A Secure IOTA-Based Mobile Payment System for Smart Vehicle (Final Report)

Advanced Security Engineering Lab

Group: SALEHI, Mohammad Masoud - ZALI, Mojtaba - BHARDWAJ, Mayank

Supervised by: Nikolaos Athanasios Anagnostopoulos

Faculty of Computer Science and Mathematics

University of Passau

Contents

List of Figures

1 Introduction

The goal of this project is to evaluate the viability of implementing an IOTA-based wallet for electric vehicles, focusing mainly on increasing the security of the wallet by exploiting a Trusted Platform Module (TPM) for the generation of seeds and any required credentials. Moreover, the wallet and the developed system should have the following characteristics:

- Receive and send payments
- Keep an account balance
- Secure communication with other wallets
- Abort transactions in lack of balance
- Secure seed generation and storage
- Secure generation of required credentials

2 System Overview

As demonstrated in the sequence diagram in Figure 1, the charger acts as a server and waits for the vehicle to be connected and requests a charge. The charger generates its own address by sending a request to the IOTA tangle and receiving the new address. The charger posts (publishes) its address in the tangle using the IOTA Streams protocol where the vehicle can retrieve it. Once the vehicle has the charger address, it can send the payment in IOTA to the charger through the tangle. Once the payment is confirmed, and its acknowledgement will also be received by the vehicle, the actual charging process can start.

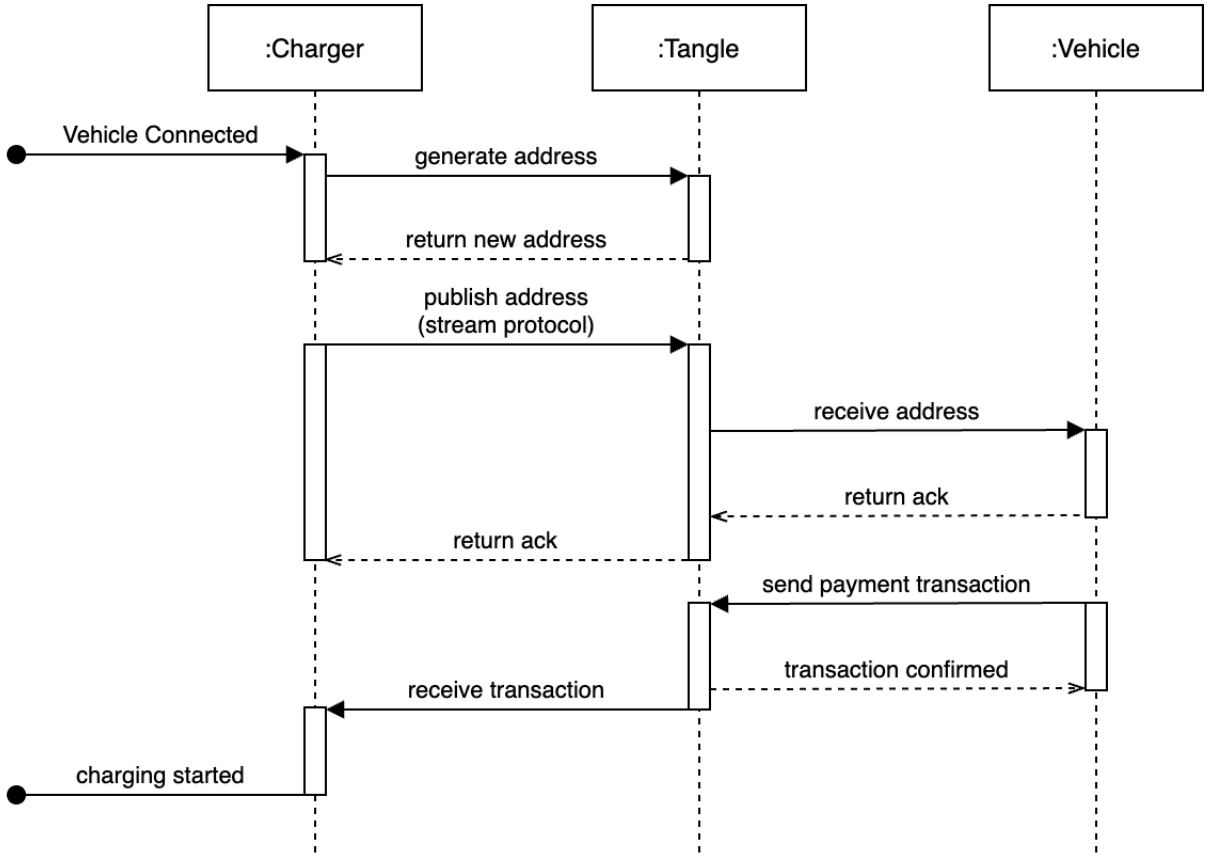

Figure 1: Sequence diagram of the automated charging process

Figure 2 presents the architecture of the implemented payment system, comprising its components. The grey components are the ones that have been set up and configured by the our team which interact with the other components that have been developed from scratch and depicted in blue (powered by Rust language) and yellow (NodeJS). In a high-level representation, however, three main components

that interact with each other are:

- Vehicle: a Raspberry Pi that has a TPM chip attached. The vehicle firefly wallet application should be able to generate and encrypt the car's seed (powered by stronghold.rs), generate addresses for the vehicle and listen to the charger address in the tangle.
- Charger: another Raspberry Pi that has a TPM chip attached. The charger wallet application should provide the same functionality as the vehicle's.
- Tangle: provides a secure communication channel between the vehicle and the charger. The tangle is responsible for validating transactions and handling the IOTA Streams protocol. This protocol is used by both parties to send and receive addresses.

The other components which are similar in both Vehicle and Charger are as follows:

- PaymentController.js: a component which handles the main functionalities of the payment system. This includes "account manager" that communicates with the API of firefly wallet to, for example, create accounts, keep the record of transactions, store tokens, etc. The "address generator" is used to generate new address via communicating with the IOTA tangle. The "balance checker" is using to check the current balance of the wallet and "transaction performer" handle all the actions associated for perfroming payment transactions.
- TPMHandler.js: a components that manages all the actions that are related to TPM, for instance, generate seeds and root key password as well as store them to and retrieve them from TPM.
- AddressPublisher.rs: a component written in Rust and is utilised to publish the generated address from PaymentController.js component such that this address is recieved by the other wallet securely through the IOTA tangle exploiting the Streams protocal.
- Firefly.rs[1]: a component developed by the IOTA community that manages all the operation of the wallet.
- Stronghold.rs[2]: a secure software implementation with the sole purpose of isolating digital secrets from exposure to hackers and accidental leaks. It uses encrypted snapshots that can be easily backed up and securely shared between devices. It is written in stable rust and has strong guarantees of memory safety and process integrity. This component stores all the data in a database powered by RocksDB.
- TPMTools.c[3]: this component includes a group of tools to manage and utilize the Trusted Computing Group's TPM hardware. TPM hardware can create, store and use RSA keys securely (without ever being exposed in memory), verify a platform's software state using cryptographic hashes and more.

