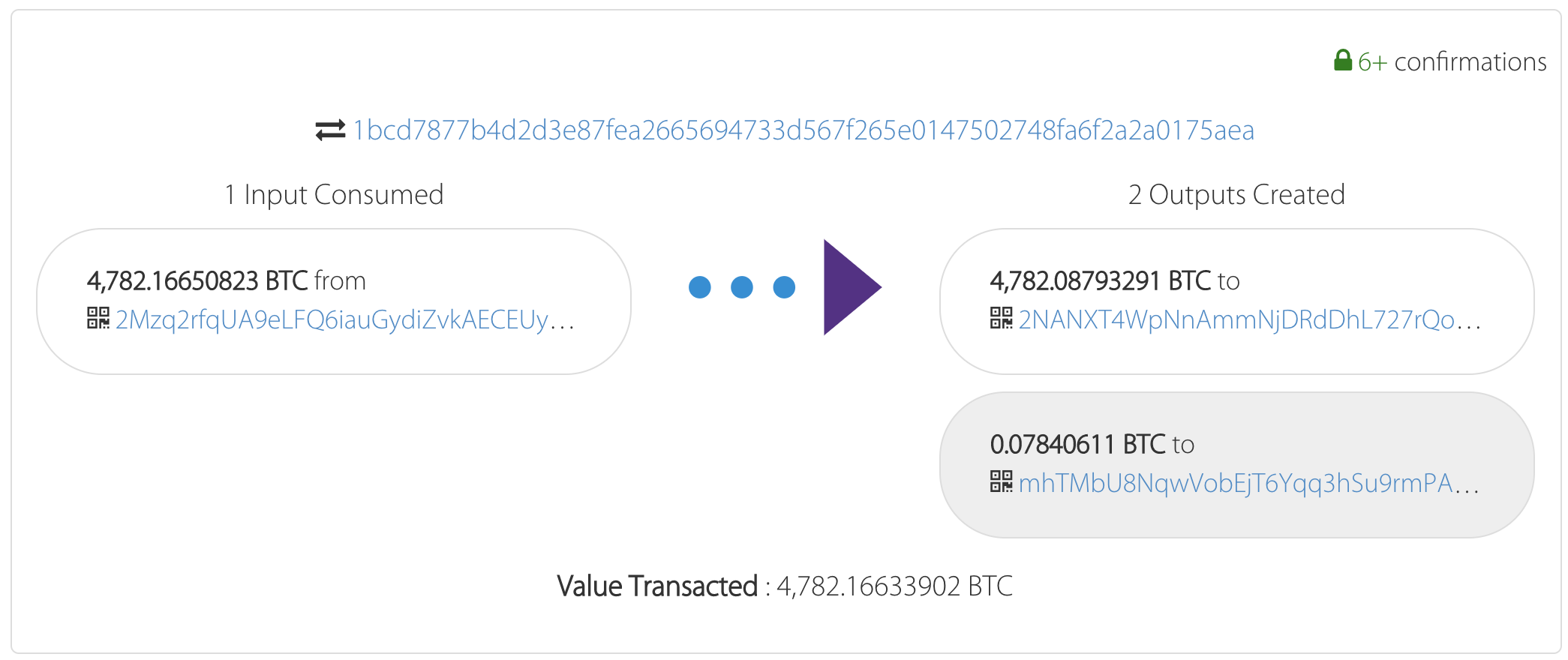
#### 2.1. Transakcje

*Stanisław Barański*

Transakcje w sieci Bitcoin pozwalają na przelewanie środków pomiędzy użytkownikami. Transakcją może być zarówno prosty bezpośredni przelew, jak i złożona operacja nazywana skryptem. W tym rozdziale przyjrzymy się dokładniej jak zbudowane są oba rodzaje transakcji.

Zanim opiszemy zasadę budowy transakcji warto zauważyć, że jednostką używaną w transakcjach jest Satoshi - nazwa pochodzi od twórcy systemu Bitcoin. 100 000 000 Satoshi jest równoważne 1 BTC [21].

Każda transakcja musi zawierać przynajmniej jedno źródło środków (Input) oraz przynajmniej jeden cel (Output), na który mają zostać przelane środki. Dodatkowo, każde źródło wskazuje na cel, z którego wychodzi. W ten sposób całość tworzy nieprzerwany łańcuch, który można zweryfikować, posiadając pełną historię sieci Bitcoin. Środki, które nie zostały nigdy wydane, nazywane są jako Unspent Transaction Output (UTXO). Za pomocą zsumowania wszystkich transakcji UTXO, możemy sprawdzić stan konta danego portfela. Aby ten mechanizm mógł działać efektywnie, uzgodniono, że środki zgromadzone na portfelu muszą zostać zawsze przelewane w całości, więc jeśli stan naszych UTXO wynosi 10 BTC, a przelać chcemy jedynie 8 BTC, musimy określić dwóch odbiorców transakcji - pierwszym jest adres docelowego odbiorcy, a drugim nasz własny adres. Różnica źródeł i celów, stanowi opłatę dla górników, którzy potwierdzą daną transakcję. Dokładny opis działania opłat został opisany w rozdziale 2.2.



Rys 2.1. Wizualizacja przykładowej transakcji przez portal https://live.blockcypher.com

Dodatkowo każda transakcja składa się z 4-bajtowego numeru wersji, który wskazuje każdemu węzłowi sposób, w jaki powinien ją zatwierdzić. Mechanizm ten pozwala na rozwój protokołu, bez konieczności zmiany poprzednich reguł walidacyjnych.

Aby uniknąć ograniczeń rozwoju protokołu Bitcoin, wprowadzono abstrakcyjny mechanizm zatwierdzania transakcji. Mechanizm ten pozwala na wprowadzanie nowych metod transakcji, bez konieczności zmieniania aktualnego protokołu. Aby to uzyskać, cele oraz źródła transakcji zabezpieczone są skryptami. Skrypt celu zabezpiecza dostęp do danych środków. Skrypt źródła zawiera argumenty, które pozwalają odblokować dany skrypt celu, a co za tym idzie, przelać środki.

Struktura źródła transakcji wraz ze skryptem przestawiona jest w tabeli 2.1, a struktura celu w tabeli 2.2.

Tabela 2.1. Struktura celu transakcji.

|  |  |  |
| --- | --- | --- |
| Rozmiar | Pole | Opis |
| 8 bajtów | Wartość | Wartość środków w jednostce Satoshi (10-8 BTC) |
| Zmienny | Skrypt zabezpieczający | Skrypt, który definiuje wymogi do spożytkowania danych środków |

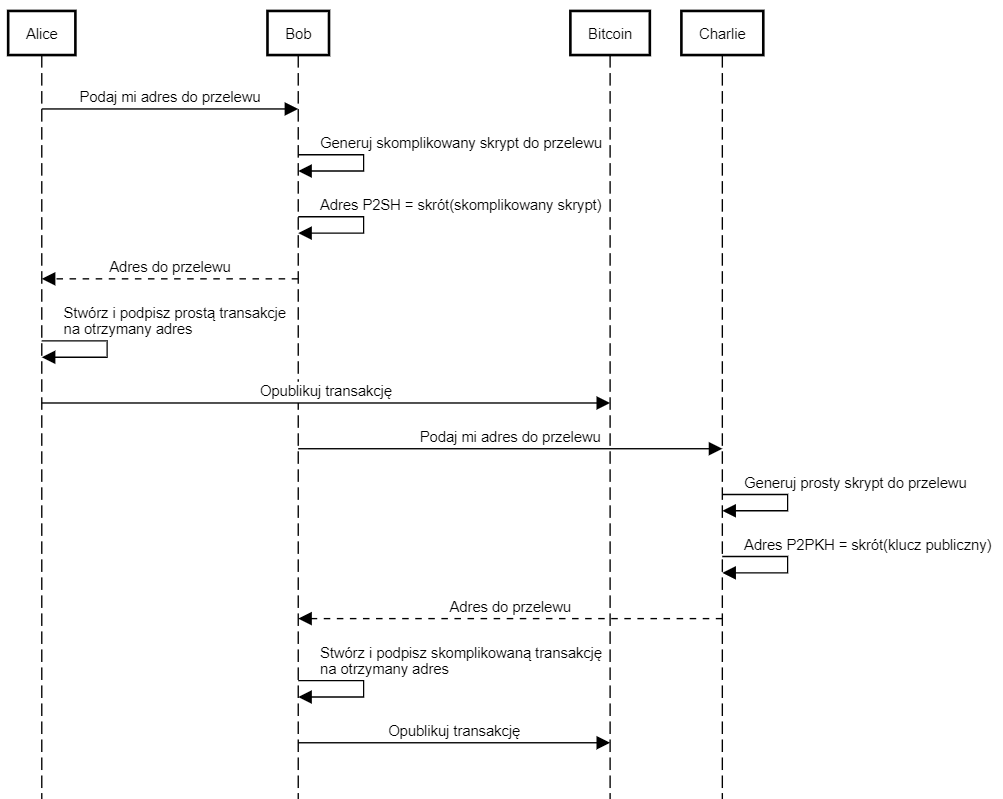
Tabela 2.2. Struktura źródła transakcji.

|  |  |  |
| --- | --- | --- |
| Rozmiar | Pole | Opis |
| 32 bajty | Skrót transakcji | Wskaźnik na transakcję, z której pochodzą środki |
| 4 bajty | Indeks celu | Indeks wskazujący, z którego celu wypływa dane źródło |
| Zmienny | Skrypt odblokowujący | Skrypt, który jest argumentem do spełnienia wymogów skryptu celu |

Najprostszy skrypt zwany ScriptPubKey lub P2PK (Pay-to-PubKey) pozwala na przelanie środków na klucz publiczny odbiorcy. Skrypty zapisane są w języku podobnym do Forth [22], bazującym na operacjach na stosie. Skrypt celu transakcji definiuje wymogi, które muszą zostać spełnione, aby dane środki zostały przelane. Najpopularniejszym aktualnie skryptem jednak jest P2PKH (Pay-to-PubKey-Hash), który pozwala na zmniejszenie rozmiarów transakcji, przez zastosowanie adresów w formie skrótów klucza publicznego za pomocą funkcji RIPE-MD. Adres taki wynosi wynosi 160 bitów zamiast pierwotnych 256, co jest bardzo pożądane ze względu na to, że każdy pełny węzeł musi przechowywać całą historię transakcji od roku 2009. Zatem skrócenie adresów o ponad ⅓, powoduje zmniejszenie ilości danych do przechowywania. Szczegóły implementacji skryptu P2PKH oraz jego dokumentację, można znaleźć w załączonych źródłach [23].

Różne typy skryptów spowodowały pewną komplikację, a rozwiązaniem tego problemu jest P2SH (Pay to Script Hash), który pozwala na stworzenie skryptu, który następnie wskazuje na skrót kolejnego już dowolnego skryptu. W ten sposób to odbiorca środków, poprzez przekazanie nadawcy adresu w postaci P2SH, definiuje, jaki skrypt chciałby użyć do zatwierdzenia przelewu. Dla przykładu: kiedy Alice chce przelać Bobowi pewną ilość środków, prosi Boba o przekazanie jej adresu. Bob generuje dowolnie skomplikowany skrypt i przekazuje jego skrót Alice jako adres. W tym momencie to Bob wie jak wygląda skrypt i jak spełnić jego wymogi w ScriptSig.

Rysunek 2.2 przedstawia scenariusz, w którym Alice przelewa środki Bobowi, które następnie Bob przelewa do Charlie. Jedyną osobą, która jest świadoma zastosowanych skomplikowanych skryptów, jest Bob. Dzięki zastosowaniu mechanizmu P2SH reszta uczestników traktuje te operacje jak zwykłe przelewy na skrót klucza publicznego.



Rys. 2.2. Sekwencja przedstawiająca przesłanie środków od Alice do Boba a następnie do Charlie.

MultisigScript, oraz jego wariacja M-of-N MultisigScript są najpopularniejszymi przykładami skryptów, które wykorzystują możliwości, jakie daje P2SH. Szczegóły działania tych skryptów zostały opisane w załączonych źródłach [24]. Warto zauważyć, że P2SH pozwala na to, żeby pośrednicy w transakcji nie musieli być świadomi tego, jakie warunki muszą zostać spełnione, aby się ona finalnie zakończyła. Dodaje to warstwę abstrakcji, która uogólnia całość.

#### 2.2 Opłaty

*Stanisław Barański*

Mechanizm opłat transakcyjnych został wprowadzony w dwóch celach. Pierwszym z nich jest obrona przed atakiem typu DoS (Denial of Service), który byłby możliwy do przeprowadzenia, przez zalanie sieci masą bardzo małych przelewów. Kolejnym powodem jest wprowadzenie mechanizmu wynagrodzenia górników po roku 2140, kiedy to wszystkie bitcoiny zostaną wydobyte i jedyną motywacją do przetwarzania bloków, staną się opłaty transakcyjne. Nadawca transakcji może ustalić dowolną stawkę opłaty za przelew, który chce wykonać. Każdy górnik z własnej chciwości stara się zdobyć jak najwięcej opłat z transakcji, które przetwarza w ramach bloku. Z racji, że rozmiar bloku jest ograniczony do 1 MB, naturalnym jest, że górnicy wybierać będą transakcje, które dają im jak największy zysk. Powoduje to sytuację, w której wartość opłaty jest wyznacznikiem czasu zatwierdzenia przelewu. Teoretycznie, jeśli obciążenie sieci jest niskie, to nawet zerowa opłata pozwolić zrealizować transakcję w najbliższym bloku, jednak wersja Bitcoin Core 0.9 wprowadziła minimalną opłatę w wysokości 1000 Satoshi. W praktyce stosuje się dynamiczną opłatę, która bazuje na aktualnym stanie obciążenia sieci, uwzględniając przy tym minimalną opłatę.

Wartość opłaty, którą deklarujemy się zapłacić, określamy przez różnice źródeł i celów w transakcji. Dlatego jeśli posiadamy źródło o wartości 1 BTC i chcemy przelać całą wartość, określając w źródle również 1BTC, to niestety, ale taka operacja się nam nie uda. Wartość celu musi zostać pomniejszona o wartość opłaty. Należy również pamiętać, że jeśli mamy źródło o wartości 10 BTC i chcemy z niego wydać jedynie 1 BTC, to określenie jedynie celu o wartości 1 BTC spowoduje, że 9 BTC trafi do szczęśliwego górnika, który taką transakcję przetworzy.

#### 2.3 Węzły w sieci Bitcoin

*Stanisław Barański*

Węzły sieci Bitcoin mogą pełnić różne role w zależności od wymagań, jakie się im stawia. Pełny węzeł pełni 4 role zobrazowane na rysunku 2.2.

Portfel

Wallet

Blockchain

Miner

Network Routing

Rys. 2.2. Role pełnego węzła

Funkcjonalność Network Routing (trasowania) pozwala węzłowi na braniu udziału w sieci P2P Bitcoina. Węzły porozumiewają się zgodnie z protokołem Bitcoin, wymieniając się transakcjami. Rola Wallet - kolorem zielonym - (portfel) pozwala przechowywać klucze prywatne, a do tego tworzyć oraz podpisywać transakcje. Na rysunku rola Blockchain (łańcuch bloków) pozwala na przechowywanie pełnej bazy danych transakcji, która umożliwia ich samodzielną walidację. Wielkość łańcucha blockchain, który nieustannie rośnie w tempie liniowym (w październiku 2018 miał rozmiar 192GB), jest czynnikiem, który skutecznie blokuje korzystanie z tej roli w wielu węzłach. Niektóre węzły przechowują jedynie część blockchaina, a dokładniej nagłówki bloków. Rola Miner (górnik), jest rolą, z której korzystają węzły, chcące poświęcić swoją moc obliczeniową na rzecz utrzymywania sieci: przetwarzając transakcje, budując nowe bloki i najważniejsze, rozwiązując algorytm Proof-of-Work.

#### 2.4 Simplified Payment Verification

*Stanisław Barański*

Rozwiązanie, z którego korzystaliśmy w naszym projekcie, nazywane lekkim węzłem lub Simplified Payment Verification (SPV), jest najczęściej spotykanym węzłem na urządzeniach z ograniczonymi zasobami, takimi jak np. smartfon, Raspberry Pi, a nawet komputer PC. Z racji, że węzeł taki nie przechowuje całej historii transakcji, nie jest w stanie sam określić, czy transakcja, którą otrzymał, jest poprawna lub czy nie została już wcześniej potwierdzona. Musi ufać innym pełnym węzłom, do których jest podłączony, co powoduje obniżenie bezpieczeństwa takiego węzła. Jedynym pocieszeniem jest to, że nasz węzeł nie akceptuje nowych połączeń, a jedynie korzysta z predefiniowanych serwerów. Dodatkowo biblioteka pozwala na śledzenie ilości potwierdzeń od węzłów o nadaniu takiej transakcji - jeśli ta liczba jest duża, to z dużym prawdopodobieństwem możemy stwierdzić, że transakcja jest poprawna. Więcej informacji o bezpieczeństwie modelu SPV jest dostępnych w dokumentacji biblioteki BitcoinJ [25].

#### *2.5. Kopia zapasowa portfela - BIP39*

*Stanisław Barański*

W przypadku naszego projektu, bardzo trudno jest zaprojektować system, który odporny będzie na kradzież lub zniszczenie. W teorii możemy założyć, że nie uwzględniamy tego przypadku i jest on w pełni zależny od użytkownika. Jednak w praktyce scenariusz utraty portfela, jest dosyć prawdopodobny. Dlatego zaczęliśmy szukać rozwiązania tego problemu, przez “zapamiętanie” ziarna portfela lub zapamiętanie samego sposobu, jak to ziarno uzyskać.

Bitcoin, jako kryptowaluta stworzona dla wszystkich ludzi - nie tylko informatyków, dba o przyjazność zawiłych aspektów kryptografii, oraz stara się eliminować błędy natury ludzkiej. Postulat BIP39, który dziś jest już standardem w branży kryptowalut, zaproponował stworzenie standardowej listy 2048 angielskich słów łatwych do zapamiętania, służących do generacji deterministycznego ziarna głównego, którego pochodnymi są klucze prywatne dla różnych kryptowalut. Jakikolwiek błąd we wpisanym słowie jest łatwy do wychwycenia i poprawienia. Na podstawie dwunastu wybranych słów mnemonicznych, możemy stworzyć funkcją PBKDF2-HMAC-SHA512 ziarno główne o długości 512 bitów, które następnie służyć nam będzie do generowania kluczy prywatnych. Taka reprezentacja ziarna głównego jest łatwiejsza do zapamiętania i przepisywania niż ciąg losowych, alfanumerycznych znaków. Jeśli ktoś posiada na danym adresie znaczną sumę kryptowalut, zaleca się zapamiętać ten ciąg, zamiast go przechowywać w wybranym miejscu, gdyż wiąże się to z niebezpieczeństwem kradzieży.



Rys. 2.3. Kopia zapasowa w postaci szablonu papierowego i metalowego

W dzisiejszych czasach BIP39 jest już standardem, a większość portfeli fizycznych dołącza dodatkowo w zestawie papierowe lub metalowe szablony, które należy wypełnić swoimi słowami. Widoczne są one na rysunku 2.3.

Metodę tę zaimplementowaliśmy w naszym projekcie, w postaci odtwarzania ziarna na podstawie podanych dwunastu słów mnemonicznych, zgodnie z BIP39. Dzięki temu, użytkownik jest w stanie odzyskać ziarno ze skradzionego/zgubionego portfela. Dodatkowo umożliwi to użytkownikom obsługiwanie swoich środków na innych dostępnych portfelach kryptowalut.

Istnieją alternatywne rozwiązania problemu zabezpieczenia się przed fizyczną utratą portfela. Metody te bazują głównie na sprytnych metodach zapamiętania głównego ziarna, wykorzystanego do wygenerowania klucza prywatnego portfela. Mają one mało wspólnego z systemami komputerowymi, a zatem i z naszym portfelem, dlatego nie zostaną one omówione.

### 

### 3. Wykorzystane technologie

*Piotr Pabis*

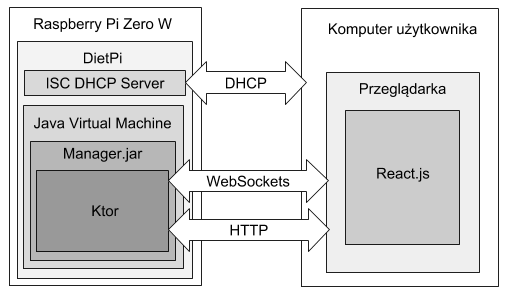
W naszej implementacji postanowiliśmy korzystać z powszechnie używanych technologii. Podstawą działania naszego projektu jest mikrokomputer Raspberry Pi Zero W, posiadający łączność 802.11 oraz port micro USB, umożliwiający podłączenie urządzenia w trybie karty sieciowej. Na karcie pamięci w Raspberry Pi zainstalowana została dystrybucja systemu operacyjnego GNU/Linux – DietPi [26], cechująca się małym narzutem zasobów. Główna aplikacja uruchomiona zostaje po starcie systemu jako serwis na maszynie wirtualnej Javy. Proces instalacji systemu DietPi jest dosyć prosty - po pierwszym włączeniu i połączeniu się z urządzeniem, zostaniemy przeprowadzeni przez interfejs, który pozwoli prawidłowo skonfigurować system. Szczegółowe instrukcje na temat instalacji zawarliśmy w naszym projekcie w formie osobnego repozytorium, w którym skupiamy informacje dotyczące instalacji naszego produktu na różnych platformach. Użytkownik może zainstalować nasz portfel na innym urządzeniu niż to, które posłużyło nam jako prototyp.

Większość projektów ukierunkowanych na urządzenia z ograniczonymi zasobami, stawia na jak najszybszy język i jak najlepszą optymalizację kosztem wygody programistów i końcowego użytkownika. W naszym projekcie chcieliśmy uniknąć tego problemu. Jesteśmy świadomi, iż zaimplementowanie tak bogatej funkcjonalności w języku kompilowanym do języka maszynowego byłoby ogromnym wyzwaniem, nie mówiąc o utrzymaniu takiego produktu i wprowadzaniu w nim zmian. Dodatkowo sami nie chcielibyśmy zajmować się technologiami, z którymi nie wiążemy naszej przyszłości. Dlatego postawiliśmy na języki kompilowane do bytecodu JVM [27], czyli Java i Kotlin [28], oraz JavaScript i TypeScript [29] w części projektu poświęconej interfejsowi użytkownika. Prawdopodobnie z tego powodu straciliśmy część zasobów na narzuty, które niesie za sobą dodatkowy poziom abstrakcji JVM, ale z kolei zyskaliśmy dużo łatwiejszy do późniejszego rozwoju kod, pozbawiony błędów w zarządzaniu pamięcią, a do tego z potencjałem na pełną wieloplatformowość. System został przetestowany na Raspberry Pi Zero W, ale ze względu na brak przeprowadzonych odpowiednich testów nie możemy zagwarantować, że będzie działał na innych urządzeniach. Jednak nic nie stoi technologicznie na przeszkodzie, aby uzyskać taką możliwość.

W celu sprawnego prototypowania użyliśmy istniejących bibliotek i frameworków. Dzięki oparciu swoich projektów o system kontroli budowania Maven, łatwo było rozdzielić duży projekt na podprojekty. Ponadto Maven podczas kompilacji sam pobiera i łączy niezbędne biblioteki, które wcześniej zostały określone w jego konfiguracji. Serwer HTTP powstał z użyciem frameworku Ktor [30], który działa z językiem Kotlin.

Frontend, czyli warstwa prezentująca się użytkownikowi w przeglądarce, oparty jest o framework React [31]. Korzysta on również z technologii WebSockets [32], w celu szybszej i co najważniejsze dwukierunkowej komunikacji pozbawionej niepotrzebnego narzutu żądań HTTP między przeglądarką a serwerem Ktor.

Na rysunku 3.1 został zobrazowany stos oprogramowania użytego w projekcie portfela, z wyłączeniem technologii użytych w modułach.



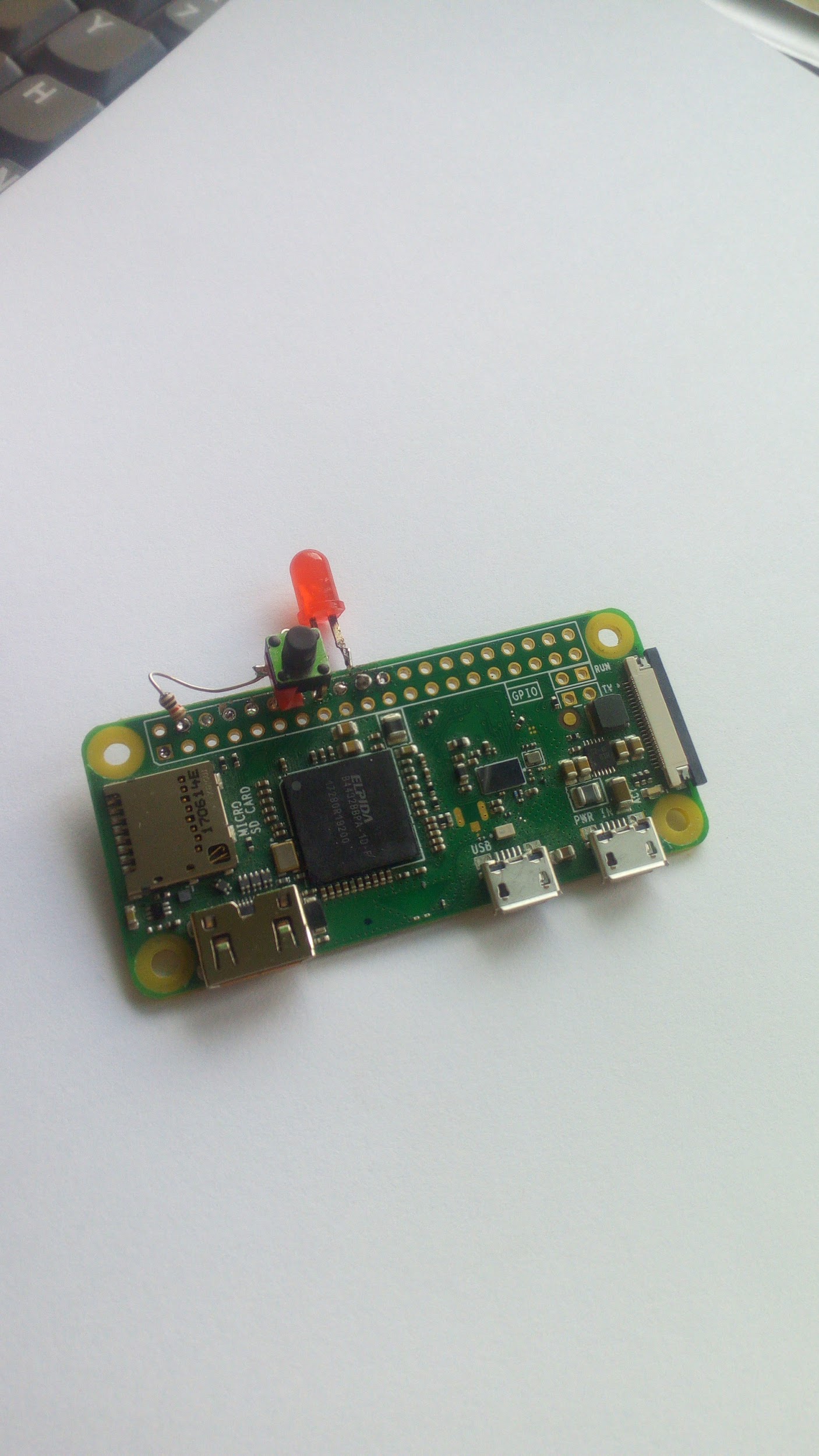
Rys. 3.1 Stos oprogramowania użytego w portfelu

Użytkownik, podłączający nasz portfel do swojego komputera. powinien móc połączyć się z konkretnym adresem IP. Nadaje mu go serwer DHCP [33] na urządzeniu, który swojej lokalnej karcie nadaje stały adres: *10.7.7.2*. Wybranym przez nas serwerem jest ISC DHCP Server [34]. Ponadto dla wygody zainstalowany jest też serwer mDNS – Avahi [35], rozgłaszający adresy zgodnie ze standardem Bonjour. Dzięki temu możliwym jest połączenie się z portfelem poprzez wpisanie adresu *wallet.local.* W przypadku systemu Windows, konieczna jest instalacja Apple Bonjour lub innego kompatybilnego klienta.

#### 3.1. Urządzenie Raspberry Pi Zero W

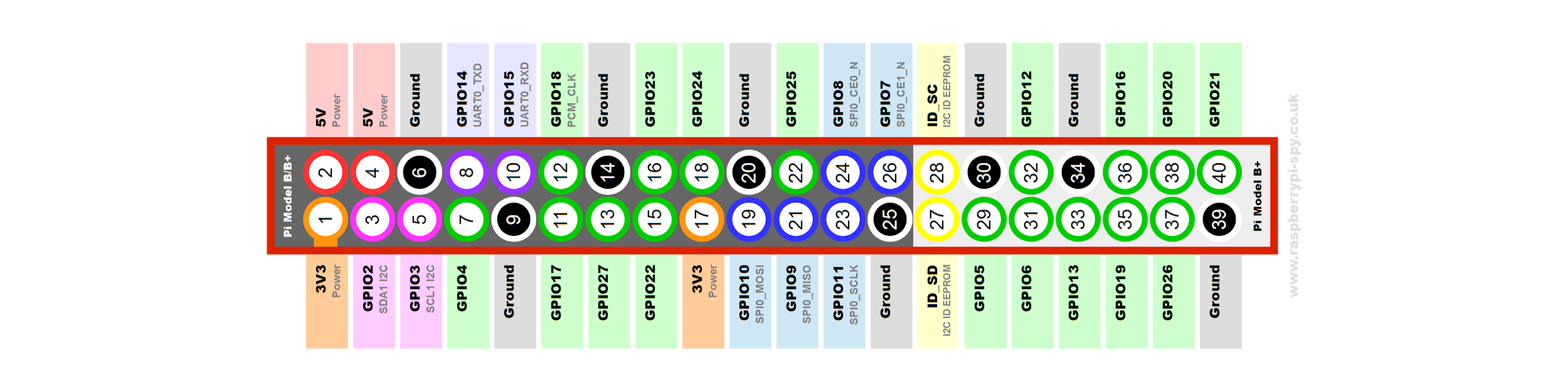
*Piotr Pabis*

Raspberry Pi Zero W (rysunek 3.2) wyposażony jest w jednordzeniowy procesor ARM BCM2835 o taktowaniu 1GHz, 512 MB pamięci RAM oraz łączności 802.11n i Bluetooth 4.1, a cały system operacyjny wczytywany jest z karty microSD. Z powodu tak ograniczonej wydajności, ważną kwestią jest optymalizacja pod względem szybkości.



Rys. 3.2 Urządzenie Raspberry Pi Zero W z przyciskiem modułowym i diodą stanu.

Ponadto urządzenie to posiada interfejs GPIO - wyjścia, których stan może być zbadany lub zmieniony przez oprogramowanie. Rozkład pinów GPIO w Raspberry Pi Zero przedstawia rysunek 3.3. Na zielono zaznaczone są piny nieużywane. Inne zazwyczaj służą do tworzenia takich portów jak np. konsola szeregowa UART, interfejsy SPI czy I2C.



Rys. 3.3. Rozkład pinów GPIO i ich zastosowania [36]

##### 3.1.1. Pomiary prędkości i optymalizacja

Podczas wyboru platformy byliśmy świadomi, iż wybór Raspberry Pi Zero W, będzie wymagał wprowadzenia pewnych optymalizacji systemu. Największą optymalizacją, którą oferuje sama sieć Bitcoin, jest zastosowanie ograniczonego węzła SPV, opisanego w rozdziale 2.4. Inne wprowadzone optymalizacje zostały opisane w tym rozdziale.

Procesor klasy ARM11 obecny w urządzeniu posiada możliwość przeliczania równoległego za pomocą rozszerzonego zestawu instrukcji NEON [37]. Jest to pomocne w przyspieszeniu działań kryptograficznych [38]. Poniżej, w tabeli 3.1, znajduje się porównanie prędkości wykonywania wybranych funkcji kryptograficznych przez procesor BCM2835 oraz wydany w roku 2016 Intel Celeron N3350 z AES New Instructions [39] ‑ zestawem instrukcji wspomagających sprzętowo wiele działań kryptograficznych. Pomiary zostały wykonane przy użyciu narzędzia *openssl speed* (z wyłączonym oszczędzaniem energii). Wynik to ilość przetworzonych milionów bajtów na sekundę.

Tabela 3.1. Pomiar szybkości działań kryptograficznych na zróżnicowanych urządzeniach

|  |  |  |  |
| --- | --- | --- | --- |
| *Funkcja* | Celeron N3350 + AES-NI | **BCM2835 + NEON Raspberry Pi Zero W** | BCM2837 Raspberry Pi 3 |
| MD5 (bloki 8192) [operacje / s] | 174 011,733 | **89 038,848** | 203 893,42 |
| AES-128-CBC (bloki 8192) [operacje / s] | 51 230,20 | **19 942,058** | 48 491,18 |

Po wykonaniu testów wydajnościowych stwierdziliśmy, iż wydajność obliczeniowa Raspberry Pi Zero W w zupełności wystarczy do wykonywania operacji w sieci Bitcoin.

Kolejną optymalizacją jest zmiana pakietu maszyny wirtualnej Java, z domyślnej implementacji OpenJDK na oficjalną wersję firmy Oracle. W tym celu również przeprowadziliśmy testy. Okazało się, że OpenJDK jest szybszy tylko w 9 na 24 przeprowadzonych testach.

Wybór dystrybucji systemu Linux miał bardzo duży wpływ na szybkość włączania się urządzenia. Naszym celem była możliwość włączenia portfela w czasie poniżej jednej minuty, który zdołaliśmy osiągnąć. Było to możliwe dzięki zastosowaniu dystrybucji DietPi, która charakteryzuje się znacznym ograniczeniem ilości instalowanych opcjonalnych paczek i modułów. Według programistów utrzymujących tę dystrybucję, udało się zmniejszyć wielkość obrazu trzykrotnie w porównaniu do dystrybucji Raspbian [40] - najpopularniejszej dystrybucji dla komputerów Raspberry Pi. Jednakże, w przyszłości chcielibyśmy stworzyć własną dystrybucję, która pozwoliłaby na wyłączenie wszystkich zbędnych modułów z naszego punktu widzenia. System ten nie jest jednak zoptymalizowany pod kątem szybkiego uruchamiania się, dlatego prawdopodobnie dałoby się to zmienić, wprowadzając zmiany, uwzględniające działanie jedynie naszej aplikacji portfela Raspberry Wallet.

#### 3.2. Bezpieczeństwo urządzenia Raspberry Pi Zero

*Piotr Pabis*

Głównym wymaganiem w naszym projekcie inżynierskim jest bezpieczeństwo. Z tego powodu należało skonfigurować środowisko uruchomieniowe DietPi, ze zwróceniem szczególnej uwagi na konfigurację zabezpieczeń oraz uprawnień.

##### 3.2.1. Użytkownik *wallet*

Aby odseparować naszą główną aplikację od reszty systemu operacyjnego, utworzony został oddzielny użytkownik *wallet*, z ograniczonymi uprawnieniami. Dzięki temu, w przypadku luki bezpieczeństwa w aplikacji, można uniknąć wielu niekorzystnych działań potencjalnego hakera, np. instalacji złośliwego oprogramowania czy przeglądania całej pamięci systemu.

Niektóre z komend wymagających wyższych uprawnień zostały ściśle określone w pliku *sudoers* (przykładowy plik w dodatku A). W ten sposób również nie ma potrzeby uruchamiania aplikacji jako użytkownik o najwyższych uprawnieniach - *root*. Wyższe uprawnienia zostały nadane grupie *netdev*, której członkiem jest użytkownik *wallet*, czyli m.in. sterowanie bezprzewodowymi kartami sieciowymi i klientem DHCP.

Sieć bezprzewodowa na stan dzisiejszy, wymaga zabezpieczenia poprzez WPA(2)-PSK. W celu zapewnienia poufności, nie jest możliwe połączenie z otwartym punktem dostępowym, ani siecią z kluczem WEP. Nie została zaimplementowana również możliwość podłączenia się do sieci Enterprise, ze względu na limit czasowy pracy inżynierskiej.

##### 3.2.2. Filtr sieciowy IPTables

Jako firewalla używamy IPTables z zestawu oprogramowania NetFilter. Reguły odblokowują tylko niezbędne porty po stronie interfejsu sieciowego przez USB jak i inne zakresy portów po stronie interfejsu karty bezprzewodowej. Ponadto stosowane są filtry zabezpieczające przed *zalewaniem* pakietami ICMP, atakami DoS oraz próbami włamywania się do serwisu SSH. Otwartymi dla połączeń przychodzących portami są:

* 5353 - rezolucja mDNS,
* 67 i 68 - serwer DHCP,
* 9080 - serwer HTTP,
* 9443 - serwer HTTP z TLS,
* 22 - SSH - w wersji produkcyjnej zostanie zamknięty.

Ponownie, aby nie musieć uruchamiać aplikacji portfela przez użytkownika *root*, który jako jedyny może otwierać porty mniejsze niż 1024, stosujemy prerouting przekierowujący przychodzący ruch na standardowe porty HTTP i HTTPS na wyższe, odpowiednio *9080* i *9443*. Dzięki temu użytkownik musi wpisać jedynie adres strony w przeglądarce, nie określając portu.

##### 3.2.3. Weryfikacja podpisu modułu

Archiwa JAR posiadają swój plik *MANIFEST.MF*. W pliku tym można znaleźć m.in. podpis archiwum, który po zmianie jakiegokolwiek pliku nie będzie się zgadzał. Pliki zostają podpisane kluczem prywatnym zawartym w certyfikacie, wystawionym przez nas dla deweloperów, za pomocą narzędzia *jarsigner*. W ten sposób główna aplikacja może zweryfikować prawdziwość i integralność archiwum modułów, przy użyciu publicznego certyfikatu. Dzięki temu użytkownik może zainstalować tylko sprawdzone moduły, co wyklucza ataki polegające na wstrzyknięciu złośliwego kodu do istniejącego modułu.

##### 3.2.4. Możliwości podniesienia bezpieczeństwa

Projektując sposób działania prototypu, musieliśmy wziąć pod uwagę ograniczony czas na ukończenie projektu. Z tego powodu pewne rozważone rozwiązania, nie zostały zgłębione lub po prostu nie mogły być fizycznie zaimplementowane.

Jednym z niebezpieczeństw, jakim musi przeciwdziałać nasz produkt, jest proste wyjęcie karty microSD z urządzenia Raspberry Pi. Urządzenie nie posiada mechanizmu szyfrowania dysku, ze względu na brak możliwości prostej implementacji tego rozwiązania, gdyż użytkownik nie jest w stanie wprowadzić hasła deszyfrującego dysk. Ponadto Raspberry Pi nie ma żadnego wspomagania sprzętowego do tego typu działań w kwestiach sprzętowego wspomagania szyfrowania dysku. Z dostępem do karty można podmienić wszystko, co znajduje się w systemie. Nie można natomiast wyczytać klucza prywatnego użytkownika ze względu na to, że jest on zarówno zaszyfrowany, jak i cała struktura bazy danych również ulega zaszyfrowaniu za pomocą hasła użytkownika, które jest podawane podczas korzystania z portfelem. Więcej na temat sposobu uwierzytelniania można znaleźć w rozdziale 4.1.

Kolejnym niebezpieczeństwem jest DNS spoofing. Rozważaliśmy podczas pierwszej połowy projektu inżynierskiego, zastosowanie DNSSec, chociażby do weryfikacji modułu serwera uwierzytelniającego 4.5.2. Na tym etapie prac, problem ten został pominięty, przez unikanie używania nazw i pozostanie przy adresach IP tam, gdzie jest to możliwe.