[1] `https://github.com/iotaledger/wallet.rs`
[2] `https://github.com/iotaledger/stronghold.rs`
[3] `https://github.com/tpm2-software/tpm2-tools`

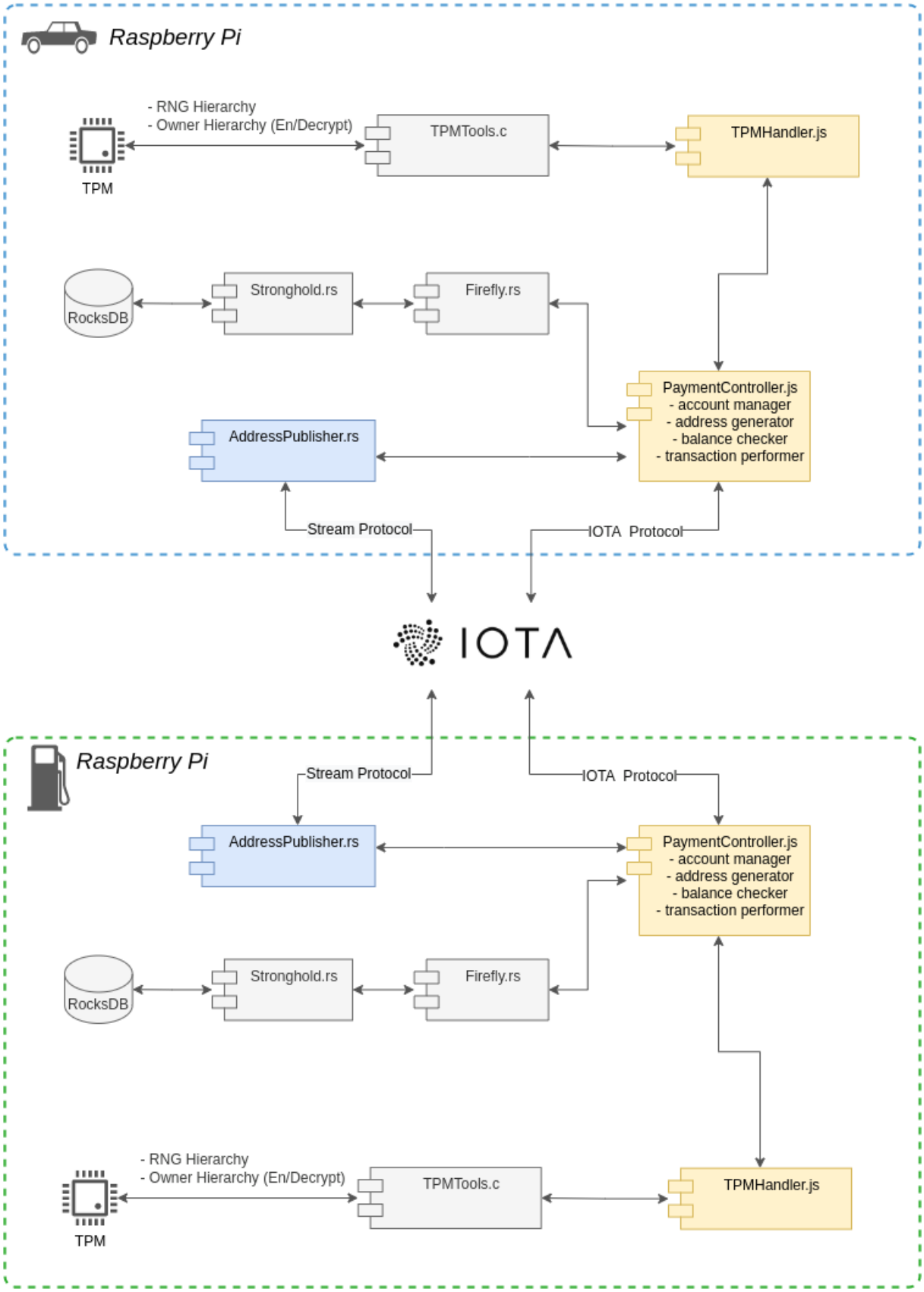

Figure 2: The components diagram of the implemented payment system

The communication flow is divided into four steps:

1. Send Address (IOTA Streams): The charger generates an address for its wallet using PaymentController.js component from the securely stored seed (managed by stronghold). The stornghold.rs is accessible only by a credential called root key. This key is generated by TPM module and stored

them internally. Eventually, the generated address is published into the tangle using addresspublisher component.

2. Receive Address (IOTA Streams): The car retrieves the charger address from the tangle and sets up the payment transaction. The components that are involved are similar to those of the charger's.

3. Send payment: The car sends its payment through the tangle to the charger.

4. Confirm payment: When the charger gets the transaction from the car, it confirms the payment to the car so the charging process can start. IOTA provides a trust layer for any device connected to the global Internet as a distributed ledger system. It also allows you and your devices to: Use the network as a source of truth for data; and Transfer value in IOTA tokens over its open network of nodes.

All of the stated processes including the creation of accounts, generating addresses, publishing messages through the tangle explioting Streams protocol, checking the balance and performing transactions are also managed by GUIs that are developed by utilising NodeJS. The detailed explanation regarding setting up the GUIs and demonstrations of their workflows are stated in the Appendix8.

In the following sections, it is described how Trusted Platform Module are set up and involved within the process of payment transactions.

3 Raspberry Pis and TPM

3.1 Setting up Raspberry Pis

We were provided with two different Raspberry Pi models for the implementation of the project:

1. Raspberry Pi 3 Model B+: quad-core A53 Processor, 1GB LPDDR2 SDRAM

2. Raspberry Pi 3 Model B+: 700MHz ARM1176JZFS Processor, 512MB SDRAM

In order to Install Raspberry Pi OS, we have used the Pi Lite OS which does not come with any desktop environment. Based on our test on these two devices and due to the limited resources, the standard version of Raspberry Pi os was barely functional.