W rozdziale 5.1 opisany został również problem braku zegara czasu rzeczywistego i konieczności korzystania z serwera NTP.

### 

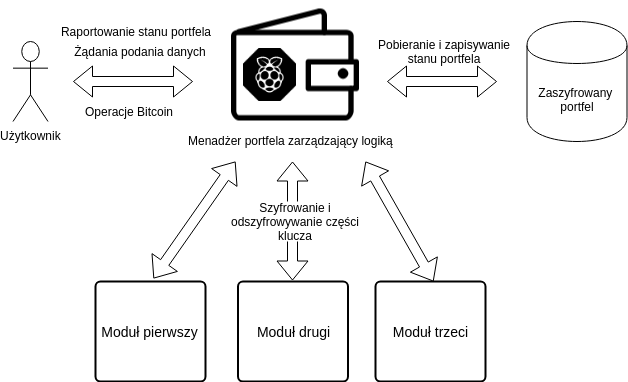
### 4. Architektura i moduły

*Patryk Milewski*

Podczas projektowania architektury naszego systemu, rozważaliśmy wiele możliwości. Naszym głównym celem jest uzyskanie rozwiązania elastycznego, zarówno pod względem bezpieczeństwa jak i wygody użycia, dlatego postawiliśmy na modularność. Takie podejście pozwala skonfigurować nasz produkt dokładnie tak, jak chce tego użytkownik. Dzięki temu jesteśmy w stanie dostarczyć produkt zarówno dla osoby, która przechowuje bardzo małe ilości waluty Bitcoin na swoim portfelu i nie jest narażona na duże ryzyko ich utraty, oraz dla użytkowników posiadających duże ilości kapitału na swoim portfelu i przez to będących głównym celem ataków.

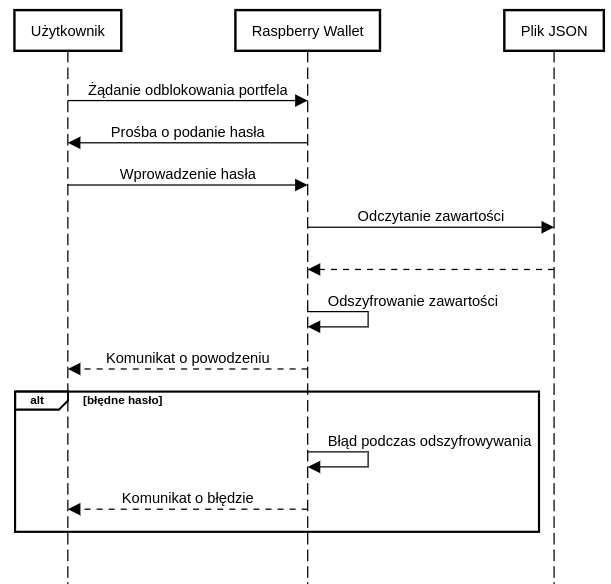
#### 4.1. Schemat architektury oraz analiza działania

*Patryk Milewski*



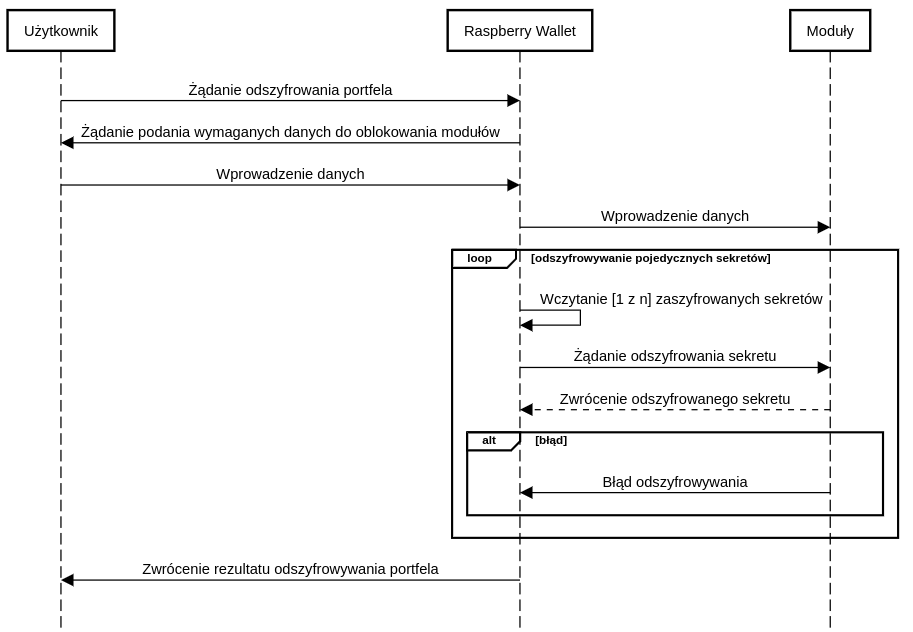
Rys. 4.1. Uproszczony schemat architektury

Na rysunku 4.1. zaprezentowany został uproszczony schemat architektury związanej z modularnością, oraz przepływem części klucza, który jest chroniony przez portfel. Łatwo można dzięki temu zobrazować bezpieczeństwo i skalowalność takiego rozwiązania. Zaszyfrowany portfel jest bezużyteczny bez modułów, które wymagają spełnienia pewnych warunków, aby odblokować ich części. Każdy moduł dodatkowo powinien być bezstanowy pod kątem danych, które umożliwiłyby go złamanie, co ma zapobiec możliwości kradzieży środków przez odczytanie pamięci wyłączonego urządzenia.



Rys. 4.2. Diagram sekwencji odblokowania portfela

Stan portfela zapisywany jest do pliku w postaci JSON. Podczas uruchomienia portfela, użytkownik musi podać hasło, które wykorzystywane jest do odszyfrowania zapisanego stanu portfela, na czas wykonywania na nim operacji. Obiekt, który jest zapisywany jako JSON i przechowuje stan całego systemu, w większości odpowiedzialny jest za przechowywanie zaszyfrowanych sekretów, które otrzymał od modułów. Dokładny przebieg sekwencji został zobrazowany na rysunku 4.2. Uwaga, odblokowanie portfela nie jest jednoznaczne z jego odszyfrowaniem.



Rys. 4.3. Diagram sekwencji odszyfrowywania całego portfela

Przebieg odszyfrowywania pojedynczych modułów został zobrazowany za pomocą diagramu sekwencji widocznego na rysunku 4.3. Nie zawiera on wszystkich szczegółów, jest jedynie ogólnym zobrazowaniem przebiegu zdarzeń podczas tej operacji. Jeżeli całość zakończy się sukcesem, czyli minimalna liczba wymaganych do odszyfrowania modułów zostanie odblokowanych, użytkownik uzyskuje dostęp do środków zgromadzonych w portfelu. Opis sposobu działania portfela w konfiguracji, która wymaga odszyfrowania N z M modułów, znajduje się w rozdziale 4.3.

Dzięki bezstanowości modułów, zablokowanie portfela jest bardzo proste. Po odebraniu żądania blokady portfela, nadpisywanie zostają wszystkie odszyfrowane sekrety z pamięci programu. Zaszyfrowany stan portfela znajdujący się w pliku JSON, nigdy nie jest zapisywany w postaci odszyfrowanej, dlatego nie ma potrzeby dokonywania w nim zmian.

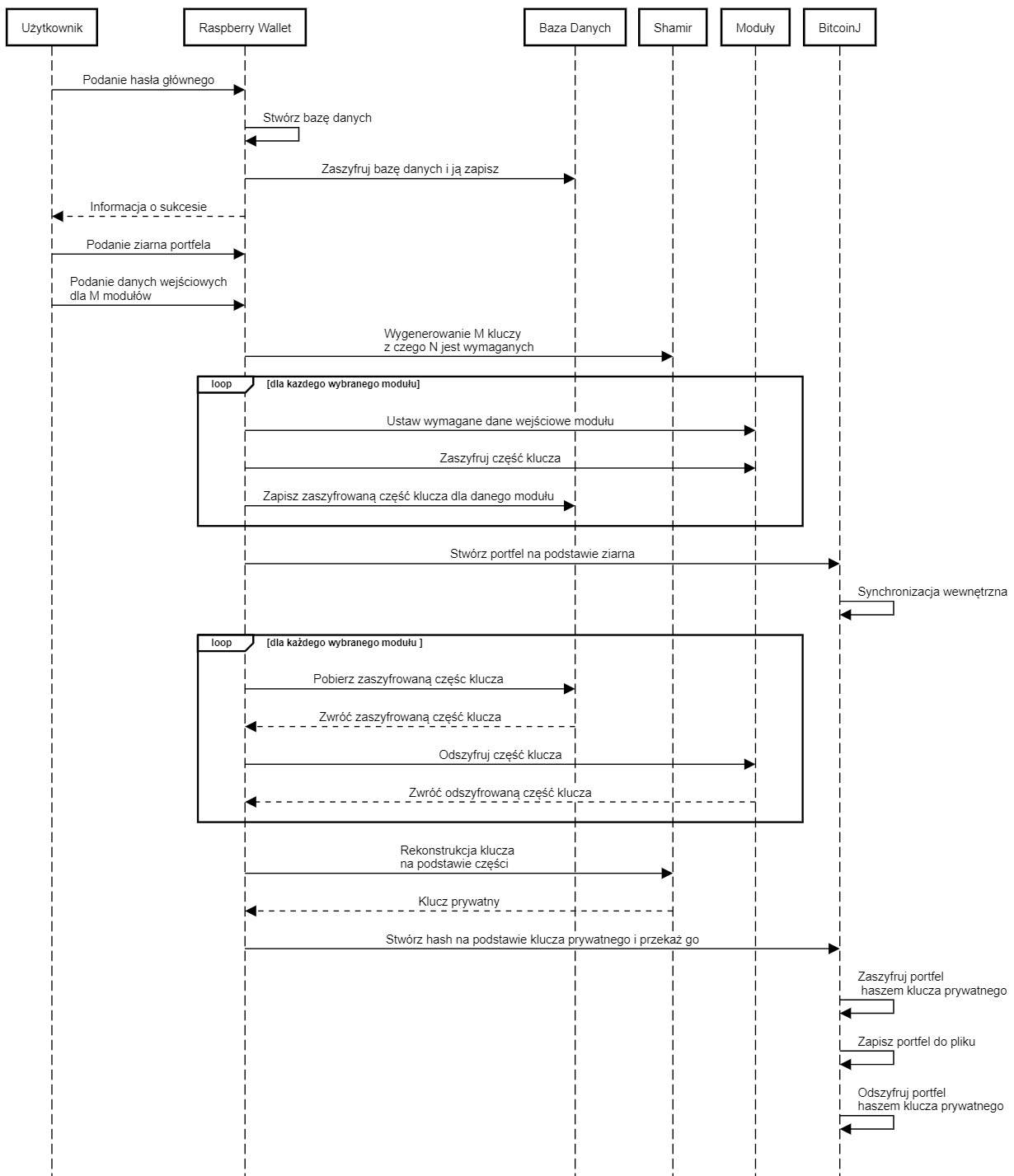
#### 4.2. Ochrona sekretu przez portfel

*Stanisław Barański*

Ze względu na złożoność działania naszego rozwiązania, w tym rozdziale zostanie omówiony przebieg inicjalnego szyfrowania klucza prywatnego portfela – zakładania portfela. Teoretycznie cały proces powinien być bardzo prosty i sprowadzać się do szyfrowania jednego klucza prywatnego. Jednak postęp w pracach zweryfikował negatywnie te założenia i dosyć mocno skomplikował rzeczywistą sytuację. Wynika to z dwóch głównych przyczyn:

* algorytm Shamira tworzy nowe sekrety, niezależne od sekretu pierwotnego, wynika to z działania algorytmu, szczegóły w kolejnym rozdziale 4.3,
* wykorzystana biblioteka BitcoinJ nie jest bezstanowa, co wymusiło pewne zmiany w całym systemie, które zostały szerzej opisane w rozdziale (5.2).

O skomplikowaności procesu inicjalizacji portfela, czyli sytuacji, w której użytkownik chce zabezpieczyć swój klucz prywatny, do momentu, w którym jest on finalnie zabezpieczony, może świadczyć długość diagramu sekwencji, widocznego na rysunku 4.4. Przebieg wszystkich operacji został opisany na diagramie.



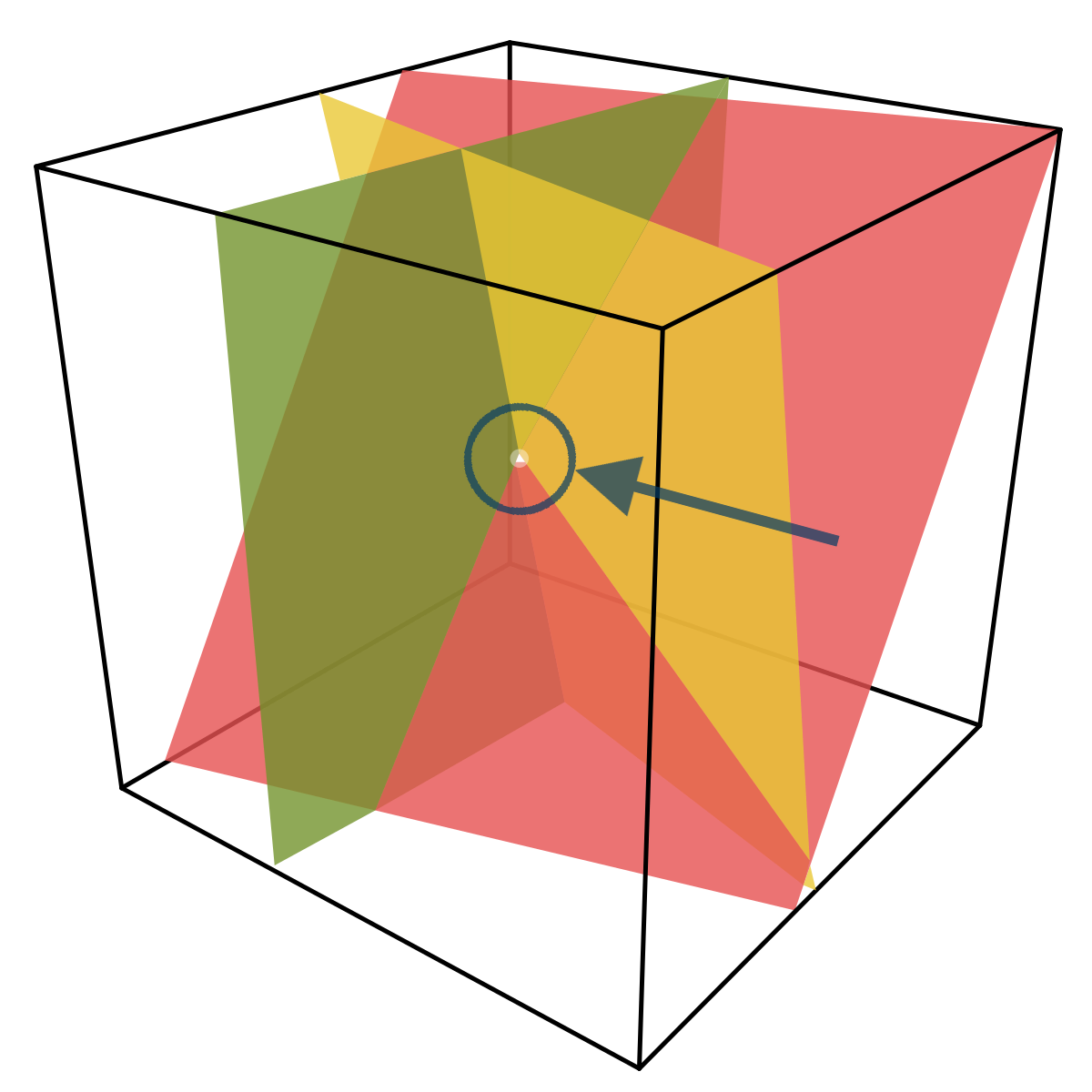
Rys. 4.4. Diagram sekwencji inicjalizacji portfela

Warto zauważyć fakt, że biblioteka BitcoinJ jest pewnego rodzaju portfelem, do którego finalnie trafia klucz prywatny użytkownika. Oprócz wykonywania operacji w sieci Bitcoin, zarządza ona również kluczem prywatnym, który został do niej przekazany. Wykorzystuje się go do szyfrowania ziarna portfela, podczas zapisu go na dysku pomiędzy transakcjami. Można powiedzieć, że w naszym portfelu znajduje się inny portfel, którego funkcjonalności zostały obudowane i znacząco rozbudowane.

#### 4.3. Dzielenie sekretu - algorytm Shamira

*Patryk Milewski*

Zaimplementowany mechanizm realizujący modularność, bazuje na algorytmie Adi Shamira [41]. Dzięki jego zastosowaniu, sekret, będący w postaci klucza prywatnego, może zostać podzielony na części pomiędzy nieskończenie wiele modułów. Aby odzyskać nasz główny sekret, potrzebne są wszystkie wymagane części. Jeżeli nie będziemy posiadać przynajmniej jednego z wymaganych sekretów, nasz główny sekret pozostaje tak samo bezpieczny, jak podczas posiadania zerowej wiedzy na temat któregokolwiek z sekretów.



Rys. 4.5. Zobrazowanie przecięcia się ”płaszczyzn sekretów” [42]

Zagadnienie to dobrze obrazuje rysunek 4.5. Sekretem, który został podzielony pomiędzy moduły, jest punkt przecięcia wszystkich płaszczyzn. Każda płaszczyzna powstała w wyniku podziału sekretu. Pomimo znajomości dwóch płaszczyzn, nadal mają one nieskończenie wiele punktów przecięcia z trzecią płaszczyzną, która jest nam nieznana. Dzięki temu wiedza na temat właściwego sekretu nie wzrasta, nawet po uzyskaniu dostępu do części sekretów, powstałych podczas podziału.

Warto też zauważyć, że części powstałe po podzieleniu, nie mogą być rozumiane, jako faktyczne części głównego sekretu. Obecny jest tutaj dodatkowy poziom abstrakcji, który tworzy nowe sekrety. Dzięki temu klucz, który posiada skończoną długość, możemy podzielić na nieskończoną ilość części, bez narażania się na jakiekolwiek ryzyko złamania go.

Dodatkową funkcjonalnością, wynikającą z użycia algorytmu Shamira, jest możliwość wymagania zadanej liczby sekretów, do odzyskania głównego sekretu. Wynika ona z działania samego algorytmu. Dzięki temu jesteśmy w stanie uzyskać konfigurację portfela typu 2 z 3, gdzie 2 odpowiada wymaganej liczbie modułów do odblokowania portfela, a 3 jest liczbą określającą ilość wszystkich modułów. Można w ten sposób realizować funkcjonalności takie jak:

* współdzielenie portfela pomiędzy podmiotami,
* zapasowe moduły, których odblokowanie jest trudniejsze, ale normalnie nie są wykorzystywane i istnieją, jako awaryjna forma odzyskiwania dostępu do portfela.

#### 4.4. Konfiguracja portfela oraz modułów

*Patryk Milewski*

Podczas uruchomienia portfela, powinien otrzymać on swoją właściwą konfigurację. Konfiguracja pozwala zmieniać podstawowe parametry portfela oraz modułów. Jest ona zapisana w postaci pliku YAML [43].

|  |
| --- |
| version: 0.5.0 base-path-prefix: /opt/wallet/ autolock-seconds: 60  bitcoin:  network: testnet  user-agent: RaspberryWallet  server:  keystore-name: RaspberryWallet.keystore  keystore-password: EXAMPLE\_PASSWORD  key-alias: ssl  port: 9080  secure-port: 9443  modules:  PinModule:  max-retry: 5   ExampleModule:  example: example  name: name |

Rys. 4.6. Przykład konfiguracji portfela w języku YAML

Ze względu na specyfikę języka YAML, może on być bezstratnie przekonwertowany na inne znane języki takie jak XML czy JSON. Zdecydowaliśmy się na takie rozwiązanie ze względu na bardzo dobrą czytelność, oraz łatwe odwzorowanie zagłębień w konfiguracji. Na rysunku 4.6. znajduje się przykład konfiguracji portfela. Warto zwrócić uwagę, na sposób konfiguracji modułów. Nazwa znacznika musi odpowiadać nazwie pakietu Java, który rozszerza abstrakcyjną klasę Module. Wartości konfiguracji modułu mapowane są bezpośrednio na wartości pól klasy, która odpowiada za konfigurację. Jej nazwa jest również stała i odpowiada nazwie modułu, gdzie dopisek “Module” zostaje zastąpiony przez “Config”. Przykłady poprawnie skonfigurowanych modułów można sprawdzić bezpośrednio w repozytorium [44].

Założyliśmy, iż użytkownik będzie sam dostarczał konfigurację do portfela w postaci pliku YAML. W implementacji naszego portfela, nie istnieje jeszcze interfejs, który by pozwalał na przeprowadzenie takiej operacji za pomocą web aplikacji, jednak można to osiągnąć małym nakładem pracy, podmieniając plik podczas instalacji.

#### 4.5. Moduły zaimplementowane w naszej architekturze

*Patryk Milewski*

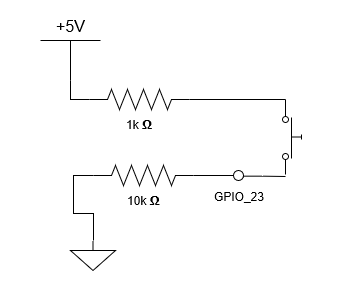
Poniższe moduły znalazły się w systemie jako przykłady. Naszym celem nie było zaimplementowanie dużej ilości zaawansowanych modułów, a stworzenie całego systemu jako podstawy, która z czasem będzie rozszerzana przez nowe rozwiązania.

Ze względu na dynamiczny rozwój projektu, dokładne wytyczne dotyczące implementacji modułów nie zostaną zamieszczone w tym dokumencie. Aktualna dokumentacja techniczna znajduje się w repozytorium w postaci skryptów w języku Bash, dokumentacji JavaDoc oraz README poszczególnych repozytoriów [45]. Znaczna część dokumentacji została zamieszczona w dodatku B.

##### 4.5.1. Przełącznik uwierzytelniający transakcję

*Patryk Milewski*

Moduł fizycznego przycisku potwierdzającego transakcje, ma za zadanie wymusić fizyczne posiadanie urządzenie przez użytkownika. Zostało to zrealizowane za pomocą GPIO wbudowanych w Raspberry Pi, które pozwalają na dwukierunkową komunikację z urządzeniami zewnętrznymi. Rozważaliśmy działanie tego modułu, pod kątem sposobu szyfrowania części sekretu. Różni się on od innych modułów, przez brak posiadania własnych informacji, które mógłby zostać użyte do zaszyfrowania otrzymanego sekretu. W przypadku fizycznej kradzieży portfela, czy dostania się do środka włączonego portfela, moduł ten jest możliwy do złamania. Dane, które trafiają do modułu przełącznika, nie są szyfrowane. Moduł jedynie zwraca błąd operacji odszyfrowania w przypadku, gdy przełącznik nie jest wciśnięty. Szczegóły tego problemy zostały opisane w rozdziale 5.8.



Rys. 4.7. Schemat instalacji przycisku modułowego.

Na rysunku 4.7. znajduje się schemat techniczny bezpiecznej instalacji przycisku. Rezystory zabezpieczają wejście GPIO przed przeciążeniem. Przycisk może zostać zamontowany do dowolnego, wolnego GPIO.

Częścią programową zajmuje się Java w połączeniu z biblioteką Pi4J [46]. Odczytywanie stanów wybranych GPIO i reagowanie na zmiany ich stanu jest dzięki temu bardzo proste. Pomimo prostoty i popularności biblioteki, natrafiliśmy na problemy, które zostały szerzej opisane w rozdziale 5.3.

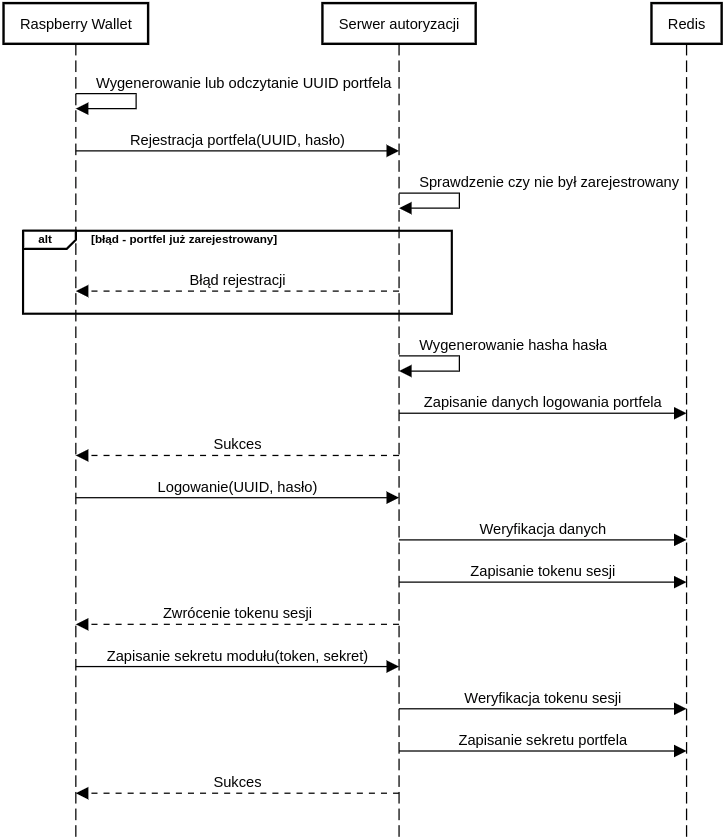
Niewątpliwą zaletą tego rozwiązania jest jego prostota oraz niska cena. Jednak największym plusem tego modułu, jest zablokowanie znacznej części scenariuszy ataków, które nie zakładają fizycznego dostępu do portfela.

##### 4.5.2. Uwierzytelniająca sieć serwerów

*Patryk Milewski*

Bardzo często spotykanym mechanizmem autoryzacji czynności, jest użycie serwera poświadczającego. Ze względu na poziom bezpieczeństwa, który takie rozwiązanie gwarantuje, postanowiliśmy wdrożyć to rozwiązanie do naszego systemu. Funkcję tą mają spełniać usługi dostępne przez REST API [47] serwera autoryzacji.

Serwer autoryzacji nie jest bezstanowym modułem, ze względu na szczególny przypadek, czyli praca na zewnętrznym środowisku, które jest odpowiedzialne za zabezpieczenie przed fizycznym dostępem osób trzecich. Dane modułu przechowywane są w formie bazy NoSQL typu key-value o nazwie Redis [48]. Dzięki takiemu podejściu, odpowiedzialność za organizowanie danych zrzucona jest na aplikację, która z niej korzysta. W przypadku tego modułu jest to korzystne, ze względu na bardzo prosty model danych, co znacznie przyspieszyło i uprościło implementację. Sposób przechowywania danych zostanie omówiony w kolejnych akapitach.



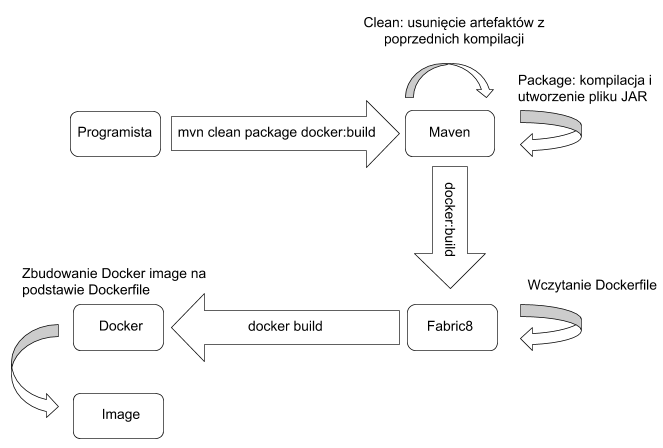
Rys. 4.8. Diagram sekwencji rejestracji nowego portfela

Podczas pierwszego połączenia się z serwerem autoryzacji, każdy portfel musi zarejestrować się, podając swój wygenerowany UUID [49] oraz hasło podane przez użytkownika. Aby dostać się do pozostałych funkcjonalności takich jak ustanowienie sekretu portfela, należy zalogować się, w celu uzyskania tokenu sesji, który ma ograniczony czas ważności i jest wygenerowanym UUID. Następnie za pomocą tokenu, portfel może ustanowić własny sekret po stronie serwera autoryzacji. Sekretem jest hasło, które będzie wykorzystywane do szyfrowania danych przekazanych do modułu. Generowane jest ono za pomocą pomocą bezpiecznego generatora wbudowanego w język Java, o nazwie SecureRandom [50]. Następnie hasło konwertowane jest do formatu Base64 [51]. Po wygenerowaniu, hasło trafia fo serwera autoryzacji, za pomocą bezpiecznego kanału SSL. Od tego momentu, każdy kto posiada kombinację odpowiedniego UUID i hasła lub ważny token sesji, jest w stanie odczytać zapisany sekret. Cały proces jest przedstawiony na diagramie sekwencji na rysunku 4.8. Nie obejmuje on alternatywnych przypadków, w których weryfikacja podanych danych nie zakończy się sukcesem.

Tabela 4.1. Opis danych przechowywanych w bazie Redis

|  |  |
| --- | --- |
| Klucz | Wartość |
| token:<UUID\_portfela> | Token w postaci wygenerowanego UUID |
| wallet:<UUID\_portfela> | Wygenerowany hash hasła portfela, potrzebny do zalogowania |
| salt:<UUID\_portfela> | Sól użyta podczas generowania hashu |
| secret:<UUID\_portfela> | Sekret portfela |

Tabela 4.1. opisuje schemat przechowywanych danych w bazie Redis. Ze względu na specyfikę bazy typu key-value, potrzebne było zapewnienie rozróżnialności danych, przez dodanie odpowiednich etykiet do kluczy. Każdy portfel posiada w bazie maksymalnie 4 rekordy. Bazy tego typu zaimplementowane są za pomocą struktury danych nazywaną mapą hashowaną, gdzie z każdego klucza obliczana jest wartość funkcji skrótu, który później jednoznacznie identyfikuje przechowywane dane. Dzięki temu wszystkie dane znajdują się w jednej kolekcji, a czas dostępu jest stały *O(1)*. Aby zapobiec kradzieży haseł, użyto algorytmu PBKDF2 (Password-Based Key Derivation Function 2) z HMAC (Hash-based Message Authentication Code) i funkcją pseudolosową SHA512. Dodatkowo została zastosowana 128 bitową sól, zwana również wektorem inicjalizującym, a funkcja skrótu zostałą skonfigurowana na 65000 iteracji, co znacznie spowalnia ataki typu brute-force. Całość zapewnia bardzo wysoki poziom zabezpieczenia haseł przed odtworzeniem ich pierwotnych wartości. Warto również wspomnieć, że Redis udostępnia mechanizm, który ustanawia czas życia danego klucza i wartości. Został on wykorzystany do określenia czasu ważności tokenów sesji.



Rys. 4.9. Schemat budowania Docker Image

Moduł został zaimplementowany w języku Java 8, użyto dodatkowo framework Spring Boot [52] służący, jako podstawa do wystawienia endpointów serwisu REST oraz dostarczenia mechanizmu Inversion of Control. Dodatkowo użyta została biblioteka Jedis [53], która pozwala na wykonywanie operacji na bazie Redis z poziomu kodu Java. Budowaniem projektu do postaci plików wykonywalnych JAR oraz zarządzaniem zależnościami, zajmuje się program Maven [54]. Tworzenie działającej instancji serwera autoryzacji jest zautomatyzowane i zajmuje się tym konteneryzacja Docker. Plugin ze środowiska Maven o nazwie Fabric8 [55], buduje obraz kontenera Docker na podstawie przygotowanego Dockerfile. Dockerfile jest zbiorem komend w języku Bash, które należy wykonać, aby otrzymać docelowy obraz kontenera Docker. Docker image budowany jest automatycznie razem z plikiem JAR, który jest w nim umieszczany. Operacja ta została zobrazowana na rysunku 4.9.

|  |
| --- |
| version: '3.5' services:  spring-app:  image: piwallet/authorization-server  ports:  - 8080:8080  - 8443:8443  networks:  - redis-net  depends\_on:  - redis   redis:  image: redis:4.0-stretch  hostname: redis  networks:  - redis-net  networks:  redis-net: |

Rys. 4.10. Konfiguracja Docker-compose

Po utworzeniu wymaganego obrazu, system uruchamiany jest za pomocą narzędzia Docker-compose. Ma ono za zadanie ułatwić zarządzanie grupą kontenerów, które składają się na jeden system. Moduł serwera autoryzacji składa się z dwóch kontenerów - pierwszym jest kontener Spring Boot z RESTful API, drugim kontenerem jest nierelacyjna baza danych Redis. Rysunek 4.10 przedstawia zawartość pliku docker-compose.yaml, który określa konfigurację całego systemu. Można w nim zdefiniować wszystkie parametry konfiguracji danego kontenera, oraz ustawić własne zmienne środowiskowe rozumiane przez nasz kontener. Dzięki takiemu rozwiązaniu wszystko znajduje się w jednym miejscu. Konfiguracja z pliku YAML jest następnie konwertowana na zwykłe polecenia Docker. Najważniejsza jest jednak możliwość uruchomiona całego systemu za pomocą jednej komendy: *docker-compose up -d.* Dodatkowo chcieliśmy wdrożyć ten moduł na platformę Heroku, która pozwoliłaby na stały dostęp do uruchomionej aplikacji. Zostało to szerzej opisane w rozdziale 5.6.

Docelowo moduł ma zostać rozproszony, aby zabezpieczyć się przed ataki typu DoS. Ze względu na użyte technologie, nie jest to dużym wyzwaniem. Technologia Redis wspiera klasteryzację i jest bardzo łatwa do konfiguracji. Z kolei ze względu na konteneryzację i przechowywanie stanu serwera autoryzacji tylko i wyłącznie w bazie danych, instancje aplikacji Spring Boot mogą być skalowane z użyciem load balancerów, które po prostu rozdzielą ruch pomiędzy instancje. Możliwe jest również zastosowanie technologii Docker Swarm, która pozwala zdefiniować wymaganą ilość działających instancji kontenera opisanego w pliku konfiguracyjnym Docker-compose. Szczegóły na ten temat można znaleźć w oficjalnej dokumentacji [56].

##### 4.5.4 Odblokowanie portfela za pomocą kodu PIN

Moduł ten jest bardzo prosty w implementacji i działaniu. Użytkownik musi wprowadzić wcześniej podany ciąg liczb, aby odblokować moduł. Ciąg wykorzystywany jest, jako hasło podczas szyfrowania części sekretu algorytmem AES.

W przypadku zbyt wielu nieudanych prób odblokowania portfela, zostaje on zablokowany na odpowiednio długi czas, aby nie doszło do prób odgadnięcia hasła za pomocą sprawdzenia wszystkich możliwości. Moduł ten jest w stanie również wymuszać długość PINu.

#### 4.6. Moduły odrzucone na etapie analizy i projektowania portfela

*Patryk Milewski*

Pomimo pomysłów na wiele modułów, niektóre z nich po przeprowadzeniu wstępnej analizy, musiały zostać odrzucone. Część modułów pomimo możliwości ich stworzenia, nie zostały zaimplementowane ze względu na ograniczenia czasu i zasobów. W tym rozdziale chcielibyśmy pokazać, jakie moduły rozważaliśmy, jakie były ich wady i zalety oraz dlaczego ostatecznie nie trafiły one do naszego systemu, gdyż czasami może to nie być aż tak bardzo oczywiste.

##### 4.6.1. Weryfikacja za pomocą telefonu

Popularną metodą uwierzytelnienia płatności w bankach, czy dostępu do kont, jest weryfikacja za pomocą kodu otrzymanego na telefon. Rozwiązań implementujących tę metodę jest wiele i różnią się one dla każdego systemu. W tym rozdziale omówione zostaną dwa główne sposoby: wysyłanie jednorazowego hasła przez sieć telefoniczną oraz potwierdzanie transakcji za pomocą prostej aplikacji dla systemu mobilnego.

Głównym problemem związanym z przesyłaniem kodów jednorazowych drogą telefoniczną jest ich koszt. Koszt wysłania jednego SMSa wynosi około 12 groszy [57] w przypadku korzystania z zewnętrznego usługodawcy, z którym można zsynchronizować się za pomocą API. Koszty te mógłby ponieść sam użytkownik, jednak wiązałoby się to z dodatkową konfiguracją, wymagającą podania formy płatności. Z kolei zrzucenie odpowiedzialności pieniężnej za ten moduł na jego autorów, naraziłoby ich to na duże koszty w przypadku wielu użytkowników. Dodatkowo potencjalny atakujący mógłby złośliwie wymuszać wysyłanie dużej ilości SMSów, co mogłoby spowodować zakłócenie w działaniu usługi, w przypadku np. braku środków na kolejne wiadomości. Rozważaliśmy również zaimplementowanie modułu, który będzie przyjmował klucz API operatora. Takie rozwiązanie jest bardzo proste i zrzuca całą odpowiedzialność na użytkownika i usługodawcę, dlatego chcielibyśmy je zaimplementować, jednak nie wystarczyło nam na to czasu.

Alternatywą jest wykorzystanie aplikacji mobilnej, która wykluczyłaby dodatkowe koszty. Co więcej mogłaby ona realizować funkcjonalności innych modułów, takich jak pochodne modułu korzystającego z kodów QR opisanego w rozdziale 4.6.2. Jednak należałoby rozważyć, czy dopuszczalne jest przeplatanie się modułów, które może doprowadzić do kompromitacji znacznej części systemu, w przypadku błędów w jednym z nich.

##### 4.6.2. Kody QR przechowujące część klucza

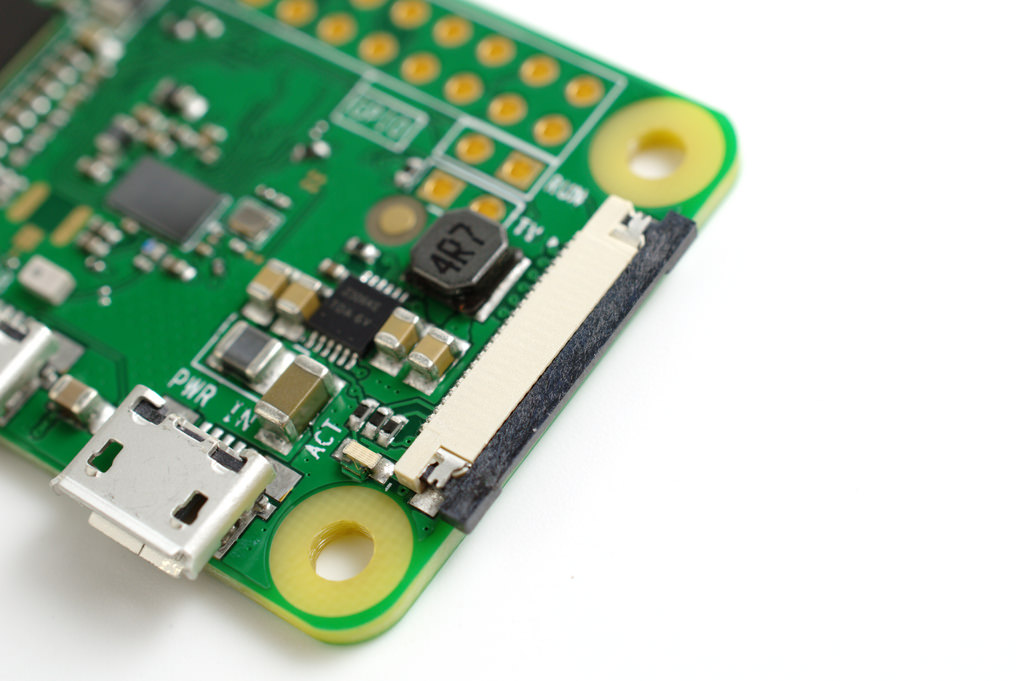
Kod QR może przechowywać dane w postaci graficznej. Dokładną specyfikację tego rozwiązania możemy znaleźć we wskazanej literaturze [58].