3.2 Setting up TPMs

A TPM module (or Trusted Platform Module) is an international standard for a secure cryptoprocessor, which is a dedicated micro-controller designed to secure hardware by integrating cryptographic keys into devices.[4]. By using TPM in this project we follow the below goals:

- Encrypt and decrypt the file containing the seed, which ensures that the seed cannot be accessed by third parties. However, as we used the latest version of the IOTA wallet, we were not able to fully integrate the generated seed in the wallet since firefly does not support the feature of importing seeds from outside at the moment. The seed in the current version is generated within the wallet and protected by the stronghold.

- Encrypt and decrypt the file containing the root key of the stronghold. As mentioned in the prior section, the wallet and stronghold are now protected by a credential called root key. We have exploited TPM for the generation and storage of this credential.

- Generate a chain of random bytes, to generate the seed securely.

To some extent, the configuration of TPM is straightforward. We just need to determine the in PI OS that we want to use the SPI interface, and to load the specific overlay for TPM that enables the kernel to talk to the TPM. After reboot, the OS should recognise the TPM. However, it is possible that the TPM is not recognised by the OS same as what happened in our project; one of the devices worked properly, and the other one failed constantly to recognise the TPM. To solve this problem, we followed the instructions in the article [5]provided by the manufacturer behind the production of our TMPs. We use OPTIGA™

[4] https://paolozaino.wordpress.com/2021/02/21/linux-configure-and-use-your-tpm-2-0-module-on-linux/

[5] https://letstrust.de/archives/9-Howto-Enable-TPM-Support-on-a-Raspberry-PI-0,-0W,-1,-2,-3,-3b+-and-make-it-work-with-the-LetsTrust-TPM.html

TPM SLx 9670 TPM2.0 for the project which is compatible with Raspberry Pi via SPI interface. As stated in this article, we basically just needed to compile the kernel while manually preloading the overlay required for the os to recognise our TPM model.

```
$ sudo vim /boot/config.txt
        001    dtparam=spi=on
        002    dtoverlay=tpm-slb9670
$ sudo reboot now
```

Figure 3: TPM2.0 SPI Board on Raspberry Pi® 4 [6]

3.3 TPM Software Stack (TSS)

The TPM Software Stack (TSS) is a software specification that provides a standard API for accessing the functions of the TPM. Application developers can use this software specification to develop inter-operable client applications for more tamper-resistant computing
The purpose of the TSS Work Group is to provide a standard set of APIs for Application vendors who wish to make use of the TPM. The group works to produce a vendor-neutral specification which will provide an abstraction of the hardware differences so that application vendors can write applications that will work regardless of the hardware, Operating System, or environment that is used. The TSS also aims to provide means for applications to talk to TPMs either locally or remotely. In order to set up the TPMs, three main components must be installed:

1. A package of libraries such as ESAPI, SAPI, and TCTI [7]

    - SAPI - System API:
      The System API is a layer of the overall TSS architecture that provides access to all the functionality of a TPM 2.0 implementation. It is designed to be used wherever low-level calls to the TPM functions are made: firmware, BIOS, applications, OS, etc.
    - ESAPI - Extended System API:
      ESAPI is the layer on top of SAPI. This provides for enhanced context management and Cryptographic operations. Even though ESAPI provides an easier way to use the TPM, using the ESAPI layer of APIs still requires an in-depth understanding of the TPM's internal workings.
    - TCTI - TPM Command Transmission Interface:
      The TCTI is an IPC abstraction layer used to send commands to and receive responses from the TPM or the TAB/RM. This provides multiple interfaces between SAPI and the lower

[6] https://www.infineon.com/dgdl/Infineon-OPTIGA_SLx_9670_TPM_2.0_Pi_4-ApplicationNotes-v07_19-EN.pdf
[7] https://dev.to/nandhithakamal/how-to-tpm-part-2-55ao

hardware layers depending on the type of TPM (physical TPM, tpm simulator, etc.) being used

2. Resource Manager (abrmd):
   The resource manager manages the TPM context in a fashion similar to the virtual memory manager. The TPM has a small memory capacity and is easily constrained by how many resources can be loaded simultaneously. The resource manager is responsible for swapping in and out of memory the various resources required by a process. In Unix, hardware devices are treated as files and can be accessed using file paths. Likewise, the tpm hardware can be accessed either using /dev/tpm0 or using /dev/tpmrm0.
   /dev/tpm0 is the direct path to the TPM hardware and at any given time, only a single process can access the TPM using this path. It is very common for multiple processes to require access to TPM. For this purpose, the /dev/tpmrm0 is used. This is the TPM's resource manager and using these multiple processes can use the TPM simultaneously. The tpm2-abrmd is a system daemon that implements the TAB, the resource manager specifications. But since The recent versions of the kernel (starting from 4.12) have an in-kernel resource manager, we did not need the tpm2-abrmd and can proceed using the in-kernal rm.

3. tpm2-tools:
   This is an end-user command line tool that can be used to generate keys, view capabilities, sign, hash, unseal, etc.

There might be some dependency issues during the installation of the components. For example, in this project, some dependencies could not be installed from repositories, thus compiling from the source.

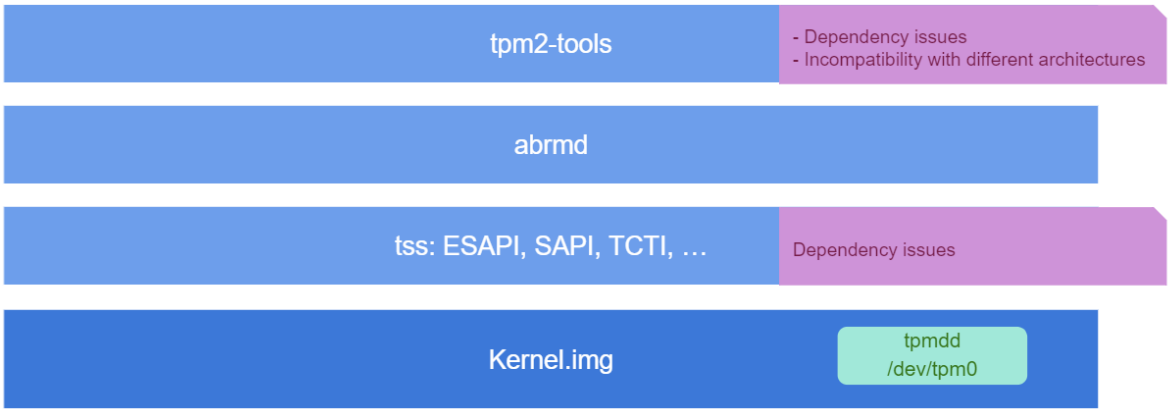

Figure 4: TPM Software Stack and Issues in this project

As another case in point, when we were compiling tpm2-tools, in a C file, the compiler was complaining about the formatting issue, which was apparently reported[8] and patched at some point, but still, it could be problematic. So in this particular issue, by replacing the format string with llu instead of lu, this file could be compiled in our environment.

```
  CC       lib/libcommon_a-tpm2_util.o
In file included from ./tools/tpm2_tool.h:11,
                 from lib/tpm2_util.c:20:
lib/tpm2_util.c: In function 'tpm2_util_tpm2_nv_to_yaml':
lib/tpm2_util.c:1282:26: error: format '%lu' expects argument of type 'long unsigned int', but argument 2 has type 'UINT64'[-Werror=format=]
 1282 |         tpm2_tool_output("counter: %lu\n", v);
  ~/uni_passau/s22/lab/project/task2/iota-charging-payments-system  main !6 ?1
❯
```

Figure 5: A Sample of Formatting Issuse

4 Seed

4.1 Secure seed generation

Up to this point, we have configured our raspberry Pis and set up the TPMs in addition to the software stack. A reason for using TPM in this project is to generate seeds securely. Everything inside TPM is

[8] https://github.com/tpm2-software/tpm2-tools/issues/1134

accessed through handles, which point to predefined objects, one of the objects in the hierarchy. In TPMs there are usually some fuses that define seeds (completely hardware base), and then the seeds are used as a basis of key generation functions of 4 hierarchies. A hierarchy is a tree structure with the parent key at the root. It wraps around and encrypts all of its children's keys. Keys can be created under any of the following 4 hierarchies:

- Endorsement: The endorsement hierarchy is the privacy-sensitive tree and is the hierarchy of choice when the user has privacy concerns. TPM and platform vendors certify that primary keys in this hierarchy are constrained to an authentic TPM attached to an authentic platform.
- Owner: Also called storage hierarchy, this is intended to be used by the platform owner, i.e., the device owner or user.
- Platform: Intended to be under the control of the platform manufacturer. This is used by the BIOS and System Management Mode (SMM).
- Null: This hierarchy is used to generate true random number.

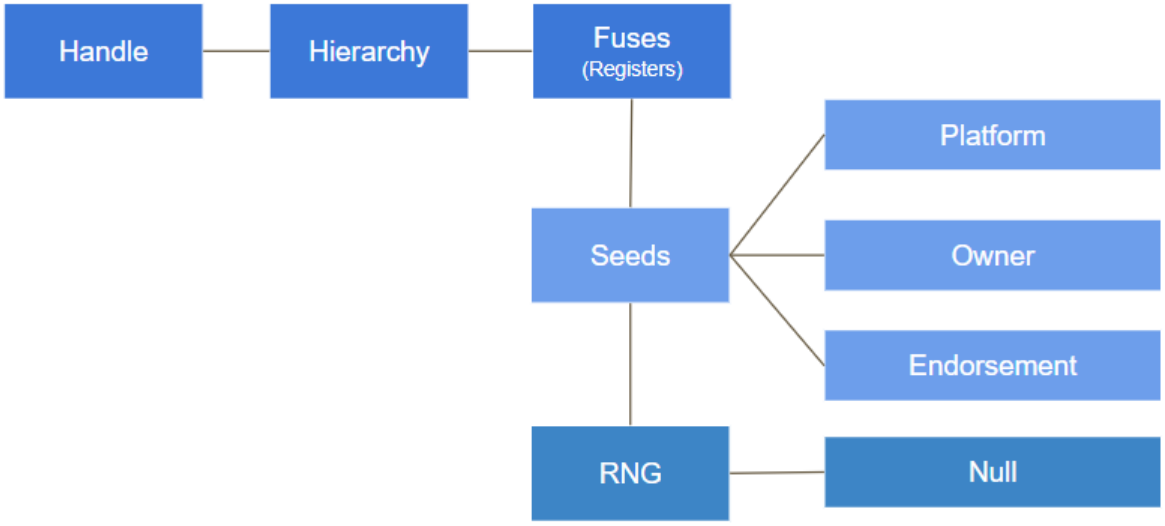

Figure 6: TPM2 Hierarchies, adapted from "Technical Introduction to the Use of Trusted Platform Module 2.0 with Linux" [10]

In order to generate the seed, we used the following commands. In generating the seed, it is important to convert byte to tryte format because IOTA does not work with byte, but trytes.

```
import fs from 'fs';
import path from 'path';
import {exec} from 'child_process';
import * as Trytes from 'trytes';

sh("tpm2 getrandom 32").then(r => {
        const stdout = r.stdout;
        const trytes = Trytes.encodeTextAsTryteString(stdout);
        const seed = trytes.substr(0, 81);
        console.log(seed);
    }
);
```

4.2 Secure seed storage

We can use TPM for securely encrypting and then storing our seeds and root keys as well. In order to do so we can basically choose either of the following hierarchies:

[10] https://lenovopress.lenovo.com/lp0599.pdf

- Platform
- Owner
- Endorsement

In this project, we use the Owner hierarchy. It is important to mention that the procedure for all of the hierarchies is the same. The first step we should do is to create a Primary Key which will be then used to generate a private key. The private key will be locked in TPM, and it cannot be moved out. Based on this key, we can generate other keys and have our own key hierarchy. For example, in our case, as performing encryption using an asymmetric key is time-consuming due to the highly limited resources in TPM. Therefore, We generate a symmetric key, which essentially is encrypted and protected by this primary key, and then we use our symmetric key for the encryption of our data.

```
##
# Offline
##

tpm2 createprimary -C TPM_RH_OWNER -G rsa2048 -c primaryKey.ctx
tpm2 create C primaryKey.ctx G aes128 c storageKey.ctx r storageKey.priv
tpm2 load C primaryKey.ctx -c storageKey.ctx
dd if=/dev/urandom of=iv.dat bs=16 count=1
tpm2 encryptdecrypt c storageKey.ctx o encryptedSeed.enc t iv.dat

##
# Online
##

import {composeAPI, generateAddress} from '@iota/core'
import fs from 'fs';
import path from 'path';
import {exec} from 'child_process';

async function readEncryptedSeed(directory) {
    console.log("\t-- Read encrypted seed...");
    const key = directory + "../key.ctx";
    const input = directory + "../secret.enc";
    const iv = directory + "../iv.dat";

    return await sh("tpm2 encryptdecrypt -c " + key + " -D -i " + input + " -t " + iv);
}
```

5 Receive and Send Payments

Having done all the previous configurations, we are ready to move on to the most important part of the project; transactions. Doing a transaction includes sending and receiving payments. But before then, let's discuss the current state of the IOTA project. IOTA 1.0 is already deprecated. IOTA 1.5, is the current version of the IOTA, and IOTA 2.0 is the next version of this project.