Rys. 4.11. Kod QR z 256 bitowym kluczem

Na rysunku 4.11 został przedstawiony przykład kodu QR dla klucza 256 bitowego, który najczęściej jest wykorzystywany podczas szyfrowania danych za pomocą algorytmu AES. W przypadku zapisywania danych w postaci takich kodów, użytkownik jest zmuszony do zabezpieczenia ich we własnym zakresie. Jest to analogiczna idea jak w przypadku papierowego portfela Bitcoin opisanego w rozdziale 1.3.4.

Pomimo pozornego skomplikowania ewentualnej implementacji takiego modułu, wykorzystanie bibliotek ograniczyłoby to do jedynie załadowania i przekazania odpowiedniej bitmapy, odpowiadającej zdjęciu z aparatu. Największym wyzwaniem jest uporanie się z problemami sprzętowymi, takimi jak dobranie odpowiedniej kamery, konfiguracja jej i integracja z resztą systemu.



Rys. 4.12. Złącze Camera Serial Interface (CSI) [59]

Urządzenia Raspberry Pi Zero W pozwala na montaż kamery za pomocą interfejsu CSI. Jest to szerokie złącze, znajdujące się na krawędzi płytki drukowanej, widocznej na rysunku 4.12. CSI jest standardem w urządzeniach Raspberry Pi, dlatego nie jest problem znalezienie kompatybilnych kamer, które można za jego pomocą podłączyć. Dekodowanie takiego kodu nie jest bardzo czasochłonne, gdyż procesor Qualcomm MSM7200 o mocy obliczeniowej 740 MIPS [60] potrafi dekodować około 15 ramek kodów QR na sekundę. Spodziewamy się podobnego wyniku na procesorze BCM2835 z mocą obliczeniową 850 MIPS [60]. Jest to całkowicie wystarczający wynik do efektywnego skanowania kodów, a nawet w przypadku osiągnięcia 5 ramek na sekundę, byłby on akceptowalny.

Moduł ten nie został zaimplementowany ze względu na dodatkowe koszty około 50 zł [61], które generuje kamera, oraz brak czasu na integrację rozwiązania. Planujemy dodanie dodanie modułu w późniejszych wersjach, niekoniecznie w jednej formie. Możliwe jest wykorzystanie do tego również smartfona, który przesłałby wyniki skanowania kodów do portfela za pomocą aplikacji mobilnej.

##### 4.6.3. Wykrywanie obecności urządzenia Bluetooth

Bluetooth, czyli standard komunikacji bezprzewodowej, obecny jest w wielu urządzeniach. Z założenia ma pozwalać na wymianę danych, jednak od wersji 4.0 jedną z jego dodatkowych funkcji jest wykrywanie bliskości urządzeń. Funkcja ta w większości przypadków bazuje na pomiarze siły sygnału urządzenia, którego odległość od nas chcemy określić. Istnieją urządzenia, takie jak iBeacon opracowane przez firmę Apple [62], czy Mi Band produkowane przez firmę Xiaomi, które realizują to w praktyce. Pierwszy służy głównie, jako system określania odległości od wielu beaconów, czyli latarni, służących za stały punkt odniesienia, za pomocą którego jesteśmy w stanie ustalić swoje położenie wewnątrz budynku. Drugie rozwiązanie jest z kolei bliższe temu, co sprawdziłoby się, jako moduł zabezpieczeń w naszym systemie. Mi Band jest opaską na rękę, która posiada wiele funkcji, w tym łączność poprzez Bluetooth ze sparowanym z nim urządzeniem. Może nim być telefon, komputer lub dowolne inne urządzenie obsługujące ten format komunikacji. Dzięki znajomości odległości od urządzenia, możemy ustanowić pewien próg, po przekroczeniu którego nasz telefon czy komputer, zostanie odblokowany bez użycia hasła. Użycie tego rozwiązania, jako modułu w naszym systemie, miałoby analogiczną funkcję odblokowywania portfela. Pomimo ogromnej wygody używania takiego rozwiązania, musimy zwrócić uwagę na jego bezpieczeństwo, które pozostawia wiele do życzenia.

Ze względu na technologiczne ograniczenia, nie jesteśmy w stanie zmierzyć czasu, który potrzebny jest na pokonanie dystansu pomiędzy urządzeniami. Wiąże się to głównie z prędkością rozchodzenia się fal radiowych, która jest równa prędkości światła. Zmierzenie odległości rzędu kilku metrów staje się przez to bardzo problematyczne, gdyż sygnał pokonuje je w bardzo krótkim czasie, który jest niemożliwy do zmierzenia przez zwykły telefon czy Raspberry Pi. Pokonanie dwóch metrów dla światła zajmuje 6,67 \* 10-9 sekundy, zatem wynika z tego, że dla zmierzenia takiej odległości potrzebny jest zegar o częstotliwości około 10 GHz. Dla odległości o jeden rząd niższych, wymagana częstotliwość zegara wzrośnie również o jeden rząd. Oczywiście obliczenia te zakładają idealne warunki, w których pasmo wolne jest od zakłóceń. Problem ten zmusza nas, do pozostania przy metodzie pomiaru siły sygnału. Jej zasadniczą wadą jest podatność na manipulacje, przez używanie wzmacniaczy i zakłócaczy sygnału. Atakujący mógłby bez naszej wiedzy fizycznie wzmocnić sygnał urządzenia i przekazać go do portfela. Wtedy pomimo dużej odległości, zabezpieczenie to nie zadziałałoby. Z kolei podchodząc do tego z drugiej strony, atakujący posiadający zakłócacz fal radiowych, mógłby spowodować blokadę naszych środków w portfelu.

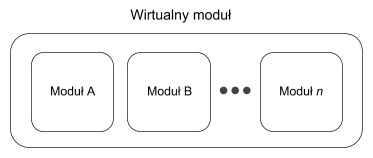
Oprócz samego faktu potrzeby zbadania odległości, w jakiej znajduje się urządzenie będące kluczem, należy znaleźć sposób na bezpieczną identyfikację go. W przypadku usług mało wrażliwych na ataki, wystarczyłby adres urządzenia Bluetooth, który jest unikatowym, 48-bitowym identyfikatorem urządzenia. Jednak jego wartość może odczytać każde urządzenie, które jest w zasięgu sygnału, dlatego nie jest to bezpieczne rozwiązanie. Najbardziej bezpiecznym rozwiązaniem tego problemu jest kryptografia asynchroniczna, która z drugiej strony jest niemożliwa do przeprowadzenia na prostych urządzeniach wspierających Bluetooth, ze względu na ich ograniczenia techniczne. Oprócz samego braku wsparcia kryptografii, trudne lub nawet niemożliwe do rozwiązania może być zapewnienie dostępu do urządzenia, aby dokonać na nim czegokolwiek innego niż to, co przewidział producent. Oczywiście moglibyśmy skonstruować sami takie urządzenie, lub zaadaptować któreś z istniejących rozwiązań, ale wymagałoby to zbyt dużo pracy, a samo rozwiązanie należałoby rozważyć, jako osobny produkt.

Moduł ten w idealnym świecie jest ciekawym i wygodnym rozwiązaniem. Pomysł odblokowywania portfela za pomocą zbliżenia się do niego z innymi swoimi urządzeniami, brzmi wręcz futurystycznie. Jednak uwzględniając wszystkie problemy i wady tego rozwiązania, musieliśmy je odrzucić.

##### 4.6.4. Wirtualne moduły

Oprócz standardowych modułów, w trakcie implementacji powstał pomysł dodania “wirtualnych modułów”. Miałyby one za zadanie tworzyć szczególne przypadki dla kombinacji innych modułów

.



Rys. 4.13. Schemat wirtualnego modułu

Wirtualny moduł powinien odpowiednio rozszerzać funkcjonalności modułów, które w sobie skupia. Przykładem może być wirtualny moduł wykorzystujący dzielenie sekretu za pomocą algorytmu Shamira, opisanego w rozdziale 4.3. W przypadku konfiguracji, która wymaga 1 z 2 sekretów do odblokowania modułu, możemy konfigurować np. zapasowe PINy, przez skojarzenie ze sobą kilku modułów wymagających PINu. Możliwe również mogłoby być zagnieżdżanie takich modułów, co daje nieskończone możliwości. Cały koncept wizualizuje rysunek 4.13.

Wirtualny moduł, który pozwala na tworzenie zapasowych modułów, jest tylko przykładem. Implementacja modułów jest otwarta i pozostawiona ich twórcom. W naszym projekcie nie zdążyliśmy zaimplementować żadnego przykładu wirtualnego modułu, ze względu na dodatkowe przypadki, które nie są aktualnie obsługiwane. Implementację tego pomysłu pozostawiamy na przyszłość.

### 

### 5. Wnioski z procesu implementacji

Ze względu na dużą skalę naszego projektu i jego charakter, czyli styczność zarówno z oprogramowaniem jak i samym sprzętem, napotkaliśmy wiele problemów. Rozdział ten jest poświęcony opisowi ciekawszych i bardziej złożonych przypadków, które często wpływały na to, jak ostatecznie wygląda nasz produkt. Każdy rozwiązany problem znacząco zwiększył naszą ogólną wiedzę i pozwolił wyciągnąć wnioski na przyszłość. Rozdział ten nie zawiera problemów “miękkich”, czyli tych związanych ze współpracą w projekcie pomiędzy jego uczestnikami, gdyż zostały one opisane w rozdziale szóstym. Pomysłodawcą dodania tego rozdziału, jest dr Tomasz Gierszewski.

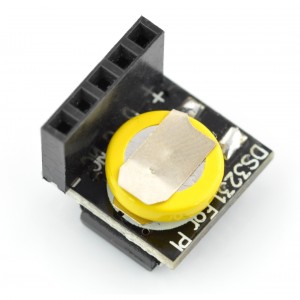
#### 5.1. Brak zegara czasu rzeczywistego w Raspberry Pi

*Piotr Pabis*

Podczas weryfikacji certyfikatu SSL, obie strony potrzebują poprawnie ustawionej daty i godziny w celu potwierdzenia prawdziwości otrzymanego certyfikatu, ponieważ są one podpisywane na określony czas. W przypadku dużej różnicy pomiędzy zegarem systemowym jednej strony a drugiej, użytkownikowi prezentowany jest komunikat o błędzie ważności certyfikatu.

Urządzenia Raspberry Pi do określania czasu korzystają z serwera NTP. Serwer NTP jest potencjalnym punktem ataku DoS. Możliwym jest, że w tej samej sieci domowej, przez którą urządzenie portfela będzie chciało zsynchronizować swój zegar, znajdzie się atakujący podający się za serwer NTP i rozgłaszający złą datę i godzinę [63]. Przez to nawiązanie bezpiecznego połączenia między komputerem użytkownika a portfelem będzie niemożliwe.

Jednym z rozwiązań jest zamontowanie rozszerzenia zegara ze swoją baterią - przykładowy przedstawiony na rysunku 5.1. Jest on w formie zewnętrznego układu o niskiej mocy, podłączanego pod interfejs GPIO, który odlicza czas bez potrzeby udziału procesora Raspberry Pi. Jednak i tutaj pojawia się problem - osoba niepowołana może wyjąć baterię pod nieobecność właściciela.



Rys. 5.1. Rozszerzenie do Raspberry Pi - zegar z baterią. [61]

Ostatecznym i prawdopodobnie najlepszym rozwiązaniem w takiej sytuacji jest możliwość skonfigurowania zegara w portfelu przez komputer użytkownika z użyciem nieszyfrowanego kanału HTTP - np. przy pomocy JavaScript i AJAX, który po poprawnym ustawieniu przekieruje użytkownika z powrotem na HTTPS. Punktem ataku w tym przypadku jest komputer użytkownika, lecz jest to już poza naszą kontrolą. W prototypie projektowym, ze względu na ograniczenia czasowe, pozostaliśmy przy serwerze NTP.

#### 5.2. Niepełna kompatybilność biblioteki BitcoinJ

*Stanisław Barański*

Biblioteka BitcoinJ, na której oparliśmy w znacznej części funkcjonalność naszego projektu, okazała się niekompatybilna z konceptem podziału klucza prywatnego. Biblioteka zakłada, że to ona zarządzać będzie kluczem prywatnym w odpowiedni dla niej sposób. Deweloperzy mogą jedynie określić moment zapisu zaszyfrowanego klucza z pamięci RAM do pamięci urządzenia. Podczas wykonywania operacji, klucz prywatny przechowywany przez bibliotekę, jest wiele razy zapisywany do pamięci trwałej w formie niezaszyfrowanej, co niszczy założenia bezstanowości portfela. Po konsultacjach z zespołem twórców biblioteki [62] ustaliliśmy, że najlepszym rozwiązaniem będzie pozwolenie bibliotece na zarządzanie kluczem w sposób, w jaki została zaprojektowana, a nasze modułowe rozwiązanie będzie jedynie zarządzało kluczem szyfrującym dany klucz prywatny, wykorzystywany przez bibliotekę.

Alternatywnym rozwiązaniem, które mogłoby pozwolić obejść ten problem, jest stworzenie specjalnej partycji, która zapisywałaby swoje dane w pamięci RAM. Takie rozwiązanie nazywane jest jako *ramdisk*, którego konfiguracja w systemie Linux ogranicza się do 4 komend Bash. W przypadku odłączenia od prądu, dane przechowywane na takiej partycji giną bezpowrotnie. Z drugiej strony rozwiązanie to ma też pewien poważny problem. W przypadku zmian w bibliotece, np. miejsca zapisu klucza prywatnego, czy zapisu jakiejkolwiek informacji w jawnej postaci do trwałej pamięci, portfel znowu przestaje być rozwiązaniem bezstanowym. Wymagałoby to śledzenia rozwoju biblioteki i ścisłej analizy jej działania, co mija się z celem i narzuca dużo dodatkowej pracy, dlatego zrezygnowaliśmy z tego rozwiązania.

#### 5.3. Nieaktualna dokumentacja biblioteki Pi4J

*Piotr Pabis*

Biblioteka Pi4J (Pi for Java) służy jako interfejs, który ułatwia komunikację z kontrolerem GPIO dostępnym na platformach Raspberry Pi i jej podobnych.

Dostępu do GPIO jest niezbędny, ze względu na opisany wcześniej moduł fizycznego przełącznika 4.5.1. Przytoczona wcześniej biblioteka oferuje możliwość bardzo łatwej realizacji tej funkcjonalności, poprzez dodanie listenera, który będzie nasłuchiwał wydarzeń na danym pinie GPIO i w razie jakiejkolwiek zmiany stanu, będzie wykonywał odpowiednią część kodu. W dokumentacji Pi4J znajduje się identyczny przypadek, jaki powinniśmy obsłużyć w naszym projekcie. Nie było to wielkim zaskoczeniem, gdyż biblioteka ta jest wykorzystywana głównie do nasłuchiwania zmianu stanów GPIO.

Problemy zaczęły pojawiać się podczas testów integracyjnych oprogramowania z urządzeniem Raspberry Pi. Nie byliśmy w stanie zlokalizować konkretnego GPIO, na którym nasłuchuje program. Po godzinie prób, zaczęliśmy powoli myśleć nad rozwiązaniem alternatywnym do wykorzystania tej biblioteki, którym miała być inna biblioteka, ale w języku Python. Działała ona w pełni poprawnie, a kod był jeszcze prostszy i krótszy. Pozornym problemem był jedynie język, który nie był kompilowany to bytecode JVM, jednak ze względu na zaprojektowaną architekturę, nie było to problemem. W ostatnim momencie przed napisaniem modułu na nowo, odkryliśmy, że pomimo identycznego wyglądu fizycznego w porównaniu do innych wersji GPIO, w naszym Raspberry Pi mapowania niektórych GPIO wyjątkowo różnią się w kilku przypadkach od pozostałych urządzeń tego producenta. Mimo tego odkrycia, nadal nie byliśmy w stanie zaobserwować poprawnego działania modułu.

Przełomem okazało się odkrycie oficjalnego repozytorium z przykładami programów, w których wykorzystano bibliotekę Pi4J. Jeden z przykładów miał za zadanie nasłuchiwać na wszystkich GPIO i wypisywać zmiany na ekran. Okazało się, że działa on poprawnie i do tego tak dobrze, że postanowiliśmy go zaadaptować do naszego projektu, jako osobny moduł do testowania GPIO. Jednak najciekawszym wnioskiem jest to, iż implementacja listenera, która w założeniach jest analogiczna jak dla przypadku z listenerem nasłuchującym na jednym GPIO, jest zupełnie inna.

Pomimo tego, że autorzy powinni założyć trafienie na dokumentację na stronie jako pierwsze w kolejności, jest ona niepoprawna i prawdopodobnie nieaktualna. Poprawną dokumentacją z przykładami jest repozytorium, do którego dużo trudniej trafić, jednak łatwiej jest deweloperom je aktualizować i tutaj doszukujemy się przyczyn tej rozbieżności. Są to oczywiście nasze przypuszczenia i nie wiemy jaki jest stan rzeczywisty, ale chcielibyśmy również zauważyć, iż biblioteka Pi4J jest najpopularniejszą biblioteką aktualnie dostępną do tego celu.

#### 5.4. Określenie finalnego interfejsu komunikacji z modułami

*Patryk Milewski*

Problem określenia interfejsu, czyli z góry ustalonego protokołu wymiany informacji pomiędzy podmiotami, jest jednym z najbardziej popularnych problemów codziennej informatyki. Ścierają się tutaj dwa fronty. Pierwszym jest potrzeba pokrycia każdego przypadku podczas komunikacji. Jeżeli nie zostanie to spełnione, będziemy prawdopodobnie zmuszeni do zmian w naszym interfejsie. Drugim frontem jest zachowanie jak najlepszej kompatybilności wstecznej. W skrócie chodzi o to, aby klienci, którzy korzystają już z naszego interfejsu, nie musieli dostosowywać się do zmian, gdyż może to być bardzo problematyczne i ciągnąć za sobą lawinę zmian. Łatwo wyobrazić sobie sytuację, gdy zmiana interfejsu wymusza zmiany w kliencie, który z kolei udostępnia swój interfejs innym klientom i w wyniku tych zmian, musi on być również zmieniony. W skrajnych przypadkach może dojść nawet do zapętleń. Jak widzimy, warunkiem koniecznym do zachowania kompatybilności, jest unikanie zmian w interfejsie, co jest z kolei bardzo trudne i czasem wręcz niemożliwe, ze względu na nowe przypadki użycia interfejsu, lub zmianę okoliczności wykorzystania poprzednich przypadków. Problem ten pomimo prostoty, nie jest trywialny. Znany jest również pod nazwą ”syndromu dnia następnego”.

Częściowo doświadczyliśmy tego w naszym projekcie Raspberry Wallet i prawdopodobnie doświadczymy go jeszcze wielokrotnie podczas dalszego rozwoju. Początkowo zaprojektowany interfejs wymiany informacji z modułami, zmienił się praktycznie w 100%, co w przypadku wczesnych faz projektu, jest raczej powszechnym zjawiskiem. Pomimo tego, iż posiadamy tylko kilka modułów, wymusiło to na nas dodatkową pracę. W przypadku każdego modułu, jego autor musiał zrozumieć zmiany w interfejsie i odpowiednio dostosować implementację modułu. Jeżeli dołożymy do tego nakłady związane z synchronizacją pracy, które w przypadku potrzeby współpracy wszystkich autorów modułów są duże, okazuje się, że każda zmiana w tym zakresie, powoduje bardzo duże straty czasu. Gdy wyobrazimy sobie sytuację, w której modułów i ich autorów jest znacznie więcej, część autorów nie utrzymuje już modułów, a niektóre moduły w wyniku zmian działają, ale pojawiają się w nich nowe błędy, widzimy prawdziwą skalę problemu.