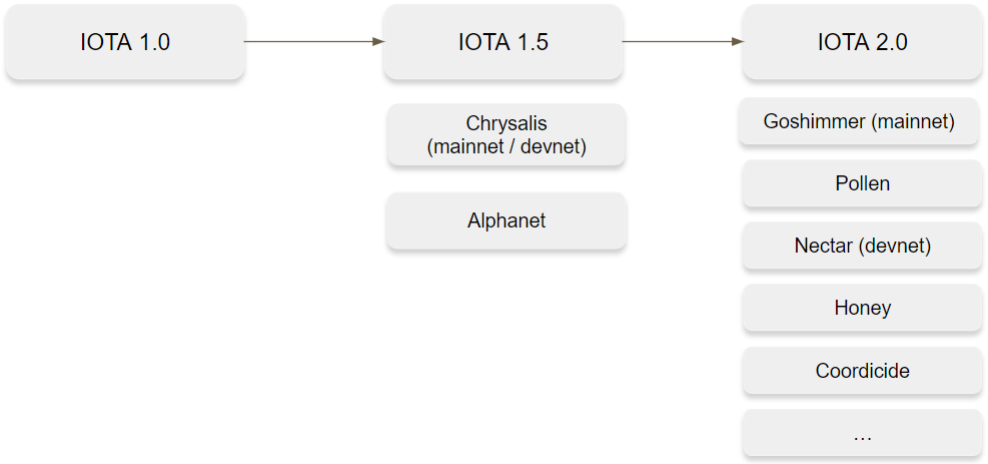

Figure 7: IOTA Development Phases [11]

There are many networks, nodes, and different wallets that have been developed mostly for making this transition from iota 1.5 to 2.0 occur. However, we were not able to find one working network properly in the latest version of IOTA, thus using IOTA 1.5 and Chrysalis APIs.

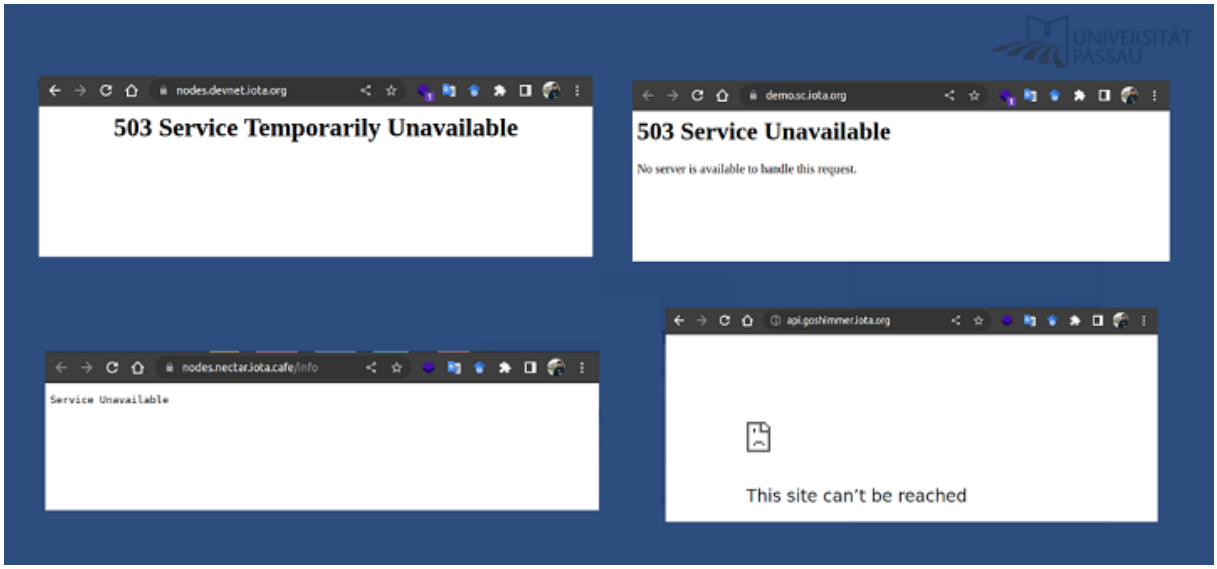

Figure 8: APIs for different iota 2 networks, none of which are working, particularly nectar which is the main development network

The wallet plays an important role in sending transactions. Currently, Firefly Wallet is used instead of Trinity because Trinity wallet has now been discontinued. Firefly Wallet is written in Rust. With the updated wallet library, developers do not need to use a self-generated seed. The seed is created and stored in Stronghold, our in-house built security enclave. It is not possible to extract the seed from Stronghold for security purposes.[12] The stronghold in the Firefly wallet is protected using a password. Since the seed is created and stored in the stronghold, we need to protect the stronghold using a strong password. To do so, we used TPM to generate and store the password of the stronghold.

5.1 Create an account

Every car and charger need an account to send and receive transactions. We implemented the following module in this project to create an account. The module is written in JavaScript.

[11] https://blog.iota.org/iota-2-0-introducing-pollen-nectar-and-honey-de7b9c4c8199/
[12] https://wiki.iota.org/introduction/guides/developer#seed

```
/**
 * This function creates a new database and account for vehicle/charging station
 *
 */

async function create_charger_account() {
    const { AccountManager, SignerType } = require('@iota/wallet');

    const manager = new AccountManager({
        storagePath: './charger-database',
    });

    try {
        manager.setStrongholdPassword(retrieveRootKey());
        let account;
        try {
            account = manager.getAccount('Charger');
        } catch (e) {
            console.log("Couldn't get account, creating a new one");
        }

        // Create account only if it does not already exist
        if (!account) {
            manager.storeMnemonic(SignerType.Stronghold);
            account = manager.createAccount({
                clientOptions: {
                    node: { url: 'https://api.lb-0.h.chrysalis-devnet.iota.cafe' },
                    localPow: true,
                },
                alias: 'Charger',
            });
            console.log('Account created:', account.id());
        }

        const synced = await account.sync();
        console.log('Synced account', synced);
    } catch (error) {
        console.log('Error: ' + error);
    }
}

create_charger_account();
```

5.2 Generate an address

To send a transaction both sender and receiver need an address. If you use an address to send a transaction, you should never use it again for either sending or receiving transactions because each time you send a transaction from an address, a part of the private key is revealed. But you can use an address to receive transactions as many times as you want.
We implemented the following module in JavaScript to generate a new address.

```
/**
 * This function generates a new address for the charging station.
 *
 * Sending addresses: Once you have sent a transaction from an address, you should never use
 * this address again. Each time you send a transaction from an address, a part of the private key
 * is revealed.
 *
 * Receiving addresses: you can receive as many transactions as you want to an address, but
 * once you make a transaction from this address, you should not reuse this address again for
 * receiving or sending transactions.
```

```
 *
 */

async function generate_address() {
    const { AccountManager } = require('@iota/wallet');
    const manager = new AccountManager({
        storagePath: './charger-database',
    });

    manager.setStrongholdPassword(retrieveRootKey());
    const account = manager.getAccount('Charger');
    console.log('Account:', account.alias());

    // The account should be always synced before any operation
    await account.sync();
    console.log('Syncing...');

    const NEW_ADDRESS = false;
    if (NEW_ADDRESS) {
        const address = account.generateAddress();
        console.log('New address:', address);
    } else {
        // We can also get the latest unused address:
        const addressObject = account.latestAddress();
        console.log('Address:', addressObject.address);
    }

    // Using the Chrysalis Faucet to send testnet tokens to the address:
    console.log(
        'Fill your address with the Faucet: https://faucet.chrysalis-devnet.iota.cafe/',
    );

    const addresses = account.listAddresses();
    console.log('Addresses:', addresses);
}

generate_address();
```

5.3 Check Balance

We have written the following module in JavaScript to check the balance of the wallet.

```
/**
 * This function checks the balance for the account of the charging station
 *
 */

async function check_balance() {
    const { AccountManager } = require('@iota/wallet');

    const manager = new AccountManager({
        storagePath: './charger-database',
    });

    manager.setStrongholdPassword(retrieveRootKey());
    const account = manager.getAccount('Charger');

    console.log('Account:', account.alias());

    // Always sync before doing anything with the account
    await account.sync();
    console.log('Syncing...');
```

```
    console.log('Available balance', account.balance().available);
}

check_balance();
```

5.4 Send Transactions

In order to send transactions to a specific address, we wrote the following module in JavaScript.

```
/**
 * This example sends IOTA tokens to an address.
 *
 */

async function send_iota_token() {
    const {
        AccountManager,
        RemainderValueStrategy,
    } = require('@iota/wallet');

    const manager = new AccountManager({
        storagePath: './car-database',
    });

    manager.setStrongholdPassword(retrieveRootKey());
    const account = manager.getAccount('Car');

    console.log('Alias', account.alias());
    console.log('Syncing...');
    await account.sync();
    console.log('Available balance', account.balance().available);

    const address = retrieveLatestAddress();
    // 'atoi1qz3ts7rwfyp4eegyccr3qlucnqpjm6wpme4fx4n0g72psrk072yvgfqwwvy'
    const amount = getPrice();

    const response = await account.send(address, amount, {
        remainderValueStrategy: RemainderValueStrategy.reuseAddress(),
    });
}

send_iota_token();
```

6 IOTA Streams

6.1 Defining the Scenario

In this project, we are trying to implement a scenario in which a car goes to the charging station and sends payment to it. After completing the transaction, the car starts being charged. However, the car must be aware of the charger's address. The charger informs the car of its address through the IOTA Streams protocol.

6.2 Introduction to Streams

IOTA Streams is a work-in-progress framework for building cryptographic messaging protocols[13]. Streams ships with a built-in protocol called Channels for sending authenticated messages between two or more

[13] https://www.iota.org/solutions/streams

parties on the Tangle. As a framework, Streams allows developers to build protocols for their specific needs. At the moment, IOTA Streams includes the following crates[14]:

- Channels Application featuring Channels Application.
- Core layers featuring spongos automaton for sponge-based authenticated encryption, pre-shared keys, pseudo-random generator;
- Keccak for core layers featuring Keccak-F[1600] as spongos transform;
- Curve25519 asymmetric crypto featuring Ed25519 signature and X25519 key exchange;
- DDML featuring data definition and manipulation language for protocol messages;
- Application layer common Application definitions.

6.3 How does Streams work?

The Streams protocol involves the components stated below [15]:

- Streams: All branches of a Stream reference a common root branch and state associated with the original publisher, thus guaranteeing data authenticity.
- The Tangle: Data is transferred and secured over an immutable distributed ledger, the Tangle.
- Publishers: Publishers can send non-encrypted, open, data in a Stream for everyone to see. Publishers can also restrict access to their data or make it completely private using public key encryption.
- Subscribers: Subscribe to different Streams and pull information from the Tangle to integrate into applications and guide your decision-making. Subscribers can also contribute data to a Stream using various types of cross-referencing messages.

6.4 Implemented Module

In order to implement IOTA Streams, we must write the modules in Rust. We have developed a single module that handles both publishing and subscribing process of exchanging addresses between two wallets. In the following snippet, which includes the main function of our executable, we can either publish or subscribe by the arguments that we pass in CLI:

```
use anyhow::Result;
use std::env;

mod single_branch_public;

#[tokio::main]
async fn main() -> Result<()> {
    let node_url = "https://api.lb-0.h.chrysalis-devnet.iota.cafe";

    let args: Vec<String> = env::args().collect();
    let api = args[1].clone();

    if api == String::from("publish") {
        let addr: &str = &args[2];
        single_branch_public::publish(node_url, addr).await?;
    } else {
        let announcement_link = args[2].clone();
        single_branch_public::subscribe(node_url, announcement_link).await?;
    }

    Ok(())
}
```

[14] https://github.com/iotaledger/streams
[15] https://www.iota.org/solutions/streams