Można powiedzieć, że w naszym projekcie problem ten nie dał do końca o sobie znać, gdyż ilość modułów jest bardzo ograniczona, a ich działanie bardzo dobrze znane. Z czasem będzie się to zmieniać na naszą niekorzyść i trudno temu przeciwdziałać. Warto mimo wszystko być świadomym tych zależności i poświęcać więcej czasu na dogłębną analizę zmian, które właściwie wykonane mogą się po prostu opłacić. Warto przytoczyć tutaj żartobliwe powiedzenie nieznanego autora: “Tydzień pisania, może zaoszczędzić 15 minut projektowania”. W przypadku projektowania interfejsów, oszczędności te mogą być jeszcze większe.

#### 5.5. Osiągnięcie wysokiej entropii generatora liczb pseudolosowych w systemie Linux

*Piotr Pabis*

Biblioteka BitcoinJ oraz wiele naszych klas związanych z szyfrowaniem, korzysta z “bezpiecznego” generatora liczb pseudolosowych *java.security.SecureRandom* [50]. Klasa ta korzysta z wbudowanego w system GNU/Linux pliku specjalnego */dev/random*. Oryginalna implementacja [64] zwraca wartości dopiero, gdy wysoka entropia zostanie osiągnięta (pod uwagę brane są zmiany w systemie, np. przerwania lub standardowe wejście do tegoż pliku), przez co program wywołujący często zawiesza się, oczekując na dane. W szczególności jest to tym prawdziwsze, iż nasz portfel nie wykonuje wielu rzeczy w tle - zaprojektowano całe oprogramowanie i konfigurację, aby obciążenie procesora było jak najmniejsze.

Rozwiązaniem tego problemu jest instalacja pakietu *rng-tools* [65]*.* dostępnego w domyślnym menadżerze paczek *apt* w systemie DietPi. Zawiera on działający w tle program *rngd*, który periodycznie wysyła dane na standardowe wejście pliku */dev/random*, oraz potrafi korzystać ze sprzętowego generatora liczb pseudolosowych w procesorach BCM2708. W ten sposób próg entropii zostaje osiągnięty dużo częściej.

#### 5.6. Testy jednostkowe wymagające dostępu do API innej aplikacji

*Patryk Milewski*

Podczas implementacji serwera autoryzacji opisanego w rozdziale 4.5.2, natrafiliśmy na typowy problem integracji systemu. Portfel podczas testów jednostkowych modułów, potrzebował działającego serwera autoryzacji, który jest zupełnie innym modułem i wymaga osobnego uruchomienia. Testy jednostkowe z założenia nie powinny testować funkcjonalności, od których zależy testowany fragment kodu. Dlatego też nie możemy uruchamiać całego środowiska zewnętrznego modułu podczas testów.

Klasycznym rozwiązaniem jest utworzenie na maszynie wirtualnej działającej instancji serwera i łączenie się podczas testów do niej. Jednak nie chcieliśmy stosować rozwiązania lokalnego i związanego z jedną maszyną, które trudno przenieść na inne komputery. Do tego narzut na wirtualizację był dla nas nie do zaakceptowania. Postanowiliśmy skorzystać z konteneryzacji Docker. Pozwalało to zbudować obraz kontenera Docker, który następnie trafia do naszego repozytorium [66], skąd może pobrać go każda osoba na świecie. W takim przypadku możliwe jest włączenie naszego modułu serwera autoryzacji za pomocą jednego polecenia na dowolnym komputerze.

Konteneryzacja na lokalnej maszynie mimo wszystko powoduje dodatkowe nakłady i problemy. Naszym głównym celem jest przeniesienie tej usługi na platformę Heroku [67]. Następuje wtedy konteneryzacja usługi i jej osadzenie w chmurze dostarczanej i obsługiwanej przez usługodawcę. Oczywiście nie jest to w pełni darmowe rozwiązanie. Pierwsze 500 godzin czasu uruchomienia usługi jest darmowe. Dodatkowo każda usługa działająca na platformie w darmowym planie subskrypcji podlega ograniczeniu, które automatycznie wyłącza ją po 30 minutach braku aktywności na niej. Nie jest to problemem, gdyż po wyłączeniu takiej maszyny, jedynie pierwsze żądanie zostanie opóźnione o czas ponownego uruchomienia usługi, które dzieje się automatycznie i zajmuje kilkanaście sekund. Kolejne żądania są obsługiwane normalnie. W przypadku integracji środowisk na potrzeby testów, jest to idealne rozwiązanie. Pozwala na dostęp do usługi z dowolnego punktu w Internecie i jest darmowe. Aktualnie jesteśmy na etapie wdrażania tej technologii w module serwera autoryzacji, ale pierwsze testy wykazały pełną poprawność działania testów jednostkowych z platformą.

Innym podobnym problemem okazały się być testy modułu przycisku. W pierwszej kolejności testy nie mogły być wykonywane na komputerze, gdyż biblioteka Pi4J zawiera w sobie skompilowane biblioteki dla architektury ARM. Nawet gdyby Pi4J było skompilowane dla procesorów x86, wciąż pozostaje problem - większość komputerów nie posiada interfejsu GPIO, a nawet jeśli posiada (jak np. Intel Celeron [68]), nie jest to interfejs łatwo dostępny dla posiadacza przeciętnej płyty głównej.

#### 5.7. Dust Payments

*Stanisław Barański*

Nasze środki w blockchainie są przechowywane przez historię transakcji. Nasz aktualny stan adresu to tak naprawdę różnica wszystkich wpłat na dany adres i wszystkich wypłat z danego adresu. Fakt ten powoduje sporo komplikacji. Wyobraźmy sobie, że dostaliśmy na konto 1000 przelewów po 0.001 BTC. Gdy chcemy zrobić przelew wychodzący o wartości 1 BTC, urządzenie będzie musiało podpisać taką transakcję 1000 razy. Wynika to z zasady działania sieci Bitcoin. Ze względu na ograniczone zasoby obliczeniowe Raspberry Pi, taka ilość podpisów może zająć nawet kilka minut.

Zalecane jest, aby przyjmować na portfel przelewy o większej wartości. Można to uzyskać poprzez zbieranie małych przelewów najpierw na pośrednie konto, które dopiero po zebraniu większej ilości środków, przesyła je na nasz bezpieczny portfel. Możliwe jest również wysłanie co pewien czas wszystkich środków do samego siebie. Dzięki temu następnym razem będziemy musieli podpisać tylko jeden przelew wychodzący. Innym rozwiązaniem tego problemu jest inwestycja w bardziej wydajną platformę, która wydajnościowo mogłaby być na tyle wysokiej klasy, aby zmniejszyć zauważalność tego zjawiska.

#### 5.8. Bezstanowość modułu fizycznego przełącznika

*Piotr Pabis*

Podczas implementacji modułu fizycznego przełącznika, który ma za zadanie wymusić fizyczne posiadanie portfela, wyłonił się problem kolidujący z założeniami architektury. Każdy moduł powinien implementować funkcje szyfrowania i odszyfrowywania zadanego ciągu bajtów. Zazwyczaj odbywa się to z użyciem algorytmu AES, którego hasło szyfrujące przechowywane jest poza portfelem. W przypadku guzika, nie jest to możliwe, gdyż nie jest on urządzeniem zdolnym do przechowywania danych oraz nie posiada w sobie żadnych informacji, które mogłaby posłużyć za hasło.

Rozważaliśmy obejście tego problemu, przez zapis hasła modułu do pliku, jednak w żadnym stopniu nie wpływa to na zwiększenie bezpieczeństwa modułu. W przypadku odczytania pamięci portfela, atakujący może odczytać zapisane hasło, tak samo jak możliwe jest odczytanie niezaszyfrowanego sekretu, w przypadku braku szyfrowania hasłem.

Ostatecznie pozostaliśmy przy prostocie, nie zmieniając modułu, dodatkowo założyliśmy implementację bardziej rozwiniętych alternatyw tego modułu w przyszłości. Przycisk mógłby zawierać w sobie pewne informacje, którą można odczytać dopiero po jego wciśnięciu. Alternatywą mogłoby być po prostu wymaganie podania hasła podczas odblokowywania modułu, które byłoby również dodatkowym zabezpieczeniem.

### 

### 6. Narzędzia i metody współpracy

*Patryk Milewski*

Ze względu na zmieniające się etapy pracy nad projektem, zastosowaliśmy różne metodyki współpracy. Zauważyliśmy, że projekt inżynierski bardzo zmienia się wraz z postępem prac. Dlatego pozostanie przy jednym sposobie pracy przez cały jego okres, byłoby nieoptymalne. Wyzwaniem była organizacja pracy naszego zespołu, składającego się z trzech osób. Z góry założyliśmy pracę w pełni zdalną, aby uniknąć zarzutów związanych ze wspólnymi spotkaniami.

Podział prac na dwa etapy nie był przypadkowy, lecz założony z góry. Wizja tego, jak ma wyglądać nasz portfel, była dosyć jasna i znana od pierwszych etapów projektu. Można założyć, że wszystkie wymagania były możliwe do określenia we wczesnej fazie projektu. Dlatego zdecydowaliśmy się wykorzystać tę właściwość projektu i spisaliśmy wszystkie wymagania oraz całą wizję w 40 stronicowym dokumencie o nazwie kodowej R-001, w ramach przedmiotu realizowanego na szóstym semestrze studiów inżynierskich o nazwie Realizacja Projektu Inżynierskiego. Jest to w pewnym stopniu protoplasta dokumentu inżynierskiego.

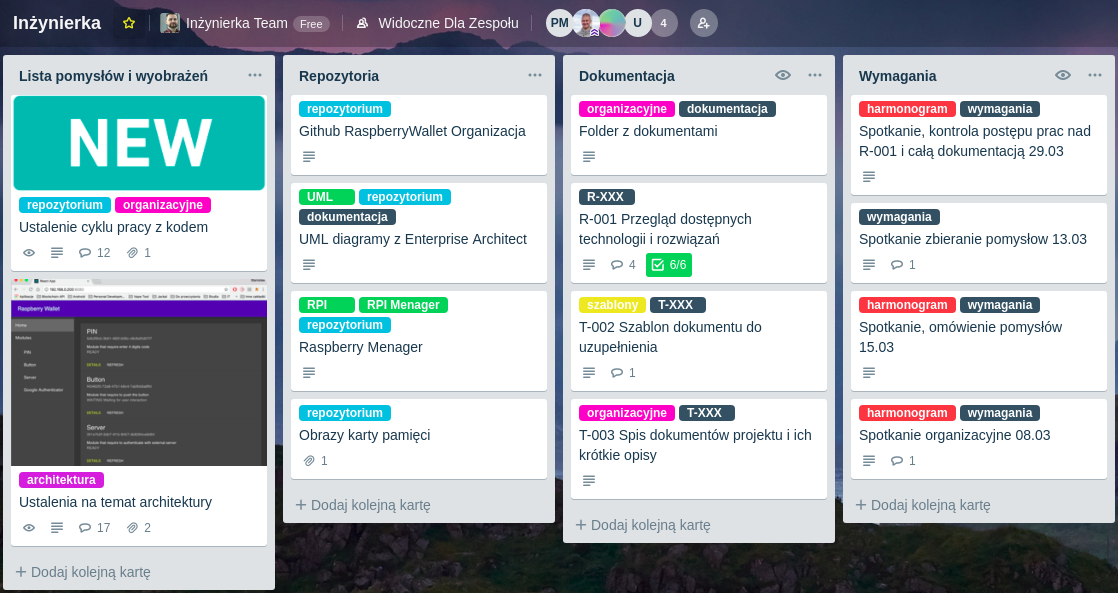
Nauczeni doświadczeniem innych zespołów inżynierskich, pozostawiliśmy okres od początku lipca do połowy września wolnym od prac. Z perspektywy czasu może wydawać się, że moglibyśmy zaimplementować więcej funkcjonalności, jeżeli poświęcilibyśmy go na pracę. Jednak z drugiej strony narazilibyśmy się na ryzyko powstania dużych błędów implementacji we wczesnej fazie projektu, ze względu na brak możliwości skupienia uwagi wszystkich członków na projekcie w tym okresie. W takim przypadku stracilibyśmy bardzo dużo czasu na poprawę kodu, który powstał z błędnymi założeniami i ewentualnie obniżyć entuzjazm zespołu, bardzo istotny w całym przedsięwzięciu.

#### 6.1. Przygotowanie wymagań i dokumentacji wstępnej

*Patryk Milewski*

Pierwszy etap prac polegał na uzgodnieniu wspólnej wizji. Projekt nie posiada bezpośredniego klienta, dlatego sami musieliśmy określić, co powinno zostać zaimplementowane. Ze względu na duże wymagania od strony bezpieczeństwa, potrzebowaliśmy osoby, z którą moglibyśmy konsultować rozwiązania. Taką osobą stał się nasz opiekun projektu, dr Tomasz Gierszewski. Podczas zbierania pomysłów i uzgadniania rozwiązań, ustalone zostały cotygodniowe spotkania w pełnym składzie zespołu. Pozwalało to w znacznym stopniu kontrolować postępy pracy i synchronizować wszystkie pomysły, które pojawiły się w ciągu tygodnia.

Głównym kanałem komunikacji stał się komunikator internetowy Messenger. Pomimo jego wielu wad, takich jak gubienie istotnych informacji w szumie całej konwersacji, brak funkcji ułatwiających organizację projektu oraz wiele innych, zdecydowaliśmy się na jego użycie, ze względu na wygodę. Każdy z nas używa na co dzień tego komunikatora do komunikacji ze znajomymi, przez co nie istnieje potrzeba używania dodatkowego narzędzia.



Rys. 6.1. Fragment tablicy Trello

Po dwóch tygodniach prac zauważyliśmy, że pewne informacje zaczynają się gubić i przestajemy o nich pamiętać. Dlatego zdecydowaliśmy się na użycie narzędzia Trello. Pozwala ono na zarządzanie informacjami w projekcie przez system kart Kanban, gdzie każda z nich odpowiada danemu wątkowi, posiada odpowiednie etykiety i opis, a cała dyskusja jest prowadzona pod nią. Dzięki temu każdy temat zachowuje ciągłość informacji. Trello było wykorzystywane przez cały pierwszy etap projektu. W drugim etapie ograniczyliśmy się jedynie do odszukiwania zawartych w nim informacji i nie dodawania nowych.

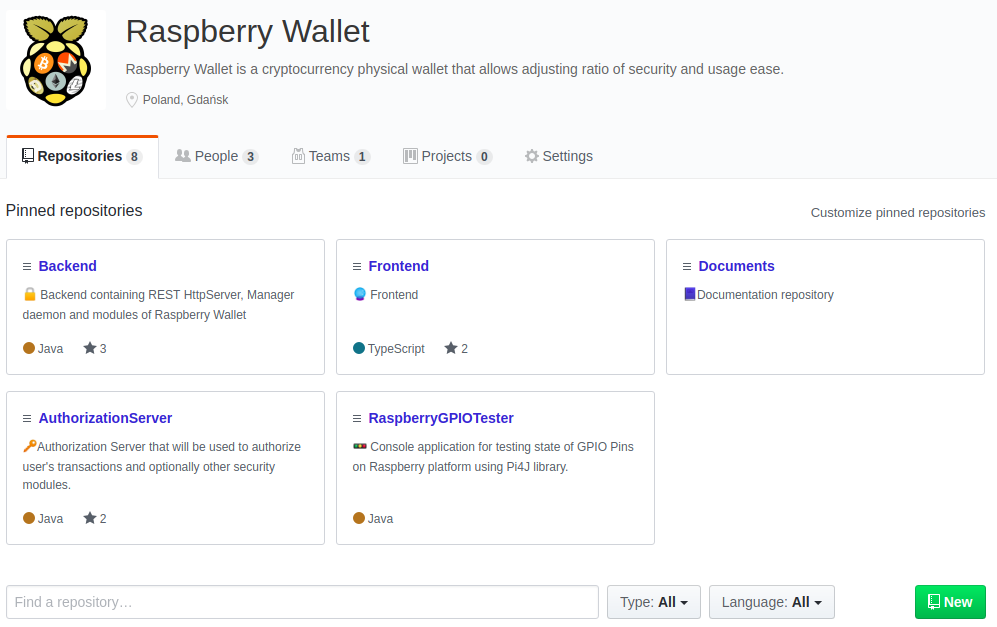
Dokumentację projektową sporządzaliśmy w postaci dokumentów w darmowym narzędziu Google Docs. Pozwala ono na jednoczesną pracę wielu osób nad dokumentem. Dodatkowo pozwala zarządzać pracą, przez system komentarzy, który zawiera funkcję przypisywania zadań osobom. Na początku projektu powstał szablon dokumentacji wewnętrznej w formacie DOC, który był wykorzystywany podczas tworzenia wszystkich dokumentów. Określał on wszystkie szczegóły formatowania tekstu, logotypy oraz informacje o projekcie. Każdy dokument posiada swój kod, który pozwala na szybszą pracę i jednoznaczną identyfikację.

Trudno jest określić metodykę, którą zastosowaliśmy podczas tego etapu. Można powiedzieć, że był to jeden, ciągły sprint, którego celem było utworzenie dokumentu ze spisem wszystkich wymagań i wizji (R-001). Na początku projektu, baliśmy się o organizację, dlatego potrzebowaliśmy formalności i określenia zasad, które miały nam zagwarantować dobrą organizację pracy. Rzeczywistość zweryfikowała te założenia negatywnie. Jeżeli formalności było zbyt wiele, a ich słuszność wydawała się wątpliwa, przestawały być one stopniowo spełniane przez członków zespołu. Pomimo braku z góry określonej organizacji i metodyki, etap ten zakończył się pełnym sukcesem w założonym czasie. Wnioskiem, który wyciągnęliśmy z tego etapu, jest unikanie zbędnej formalizacji i usilnego określenia metod pracy w projektach, których zespoły są bardzo małe i dobrze zorganizowane. Warto jednak pamiętać, że duży wpływ na sukces miała motywacja zespołu i dobra atmosfera pracy. W takich warunkach, zespół potrafi się samoorganizować i dopasowywać do aktualnej sytuacji w projekcie.

#### 6.2. Implementacja kodu

*Patryk Milewski*

Po zakończeniu pierwszego etapu z pełnym sukcesem i odpoczynku podczas przerwy wakacyjnej, w połowie września 2018 roku rozpoczęły się prace nad implementacją. Pierwsze spotkanie po wakacjach było kluczowe dla sukcesu całego projektu. Nie baliśmy się zmian i postanowiliśmy zupełnie zmienić podejście. Narzędzia, które świetnie spisały się wcześniej, zostały zupełnie odstawione i zastąpione. Decyzja ta została podjęta ze względu na świadomość problemu, który pojawia się podczas używania wielu różnych narzędzi w projekcie. Chcieliśmy ograniczyć się do bezwzględnego minimum, którym stała się platforma GitHub.