The main.rs will call the single-branch-public.rs crate which is responsible to publish/subscribing to a message. Here, we use our generated address as a message that needs to be send from the charger to the vehicle:

```
use iota_streams::{
    app::transport::tangle::client::Client,
    app_channels::api::tangle::{Address, Author, Bytes, ChannelType, Subscriber},
    core::{println, Result},
};

use core::str::FromStr;
use iota_streams::app_channels::api::tangle::MessageContent;
use rand::Rng;

pub async fn publish(node_url: &str, addr: &str) -> Result<()> {
    let ALPH9: &str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ9";
    // Generate a unique seed for the author, this is only used for the streams protocol
    let seed: &str = &(0..81)
        .map(|_| {
            ALPH9
                .chars()
                .nth(rand::thread_rng().gen_range(0, 27))
                .unwrap()
        })
        .collect::<String>();

    // Create the Transport Client
    let client = Client::new_from_url(node_url);

    // Generate an Author
    let mut author = Author::new(seed, ChannelType::SingleBranch, client.clone());

    // Create the channel with an announcement message. Make sure to save the resulting link somewhere,
    let announcement_link = author.send_announce().await?;
    // This link acts as a root for the channel itself
    let ann_link_string = announcement_link.to_string();

    println!(
        "Address: {}\nAnnouncement Link: {}\nTangle Index: {:#}\n",
        addr,
        announcement_link.to_msg_index(),
        ann_link_string
    );

    let mut prev_msg_link = announcement_link;
    let (msg_link, _seq_link) = author
        .send_signed_packet(
            &prev_msg_link,
            &Bytes::default(),
            &Bytes(addr.as_bytes().to_vec()),
        )
        .await?;
    prev_msg_link = msg_link;

    Ok(())
}

pub async fn subscribe(node_url: &str, ann_link_string: String) -> Result<()> {
    // In their own separate instances generate the subscriber(s) that will be attached to the channel
    let client = Client::new_from_url(node_url);
    let mut subscriber = Subscriber::new("Car", client);

    // Generate an Address object from the provided announcement link string from the Author
```

```
    let ann_address = Address::from_str(&ann_link_string)?;

    // Receive the announcement message to start listening to the channel
    subscriber.receive_announcement(&ann_address).await?;

    let retrieved = subscriber.fetch_all_next_msgs().await;

    let processed_msgs = retrieved
        .iter()
        .map(|msg| {
            let content = &msg.body;
            match content {
                MessageContent::SignedPacket {
                    pk: _,
                    public_payload: _,
                    masked_payload,
                } => String::from_utf8(masked_payload.0.to_vec()).unwrap(),
                _ => String::default(),
            }
        })
        .filter(|s| s != &String::default())
        .collect::<Vec<String>>();

    print!("Retrieved messages: ");
    for i in 0..processed_msgs.len() {
        print!("{}, ", processed_msgs[i]);
    }
    println!();

    Ok(())
}
```

7 IOTA Evaluation

7.1 Reasons to give up on IOTA

In the following section, we have summarised the most notable reasons why IOTA might not be an appropriate solution for cryptocurrency in the area of the Internet of Things.

7.1.1 Address reusing

The first problem is related to address generation. IOTA uses a one-time cryptographic signature in which the addresses should not be reused. The wallets have to check against a list, before each transaction to see if the address has been used before and to avoid user errors. With the constant growth of the list, the search will take a longer time and the scalability will never become high[16].

7.1.2 Lack of a descent wallet

We have used the latest version of the wallet, called firefly, which is still in the development phase, but the prior version, the trinity wallet, which is used widely in mainnet is vulnerable so much so that has been hacked a couple of times, such that in one incident only, iota tokens worth 4 million dollars were stolen[17]. Actually, this was one of our motivations to use the latest version of the wallet as well as exploit TPM to increase the security of the wallet.

7.1.3 Unstable Networks

As mentioned earlier, all of the IOTA networks are highly unstable. There are in fact many reasons involved, albeit, the most notable of which is related to its fundamental design. IOTA has had trouble

[16] https://www.researchgate.net/publication/334286896_Curbing_Address_Reuse_in_the_IOTA_Distributed_Ledger_A_Cuckoo-Filter-Based_Approach

[17] https://news.bitcoin.com/iota-attacked-for-subpar-wallet-security-following-4m-hack/

getting enough nodes on the network. Due to the fee-free status and the fact that all tokens already exist, there is no monetary incentive for operators to join the network, which is, after all, the pillars of the entire network to operate it and, if necessary, to bear, for instance, the server costs.

7.1.4 Broken Promises

IOTA founders started changing everything that was on the original roadmap year by year. To name a few[18]:

1. JINN: Their decision to pursue ternary computing instead of the binary was pretty radical. The whole world's computing processors run on binary. Ternary on the other hand has only been discussed in the research world while no major tech company seriously pursuing this. JINN was supposed to be the revolutionary microprocessor for Ternary computation to let any IOT device in the world be able to confirm and send transactions on the network. In 2020, with the departure of one of the founders, JINN declared as a failed project.
2. Qubic: The other promise was the Qubic project. Qubic protocol was supposed to sit on top of IOTA and use the tangle as a backbone. It would enable features like outsourced computing and smart contracts that were not available on IOTA natively. However, in 2020, they decided to just focus on the smart contracts layer instead.
3. Decentralisation: The biggest broken promise was that IOTA would become decentralised, while there is still the coordinator controlling the IOTA network now. The coordinator is a centralised app that protects the network and maintains consensus and has become the bottleneck and single point of failure for the entire platform.

7.1.5 No in-practice implementation

Over the years, IOTA had many impressive partnerships. The announcements of these partnerships even helped the price of tokens to pop up, but most of them were not more than projects to explore the technology. It has been several years now and none of these produced anything significant; no real word implementation, and no adoption.

7.2 Alternatives for IOTA

7.2.1 Nano:

Nano is a peer-to-peer open-source cryptocurrency. The currency is based on a directed acyclic graph data structure and distributed ledger, making it possible for Nano to work without intermediaries. To agree on what transactions to commit (i.e. achieving consensus), it uses a system called "Open Representative Voting" with weight based on the number of currency accounts hold[19]

Advantages
Nano shares many of the characteristics that give all cryptocurrencies value, including durability, portability and scarcity. The maximum supply of Nano is 133,348,297 NANO.[20]

- Near-instant
- No transaction fees
- No inflation (since all the coins to ever exist are already distributed
- No miners (saving electricity)
- No incentive structure

How does Nano work?

Instead of a single universal blockchain, nano uses a block-lattice structure. Every wallet has its own personal blockchain. Your blockchain contains your entire transaction history with each block containing only one single transaction. Every time a transaction is made, two new blocks are created; one on

[18] https://www.youtube.com/watch?v=k_pfuzuL2f8&t=802s
[19] https://ieeexplore.ieee.org/document/9136665
[20] https://www.kraken.com/learn/what-is-nano>

the sender's blockchain, and one on the recipient's blockchain. Being peer-to-peer, there is no need for transaction fees since the only parties involved are the ones taking part in the transaction and you also don't have to wait for anyone. To place your transaction in a block, although transactions appear instant, this isn't exactly the case. When a block is created, a small amount of proof of work must be done; a small cryptographic puzzle. This only takes about 30 seconds but can be precalculated. So when the transaction is initiated, it immediately goes through the proof of work and does not strain your computer, and is very noticeable it is primarily to prevent a malicious party from spamming. Looking for any possibility of fraud in a fishy activity, and any scandals, Nano seems pretty clean.