Rys. 6.2. Główna strona organizacji Raspberry Wallet na platformie GitHub

Główną funkcjonalnością GitHuba jest kontrola wersji za pomocą programu Git. Platforma ta jednak wyróżnia się dodatkowymi funkcjami, takimi jak:

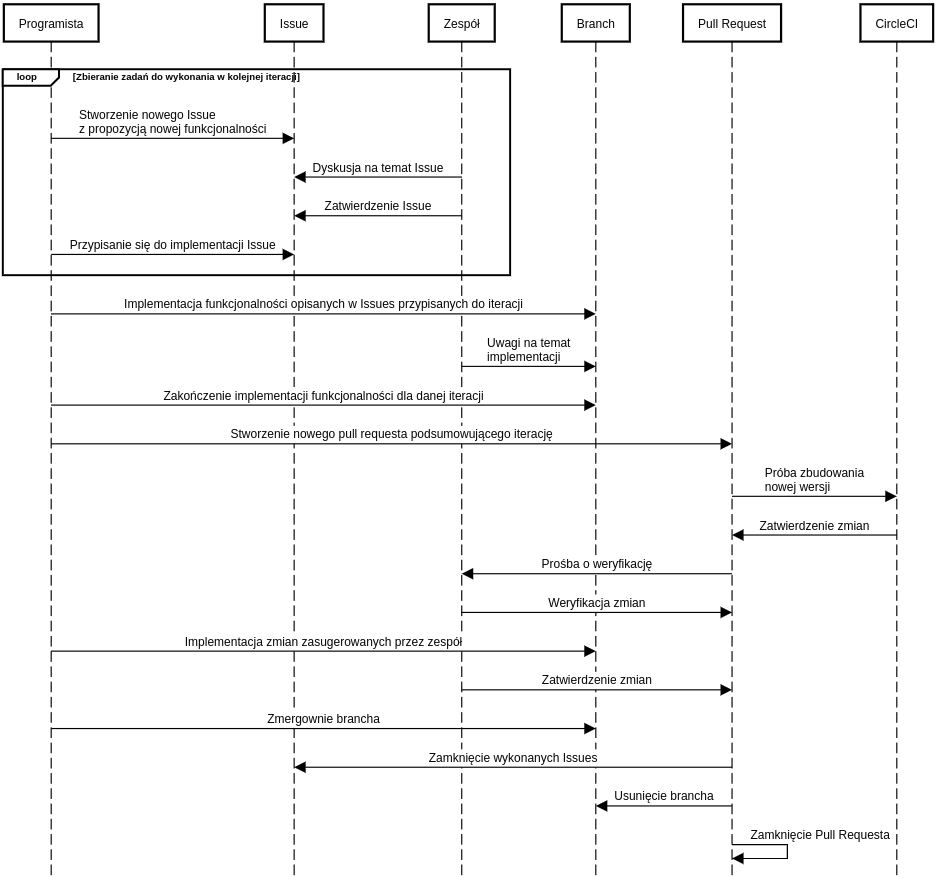
* możliwość zakładania organizacji reprezentujących dany produkt czy firmę,
* zarządzanie zespołami osób w organizacji,
* zarządzanie projektami przez metodykę Kanban,
* śledzenie błędów zgłaszanych przez programistów i użytkowników w systemie Issues (problemy),
* możliwość przeprowadzania Pull Requestów, które polegają na dodaniu zbioru nowych funkcjonalności, które muszą przejść najpierw weryfikację i mogą zostać poddane dyskusji,
* wsparcie dokumentacji przez wewnętrzną Wiki znajdującą się w każdym repozytorium,
* możliwość prowadzenia rozbudowanej dokumentacji za pomocą języka Markdown w plikach README każdego repozytorium,
* integracja z rozszerzeniami zaimplementowanymi przez społeczność.

Funkcjonalność platformy pozwala pokryć wszystkie wymagania stawiane narzędziom do zarządzania projektem. Dlatego wszystkie prace prowadzone w drugim etapie, zostały przeprowadzone na GitHubie. Organizacja RaspberryWallet [69] powstała w celu skupienia wszystkich repozytoriów w jednym miejscu. Na rysunku 6.2 znajduje się widok na stronę organizacji na platformie GitHub. Widoczne są tam również wyróżnione repozytoria oraz ich opisy. Każde repozytorium reprezentuje inną część projektu. Szczegóły repozytoriów zostały przedstawione w tabeli 6.1.

Tabela 6.1. Lista repozytoriów w ramach organizacji RaspberryWallet i ich opis

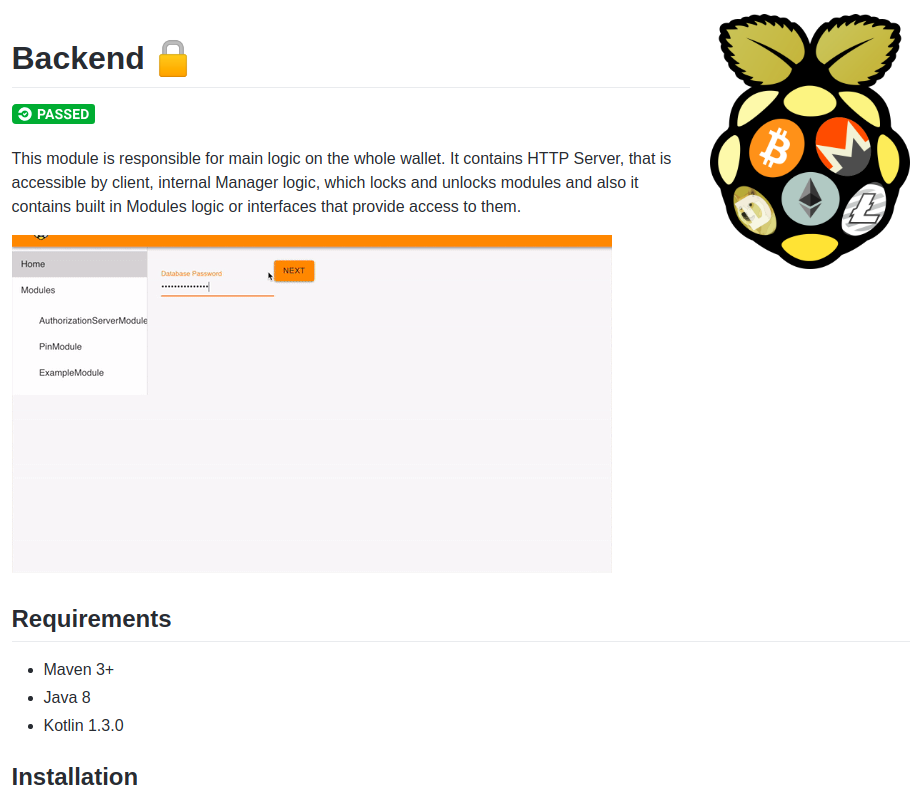
|  |  |
| --- | --- |
| **Nazwa repozytorium** | **Opis zawartości oraz realizowanej funkcjonalności** |
| Backend | Główna logika systemu w językach Java i Kotlin, zarządzanie modułami, kryptografia, implementacja przykładowych modułów, logika odpowiedzialna za wczytywanie konfiguracji całego portfela, serwer z REST API udostępniające dane dla web aplikacji, integracja z zewnętrznymi usługami. |
| Frontend | Web aplikacja w języku TypeScript z użyciem frameworku React, wszystkie widoki aplikacji, logika pobierająca dane z Backendu i wyświetlająca je użytkownikowi, przesyłanie danych wprowadzonych przez użytkownika do Backendu. |
| RaspberryWallet | Konfiguracje systemu, skrypty i instrukcje instalacji. |
| AuthorizationServer | Moduł serwera autoryzacji, REST API udostępniające całą funkcjonalność, która jest konsumowana przez Backend. |
| Documents | Dokumentacja ogólna systemu. |
| UMLDiagrams | Zbiór diagramów UML powstałych podczas pracy nad projektem. |
| RaspberryGPIOTester | Aplikacja obrazująca stany GPIO wszystkich urządzeń producenta Raspberry Pi. Autorami kodu są autorzy przykładów do projektu Pi4J. Umieszczona zgodnie z licencją. |

Metodyka przyjęta podczas tego etapu prac nie została wyspecyfikowana. Założyliśmy podejście iteracyjne. Każdą iteracją był ciąg wydawniczy zbioru funkcjonalności. Iteracja zaczynała się od wybrania przez programistę zbioru funkcji, które chciałby w niej zaimplementować. Listą potrzeb jest zbiór Issues dostępny w danym repozytorium. Następnie zakładany jest specjalny branch (gałąź) w systemie kontroli wersji Git, który przechowywać będzie zmiany dotyczące danej iteracji. Podczas implementacji zmian, każdy może skomentować wprowadzone zmiany. Po skończonym procesie implementacji, zawsze następuje weryfikacja w postaci Pull Requestu. Programista musi stworzyć go na platformie, opisać zmiany dokonane w danej iteracji, wybrać Issues, które zostaną automatycznie zamknięte po pomyślnym przejściu procesu weryfikacji, oraz poprosić o sprawdzenie zmian przez przynajmniej jednego z członków zespołu. Dodatkowo przeprowadzone zostają automatyczne procesy weryfikacji Pull Requestu, aby zapewnić ciągłą integrację projektu, za pomocą narzędzia CircleCI. Jest to usługa, która wykonuje zadane polecenia, aby zbudować projekt i sprawdzić, czy proces zakończy się pomyślnie po wprowadzeniu nowych zmian. Po dokonaniu wszystkich sprawdzeń, zmiany dokonane na branchu iteracji zostają dodane do głównego brancha repozytorium. Przebieg tego procesu został zobrazowany na rysunku 6.3.



Rys. 6.3. Schemat pojedynczej iteracji

Zdecydowaliśmy się na prowadzenie dokumentacji w formie README w języku Markdown. Pliki o nazwie *README.md* są automatycznie rozpoznawane i wyświetlane, podczas przeglądania repozytorium. Pozwala to na integrację dokumentacji oraz kodu w jednym repozytorium. Rysunek 6.4 przedstawia wygląd aktualnego README dla repozytorium Backend.



Rys. 6.4. Widok początku wygenerowanej dokumentacji na podstawie README.md

### 7. PODSUMOWANIE

*Patryk Milewski*

Celem naszej pracy było stworzenie działającego prototypu fizycznego portfela kryptowaluty Bitcoin na platformie Raspberry Pi. Uważamy, że cel ten został w pełni osiągnięty. Prototyp nie jest idealny i wymaga jeszcze dużej ilości pracy, ale jesteśmy bardzo zadowoleni z postępów w pracy, uwzględniając krótki okres czasu projektu - 3 miesiące projektowania i 3 miesiące implementacji. Sumaryczna liczba zaimplementowanych funkcji jest wyższa, niż zakładaliśmy podczas etapu projektowania i analizy. Co prawda nie udało się nam zaimplementować dokładnie wszystkich zakładanych przy podejmowaniu pracy funkcjonalności, jednak wynika to głównie z elastyczności projektu i potrzeby zmiany niektórych założeń. Nie uważamy tego za porażkę, gdyż zaimplementowaliśmy kilka innych funkcjonalności, które wydawały się nam wcześniej zbędne, jednak z czasem potrzeba ich dodania wzrosła.

#### 7.1. Portfel kryptowalut

*Patryk Milewski*

Po dokonaniu analizy rynku portfeli kryptowalut, stwierdzamy, iż jest on bardzo słabo rozwinięty i zdominowany przez giełdy kryptowalut w połączeniu z kilkoma implementacjami fizycznych portfeli. Brak rozwiązań dla pewnych grup użytkowników jest zauważalny gołym okiem. Jesteśmy przekonani, iż istnieje mnóstwo wolnego miejsca, które należałoby zagospodarować za pomocą naszego portfela. Rozwiązanie to jest bardzo elastyczne i skalowalne, co jest niezwykle ważne, ze względu na szybkość zmian na rynku kryptowalut. Na tym etapie prac staramy się dostosować do jak największej ilości potencjalnych użytkowników, jednak nie wykluczamy, iż wraz z postępem prac i ulokowaniu się naszego produktu na rynku, rozwój portfela będzie musiał podporządkować się wymaganiom docelowej grupie odbiorców.

Uważamy, że kryptowaluty są przyszłością i wspieranie ich użytkowników przez doskonalenie rozwiązań, które ułatwiają korzystanie z nich, jest warte poświęconego czasu. Można powiedzieć, że jesteśmy dopiero na początku ery nowej bankowości, którą zapoczątkował Bitcoin. Ogromne zainteresowanie tym tematem wskazuje na to, iż nie jest to tylko tymczasowy trend. Ciekawym przypadkiem są kraje pogrążone w kryzysie gospodarczym, których mieszkańcy sięgnęli po kryptowaluty, w celu ochrony przed hiperinflacją. Wenezuela jest jednym z takich krajów. Warto również wspomnieć, że rząd tego państwa próbował ratować sytuację gospodarczą przez emisję własnej kryptowaluty o nazwie Petro. Przedsięwzięcie zakończyło się porażką, ale pokazuje jak istotne stają się kryptowaluty.

#### 7.2. Zastosowane technologie

*Patryk Milewski*

Jednym z powodów sukcesu naszego przedsięwzięcia jest prawidłowy dobór technologii. Wybraliśmy rozwiązania, które znamy, lub takie, które są im pokrewne. Dzięki temu nie musieliśmy pokonywać bariery nauki zupełnie nowych rozwiązań. Nie natrafiliśmy na duże rozbieżności w oszacowaniach złożoności problemów i byliśmy w stanie dosyć dobrze określić zawiłość danego rozwiązania. W naszym przekonaniu czasy dominacji języków z rodziny C w urządzeniach o ograniczonych zasobach minęły. Wybór trudnych technologii w tak małych projektach, niekoniecznie zapewnia spodziewany zysk i może przyczynić się do problemów podczas implementacji. Warto wybierać nowoczesne platformy, gdyż podnosi to ogólną atrakcyjność projektu i zachęca nowych programistów do wzięcia udziału w projekcie, co jest bardzo ważne w przypadku projektów open-source.

#### 7.3. Urządzenie Raspberry Pi Zero W

*Piotr Pabis*

Jednym z wniosków z naszej pracy jest ocena platformy, która została użyta do prototypowania. Uważamy, iż wybrane przez nas urządzenie jest bardzo dobrą platformą prototypową. Głównymi czynnikami, które wpłynęły na nasz wybór, jest posiadanie złącza OTG w postaci micro USB, oraz wbudowana karta sieci bezprzewodowej komunikująca się w paśmie 2,4 GHz. Oprócz tych parametrów, uwzględniliśmy inne zalety, takie jak bardzo duża społeczność Raspberry Pi, wysoka popularność samej platformy, rozpoznawalność marki i przede wszystkim niska cena, zniechęcają do poszukiwania innego rozwiązania. W przypadku takich projektów jak nasz, głównym wymaganiem od platformy jest możliwość sprawdzenia, czy system rzeczywiście będzie działał w środowisku o ograniczonych zasobach. Czas poświęcony na poszukiwania optymalnej platformy lepiej poświęcić jest na implementację samego rozwiązania, która dużo więcej wniesie do projektu.

#### 7.4. Zastosowany model współpracy

*Patryk Milewski*

Nasz model współpracy zakładał podział prac na dwa etapy - analizy i projektowania oraz implementacji i integracji. W przypadku projektów, których wymagania jesteśmy w stanie określić z góry, jest to bardzo skuteczne podejście. Podczas pierwszego etapu organizowaliśmy regularne, cotygodniowe spotkania, które owocowały wartościową dyskusją, identyfikacją problemów przed ich wystąpieniem, synchronizacją prac oraz utrzymaniem terminowości. Ten etap trwał około trzy miesiące, a w jego wyniku powstał około 40 stronicowy dokument, określający naszą wizję implementacji portfela. Druga faza rozpoczęła się po okresie wakacyjnym i trwała również około trzech miesięcy. Podczas tego etapu prac nie określiliśmy z góry żadnej metodyki, jedynie założyliśmy luźne, iteracyjne podejście, bazujące na zmiennych iteracjach, które składały się ze zbioru funkcjonalności. Cały proces przeprowadzany był na platformie GitHub, która bardzo wspomogła nasz projekt. Dokumentacja techniczna powstała w postaci JavaDocs oraz plików README w języku Markdown. Dodatkowo założyliśmy wymóg pełnego przegląd kodu i zgłaszania uwag, podczas zakończenia implementacji danej iteracji. Każdą iteracją zajmowała się tylko jedna osoba.

Stwierdzamy, że takie podejście było słuszne i zagwarantowało nam bardzo wysoką jakość kodu i jego znajomości przez wszystkich członków zespołu. Nie pracowaliśmy wcześniej w takim składzie, a tym bardziej w takiej metodyce. Podejście po części zmieniało się podczas trwania projektu, ale główny zarys istniał już od samego początku. W przypadku małych zespołów, które są w stanie same się motywować do pracy i organizować ją, narzucanie sztucznej metodyki pracy jest ograniczeniem, które może hamować postęp.

#### 7.5. Plany na przyszłość

*Patryk Milewski*

Od samego początku projektu nie traktowaliśmy go tylko i wyłącznie jako projektu studenckiego, stworzonego pod przymusem, z założeniem jego zakończenia po otrzymaniu oceny. Na każdym etapie prac zakładaliśmy, iż 14 grudnia 2018 roku, czyli termin zakończenia prac nad projektami inżynierskimi, jest jedynie terminem wydania pierwszej wersji Raspberry Wallet. Podczas analizy i projektowania całego rozwiązania, część rozwiązań omówiona była pod kątem implementacji w przyszłości, w ramach kolejnych wydań. Z kolei podczas implementowania portfela, powstała lista “After first release features”, która skupia funkcjonalności pozostawione na przyszłość, czyli okres po pierwszym wydaniu. Na ten moment lista ta zawiera około 10 pozycji, z czego 3 wymagają dużo większej ilości pracy. Mimo wszytko po zakończeniu wszystkich aktualnych Issues obecnych na platformie GitHub, na pewno powstałyby nowe wątki, które wymagałyby nowych funkcjonalności, gdyż wiele pomysłów nie zostało jeszcze opisanych i są tylko naszymi wyobrażeniami na przyszły rozwój projektu.

W przyszłości chcielibyśmy dodać obsługę dużo większej ilości kryptowalut, co zwiększyłoby potencjalny zasięg produktu. Również interfejs portfela w postaci web aplikacji potrzebuje rozbudowania, gdyż posiada on aktualnie tylko niezbędne funkcje do działania, a mógłby prezentować dużo więcej informacji o portfelu. Bardzo ważne jest przeprowadzenie dokładnej analizy statycznej kodu, oraz wprowadzenia większej ilości automatycznych testów integracyjnych i funkcjonalnych. Poprawiłoby to jeszcze bardziej ogólną jakość kodu i zagwarantowało większą stabilność całego systemu. Jesteśmy świadomi, iż prace te pochłoną przynajmniej tak samo dużo czasu, jak te wykonane dotychczas. Naszym celem nie jest tylko i wyłącznie odniesienie sukcesu na rynku. Jednym z zysków jest możliwość pochwalenia się bardzo ciekawym projektem o otwartym kodzie, w którym można sprawdzić, na jakim poziomie są nasze umiejętności programistyczne i pracy w zespole. Jest to pewnego rodzaju wizytówka, którą niewiele osób w naszym wieku może się pochwalić.

### Wykaz literatury

|  |  |
| --- | --- |
| [1] | S. Nakamoto, „Bitcoin: A Peer-to-Peer Electronic Cash System” 2009. https://bitcoin.org/bitcoin.pdf. [Data uzyskania dostępu: 8-12-2018]. |
| [2] | Ledger Team, „Ledger Documentation Hub”. https://ledger.readthedocs.io/en/latest/. [Data uzyskania dostępu: 30 3 2018]. |
| [3] | A. M. Antonopoulos, "Mastering Bitcoin" 2014. |
| [4] | N. Bacca, „Secure Hardware and Open Source”, https://www.ledger.fr/2016/06/09/secure-hardware-and-open-source/ [Data uzyskania dostępu: 30-3-2018]. |
| [5] | I. Bashir, "Mastering Blockchain", 2017. |
| [6] | P. Wuille, „Bitcoin Improvement Proposals 32”. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki [Data uzyskania dostępu: 8-12-2018]. |
| [7] | Wielu autorów, „Bitcoin Improvement Proposals 39”. https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki [Data uzyskania dostępu: 8-12-2018]. |
| [8] | B. Schneier, "Kryptografia dla praktyków. Protokoły, algorytmy i programy źródłowe w języku C" 2002. |
| [9] | Strona internetowa projektu BitcoinJ. https://bitcoinj.github.io/ [Data uzyskania dostępu: 8-12-2018]. |
| [10] | Dokumentacja protokołu Bitcoin. https://bitcoin.org/en/developer-guide [Data uzyskania dostępu: 8-12-2018]. |
| [11] | Kod źródłowy systemu Bitcoin na platformie GitHub. https://github.com/bitcoin/bitcoin/blob/master/src/amount.h [Data uzyskania dostępu: 8-12-2018]. |
| [12] | Bitcoin.it, „Confirmation”. https://en.bitcoin.it/wiki/Confirmation [Data uzyskania dostępu: 8-12-2018]. |
| [13] | Bitcoin.it, „Genesis block”. https://en.bitcoin.it/wiki/Genesis\_block [Data uzyskania dostępu: 8-12-2018]. |
| [14] | Krystian Matusiewicz, Josef Pieprzyk, Norbert Pramstaller, Christian Rechberger i Vincent Rijmen, „Analysis of simplified variants of sha-256. In WEWoRC 2005 - Western European Workshop on Research in Cryptology” 2005, strony 123-134. |
| [15] | David Harrison, „Specyfikacja protokołu BitTorrent” 2008. http://www.bittorrent.org/beps/bep\_0000.html [Data uzyskania dostępu: 8-12-2018]. |
| [16] | Wikipedia, „Pay-to-Pubkey Hash”. https://en.bitcoinwiki.org/wiki/Pay-to-Pubkey\_Hash [Data uzyskania dostępu: 8-12-2018]. |
| [17] | Wikipedia, „Controlled supply”. https://en.bitcoin.it/wiki/Controlled\_supply [Data uzyskania dostępu: 8-12-2018]. |
| [18] | „Denominations”. https://bitcoin.org/en/glossary/denominations [Data uzyskania dostępu: 8-12-2018]. |
| [19] | Raspberrypi-spy.co.uk, „Simple Guide to the Raspberry Pi GPIO Header and Pins” 2012. https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/ [Data uzyskania dostępu: 8-12-2018]. |
| [20] | Wikipedia, „Secret sharing”. https://en.wikipedia.org/wiki/Secret\_sharing [Data uzyskania dostępu: 8-12-2018]. |
| [21] | A. Shamir, "How to share a secret", 1979. |
| [22] | Strona internetowa projektu Pi4J. http://pi4j.com/ [Data uzyskania dostępu: 8-12-2018]. |
| [23] | Arch Linux Wiki, „Rng-tools”. https://wiki.archlinux.org/index.php/Rng-tools  [Data uzyskania dostępu: 8-12-2018]. |
| [24] | Strona usługodawcy bramki SMS w postaci interfejsu programistycznego API. https://www.smsapi.com/en/pricing [Data uzyskania dostępu: 8-12-2018]. |
| [25] | BitcoinJ Team, „Security-model”. https://bitcoinj.github.io/security-model  [Data uzyskania dostępu: 8-12-2018]. |
| [26] | C. Aktas, "The Evolution and Emergence of QR Codes, Newcastle upon Tyne", UK : Cambridge Scholars Publishing, 2017. |
| [27] | Bitcoin.it „B-money”. https://en.bitcoin.it/wiki/B-money [Data uzyskania dostępu: 8-12-2018]. |
| [28] | Bitcoin.it „Hashcash”. https://en.bitcoin.it/wiki/Hashcash [Data uzyskania dostępu: 8-12-2018]. |
| [29] | S. Fotheringham „Wireless Broadband Conflict and Convergence” 2008, strona 57. |
| [30] | Strona projektu mDNS Avahi. https://www.avahi.org/ [Data uzyskania dostępu: 8-12-2018]. |
| [31] | Kamera kompatybilna z Raspberry Pi Zero W. https://botland.com.pl/pl/kamery-dla-raspberry-pi/12746-kamera-zerocam-ov5647-5mpx-dla-raspberry-pi-zero.html [Data uzyskania dostępu: 8-12-2018]. |
| [32] | Heroku Ltd „How Heroku works”. https://devcenter.heroku.com/articles/how-heroku-works  [Data uzyskania dostępu: 8-12-2018]. |
| [33] | Specyfikacja języka YAML na oficjalnej stronie internetowej projektu. https://yaml.org/spec/  [Data uzyskania dostępu: 8-12-2018]. |
| [34] | Manager w repozytorium Backend. https://github.com/RaspberryWallet/Backend/tree/master/Manager/src/main/java/io/raspberrywallet/manager/modules [Data uzyskania dostępu: 8-12-2018]. |
| [35] | Repozytorium Backend. https://github.com/RaspberryWallet/Backend  [Data uzyskania dostępu: 8-12-2018]. |
| [36] | G. Halfacree „Raspberry Pi Zero W", licencja Creative Commons 2.0. https://www.flickr.com/photos/120586634@N05/33052396492. [Data uzyskania dostępu: 8-12-2018]. |
| [37] | S. R. Leonard Richardson "RESTful Web Services", 2008. |
| [38] | Redis „Introduction to Redis”. https://redis.io/topics/introduction [Data uzyskania dostępu: 8-12-2018]. |
| [39] | Wikipedia „Universally unique identifier”. https://en.wikipedia.org/wiki/Universally\_unique\_identifier.  [Data uzyskania dostępu: 8-12-2018]. |
| [40] | Oracle „Class SecureRandom”. https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html  [Data uzyskania dostępu: 8-12-2018]. |
| [41] | Aanchal Malhotra; Isaac E. Cohen; Erik Brakke i Sharon Goldberg "Attacking the Network Time Protocol", Boston University, 2015. |
| [42] | Arm Ltd „Arm TrustZone”. https://developer.arm.com/technologies/trustzone [Data uzyskania dostępu: 8-12-2018]. |
| [43] | Samsung Electronics co. Ltd „Knox Platform for Enterprise”. https://docs.samsungknox.com/whitepapers/knox-platform/samsung-knox.html  [Data uzyskania dostępu: 8-12-2018]. |
| [44] | Apple Inc. „iOS Security” 2018. https://www.apple.com/business/site/docs/iOS\_Security\_Guide.pdf [Data uzyskania dostępu: 8-12-2018]. |
| [45] | S. Josefsson „The Base16, Base32, and Base64 Data Encodings” 2006. https://tools.ietf.org/html/rfc4648 [Data uzyskania dostępu: 8-12-2018]. |
| [46] | Dokumentacja projektu Spring Boot. https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ [Data uzyskania dostępu: 8-12-2018]. |
| [47] | Strona projekut Jedis na platformie GitHub. https://github.com/xetorthio/jedis  [Data uzyskania dostępu: 8-12-2018]. |
| [48] | Apache Software Foundation „Apache Maven” 2018. https://maven.apache.org/  [Data uzyskania dostępu: 8-12-2018]. |
| [49] | Red Hat Inc. „fabric8 documentation”. http://fabric8.io/docs/index.html  [Data uzyskania dostępu: 8-12-2018]. |
| [50] | GitHub Inc. „GitHub Flavored Markdown Spec” 2017. https://github.github.com/gfm/.  [Data uzyskania dostępu: 8-12-2018]. |
| [51] | Organizacja Raspberry Wallet na platformie GitHub. https://github.com/RaspberryWallet  [Data uzyskania dostępu: 8-12-2018]. |
| [52] | R. Gertrude Chavez-Dreyfuss „About $1.2 billion in cryptocurrency stolen since 2017: cybercrime group” 2018. https://www.reuters.com/article/us-crypto-currency-crime/about-1-2-billion-in-cryptocurrency-stolen-since-2017-cybercrime-group-idUSKCN1IP2LU [Data uzyskania dostępu: 8-12-2018]. |
| [53] | C. Kate Rooney „$1.1 billion in cryptocurrency has been stolen this year, and it was apparently easy to do” 2018. https://www.cnbc.com/2018/06/07/1-point-1b-in-cryptocurrency-was-stolen-this-year-and-it-was-easy-to-do.html [Data uzyskania dostępu: 8-12-2018]. |
| [54] | Dokumentacja Docker-compose, rozdział "replicas”. https://docs.docker.com/compose/compose-file/#replicas [Data uzyskania dostępu: 8-12-2018]. |
| [55] | Docker image modułu Authorization-server 2018. https://hub.docker.com/r/piwallet/authorization-server/ [Data uzyskania dostępu: 8-12-2018]. |
| [56] | Bitcoin.it „Multisignature” 2018. https://en.bitcoin.it/wiki/Multisignature.  [Data uzyskania dostępu: 8-12-2018]. |
| [57] | Pavol Rusnak, Marek Palatinus „SLIP-0044 : Registered coin types for BIP-0044” 2014. https://github.com/satoshilabs/slips/blob/master/slip-0044.md [Data uzyskania dostępu: 8-12-2018]. |
| [58] | Apple Inc., dokumentacja technologii iBeacon dla deweloperów. https://developer.apple.com/ibeacon/ [Data uzyskania dostępu: 8-12-2018]. |
| [59] | Forth Inc. „What is the Forth programming language?”. https://www.forth.com/forth  [Data uzyskania dostępu: 8-12-2018]. |
| [60] | Moduł RTC DS3231 I2C - zegar czasu rzeczywistego https://botland.com.pl/pl/moduly-rtc/3790-modul-rtc-ds3231-i2c-zegar-czasu-rzeczywistego-bateria.html [Data uzyskania dostępu: 8-12-2018]. |
| [61] | Dyskusja odnośnie założeń projektowych biblioteki BitcoinJ, 2018. https://groups.google.com/forum/#!topic/bitcoinj/eCRiK0RBdjw. [Data uzyskania dostępu: 8-12-2018]. |
| [62] | Jason. A. Donenfeld, implementacja generatora liczb losowych w systemie operacyjnym Linux. https://github.com/torvalds/linux/blob/master/drivers/char/random.c [Data uzyskania dostępu: 8-12-2018]. |
| [63] | Intel Corporation „Intel® Pentium® and Celeron® Processor N- and J- Series Datasheet volume 1” 2016, strony 45-59. |
| [64] | Strona domowa desktopowego portfela kryptowalut Electrum. https://electrum.org/#home  [Data uzyskania dostępu: 8-12-2018]. |
| [65] | Strona domowa wieloplatformowego portfela kryptowalut Jaxx. https://jaxx.io/  [Data uzyskania dostępu: 8-12-2018]. |
| [66] | Strona domowa wieloplatformowego portfela kryptowalut Exodus. https://www.exodus.io/  [Data uzyskania dostępu: 8-12-2018]. |
| [67] | Strona domowa wieloplatformowego portfela kryptowalut Copay. https://copay.io/  [Data uzyskania dostępu: 8-12-2018]. |
| [68] | Julia Magas, Cointelegraph, „Crypto Exchange Hacks in Review: Proactive Steps and Expert Advice” 2018. https://cointelegraph.com/news/crypto-exchange-hacks-in-review-proactive-steps-and-expert-advice [Data uzyskania dostępu: 8-12-2018]. |
| [69] | Dokumentacja systemu Raspbian. https://www.raspberrypi.org/documentation/raspbian/  [Data uzyskania dostępu: 8-12-2018]. |
| [70] | Strona projektu DietPi. https://dietpi.com/ [Data uzyskania dostępu: 8-12-2018]. |
| [71] | Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley „The Java® Virtual Machine Specification” 2015. https://docs.oracle.com/javase/specs/jvms/se8/html/ [Data uzyskania dostępu: 8-12-2018]. |
| [72] | Strona domowa projektu Kotlin. https://kotlinlang.org/ [Data uzyskania dostępu: 8-12-2018]. |
| [73] | Strona domowa projektu TypeScript. http://www.typescriptlang.org/  [Data uzyskania dostępu: 8-12-2018]. |
| [74] | Specyfikacja protokołu HTTP 1.1, 1999. https://www.w3.org/Protocols/rfc2616/rfc2616.html.  [Data uzyskania dostępu: 8-12-2018]. |
| [75] | Strona domowa serwera HTTP Ktor. https://ktor.io/ [Data uzyskania dostępu: 8-12-2018]. |
| [76] | Strona domowa frameworka React. https://reactjs.org/ [Data uzyskania dostępu: 8-12-2018]. |
| [77] | Dokumentacja technologii WebSockets. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\_API [Data uzyskania dostępu: 8-12-2018]. |
| [78] | R. Droms, „Dynamic Host Configuration Protocol” 1997. https://www.ietf.org/rfc/rfc2131.txt  [Data uzyskania dostępu: 8-12-2018]. |
| [79] | Dokumentacja serwera DHCP organizacji ISC. https://www.isc.org/dhcp-manual-pages/  [Data uzyskania dostępu: 8-12-2018]. |
| [80] | Informacje na temat rozszerzenia AES New Instructions (AES-NI). https://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard--aes-/data-protection-aes-general-technology.html [Data uzyskania dostępu: 8-12-2018]. |
| [81] | Informacje na temat rozszerzenia NEON. https://developer.arm.com/technologies/neon  [Data uzyskania dostępu: 8-12-2018]. |
| [82] | Zmiany w bibliotece OpenSSL z wersji 1.0.0h na 1.0.1 dotyczące koprocesora NEON. https://www.openssl.org/news/cl102.txt [Data uzyskania dostępu: 8-12-2018]. |

### Dodatek A

#### Przykładowa zawartość pliku *sudoers*

|  |
| --- |
| **#** **#** This file MUST be edited with the 'visudo' command as root. **#** **#** Please consider adding local content **in** /etc/sudoers.d/ instead of **#** directly modifying this file. **#** **#** See the man page **for** details on how to write a sudoers file. **#** Defaults env\_reset Defaults mail\_badpass Defaults secure\_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"  **#** Host alias specification  **#** User alias specification  **#** Cmnd alias specification  **#** User privilege specification root ALL=(ALL:ALL) ALL  **#** Allow members of group sudo to execute any command **%**sudo ALL=(ALL:ALL) ALL  **#** See sudoers(5) **for** more information on "#include" directives:  **%**netdev ALL=(root) NOPASSWD: /sbin/iwlist wlan0 scan **%**netdev ALL=(root) NOPASSWD: /sbin/wpa\_supplicant \* **%**netdev ALL=(root) NOPASSWD: /sbin/dhclient \* **%**netdev ALL=(root) NOPASSWD: /usr/bin/killall wpa\_supplicant **%**netdev ALL=(root) NOPASSWD: /sbin/ifdown \* **%**netdev ALL=(root) NOPASSWD: /sbin/ifconfig wlan0 \* |

#### 

### Dodatek B

#### Zawartość abstrakcyjnej klasy Module modułu Backend Manager

|  |
| --- |
| **package** io.raspberrywallet.manager.modules;  */\*\*  \* Module base class.  \*  \* @param <Config> type of module specific configuration class implementing ModuleConfig base interface  \*/* **@ToString** **public** **abstract** **class** **Module**<**Config** **extends** **ModuleConfig**> {  **@NotNull**  **private** String statusString;   **@NotNull**  **private** HashMap<String, String> input = **new** HashMap<>();   */\*\*  \* Configuration object available for every module.  \* This object should hold every customizable module specific property.  \*/*  **@NotNull**  **protected** Config configuration;  */\*\*  \* This constructor enforce that the state and configuration is always present.  \*  \* @param initialStatusString - initial module status string  \* @param configClass - class representation of module specific Config type,  \* required for dynamic Config class initialization, either from file and default value  \*/*  **public** **Module**(@NotNull String initialStatusString, Class<Config> configClass)  **throws** IllegalAccessException, InstantiationException {   statusString = initialStatusString;  configuration = configClass.newInstance();  }   */\*\*  \* This constructor tries to parse configuration from yaml file else initialize with default one.  \* This constructor is required for dynamic instantiation  \*  \* @param initialStatusString - initial module status string  \*/*  **public** **Module**(@NotNull String initialStatusString, Configuration.ModulesConfiguration modulesConfiguration,  Class<Config> configClass) **throws** IllegalAccessException, InstantiationException {   statusString = initialStatusString;  Config newConfiguration = parseConfigurationFrom(modulesConfiguration, configClass);  **if** (newConfiguration == **null**)  newConfiguration = configClass.newInstance();   configuration = newConfiguration;  }  */\*\*  \* Parses config yaml file representation to module specific Config object.  \*  \* @param moduleConfiguration whole `modules` node of yaml file  \* @param configClass class representation of module specific Config type  \* @return module specific config object  \*/*  **private** Config **parseConfigurationFrom**(Configuration.ModulesConfiguration moduleConfiguration, Class<Config> configClass) {    String configId = getId();  JsonNode jsonNode = moduleConfiguration.get(configId);  ObjectMapper mapper = **new** ObjectMapper();  **try** {  **return** configuration = mapper.treeToValue(jsonNode, configClass);  } **catch** (IOException e) {  Logger.err("Failed to parse configuration");  e.printStackTrace();  **return** **null**;  }  }   */\*\*  \* Used in all sort of identifications like config.yaml, internal module mapping and UI naming.  \* For now, it's just simplified SimpleClassName  \* @return module identifier.  \*/*  **@NotNull**  **public** String **getId**() {  **return** **this**.getClass().getSimpleName();  }   */\*\*  \* @return Server-side module representation  \*/*  **@NotNull**  **public** io.raspberrywallet.contract.module.Module **asServerModule**() {  **return** **new** io.raspberrywallet.contract.module.Module(getId(), getId(), getDescription(), getHtmlUi()) {};  }  */\*\*  \* This wrapper enforce module to validateInputs and throw exception if they are absent.  \*  \* @param keyPart - unencrypted key part  \* @return encrypted payload  \*/*  **@NotNull**  **public** **byte**[] encryptKeyPart(**@NotNull** **byte**[] keyPart) **throws** EncryptionException, RequiredInputNotFound, InternalModuleException {  validateInputs();  **return** **encrypt**(keyPart);  }   */\*\*  \* This wrapper enforce module to validateInputs and throw exception if they are absent.  \*  \* @param payload - encrypted payload  \* @return decrypted key part  \*/*  **@NotNull**  **public** **byte**[] decryptKeyPart(**@NotNull** **byte**[] payload) **throws** DecryptionException, RequiredInputNotFound, InternalModuleException {  validateInputs();  **return** **decrypt**(payload);  }   */\*\*  \* method to override by module, validation should not be called here, use validateInputs() instead  \*  \* @param keyPart - unencrypted key part  \* @return encrypted payload  \*/*  **protected** **abstract** **byte**[] encrypt(**byte**[] keyPart) **throws** EncryptionException, InternalModuleException;  */\*\*  \* Method to override by module, validation should not be called here, use validateInputs() instead.  \*  \* @param payload - encrypted payload  \* @return decrypted key part  \*/*  **protected** **abstract** **byte**[] decrypt(**byte**[] payload) **throws** DecryptionException, InternalModuleException;   **protected** **abstract** **void** **validateInputs**() **throws** RequiredInputNotFound;  */\*\*  \* Returns general description what this module does  \*  \* @return message  \*/*  **@NotNull**  **public** **abstract** String **getDescription**();   */\*\*  \* This method should return HTML UI form or null if not required.  \*/*  **@Nullable**  **public** **abstract** String **getHtmlUi**();   */\*\*  \* Returns current status of the module to show to the user.  \*  \* @return message  \*/*  **@NotNull**  **public** String **getStatusString**() {  **return** statusString == **null** ? "null" : statusString;  }  */\*\*  \* Setting current status message of the module.  \*  \* @param status - new status  \*/*  **public** **void** **setStatusString**(@NotNull String status) {  **this**.statusString = status;  }   */\*\*  \* Sets input for this Module from user.  \*  \* @param key - key of the parameter  \* @param value - value of the parameter  \*/*  **public** **void** **setInput**(@NotNull String key, @NotNull String value) {  input.put(key, value);  }   */\*\*  \* Sets inputs for this Module from user.  \*/*  **public** **void** **setInputs**(@NotNull Map<String, String> inputs) {  input.putAll(inputs);  }   */\*\*  \* Checks if there is input with given key.  \*/*  **public** **boolean** **hasInput**(@NotNull String key) {  **return** input.containsKey(key) && !input.get(key).isEmpty();  }  */\*\*  \* Checks if user has submitted any input  \*  \* @param key - key of the parameter  \* @return - if key exists  \*/*  **@Nullable**  **public** String **getInput**(String key) {  **return** input.**get**(key);  }   */\*\*  \* Clear the user inputs, prepare for new  \*/*  **public** **void** **clearInputs**() {  input.clear();  } } |