7.2.2 IoTeX

IoTeX combines secure blockchain tech with the Internet of Things (IoT). The IoTeX blockchain already powers real devices, including award-winning blockchain-powered cameras from Consumer Electronic Show (CES) and the pebble geo device, perfect for supply chain optimization in any industry. IoTeX seems to be a strong contender to IOTA as well. They are life already after all. They've been live for two and a half years actually.

- Privacy preserving
- Scalable
- supports around 3 000 transactions per second
- Finality in under five seconds (Finality is the assurance or guarantee that cryptocurrency transactions cannot be altered, reversed, or cancelled after they are completed.)
- so cheap that their transactions cost like fractions of a cent (so with just one dollar you can send thousands of transactions and which is necessary for the internet of things)
- Fully Decentralized(Unlike IOTA They do not have a central coordinator and have never faced a network shutdown before, and they even say that they cannot shut it down even if they wanted to)

Adoption In addition to the advantages stated, IoTeX has been very successful in being adopted in the IoT industry. They have built a blockchain-based home security camera called ucam which is powered by their platform.

- Secured with their blockchain-based login
- End-to-end encryption
- Edge computing architecture

They also have a device called the pebble which is a powerful sensor that can capture data like temperature light location, motion, and more. They designed it to be tamper-proof on both the hardware and software sides. There were hundreds of teams building projects that utilize pebble; some of those may appear in apps that businesses or consumers can use with their devices soon.

8 Appendix

8.1 GUIs: Setups and Processes

The description regarding the different sections of the GUIs and the guidelines for their setups and configurations are presented as follows.

8.1.1 Car UI

setup

- Setup the Car UI under the repository iota-asl-gui through the README.md file.
- Run the Compile command 'npm run serve' (Also mentioned in the README.md)
- Open the localhost link on a browser.

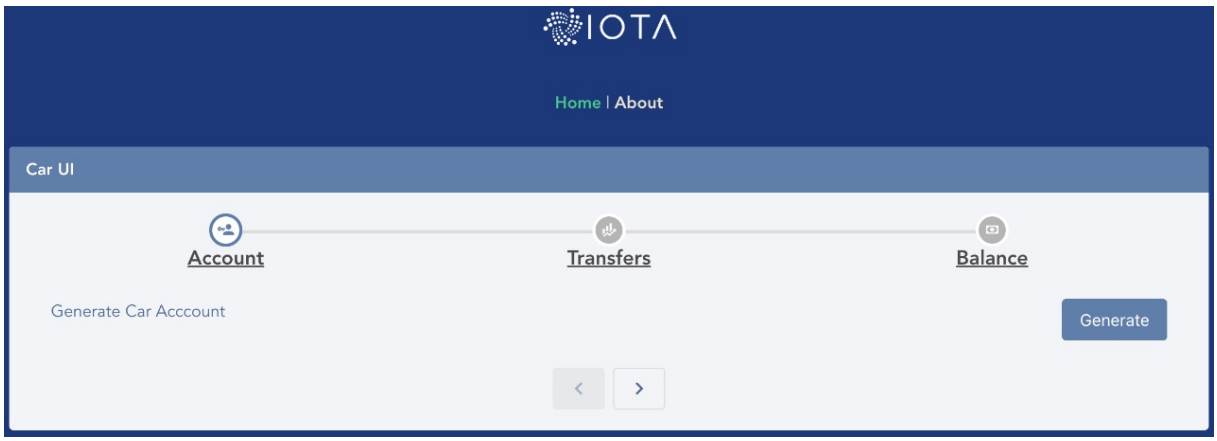

Figure 9: Main page in the car GUI

Generate Account
To generate the account and address for the car, hit the Generate Button under the Account tab of the UI as seen in Figure 10. You should see a notification as in the figure below.

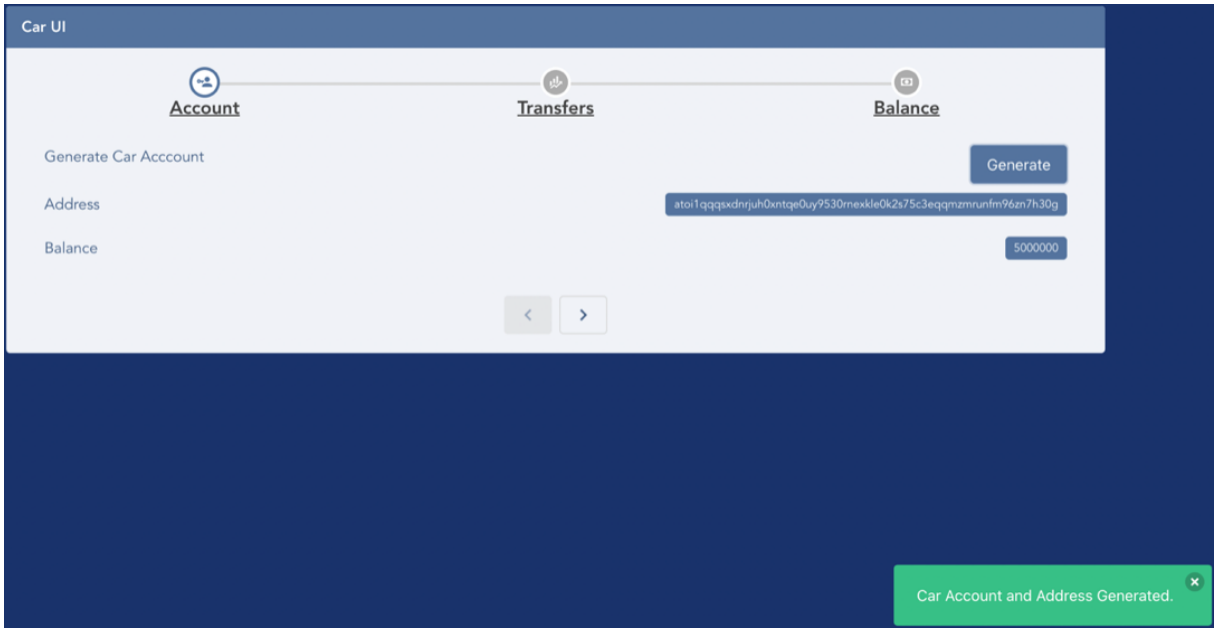

Figure 10: Generate account and address for the car

Transfer Transactions
Hit the Right Chevron Arrow to enter the Transfers sections.

Figure 11: Transfer transaction in the car UI

A minimum amount has been preset but you can change that amount to any amount above 100000 otherwise it will throw an error. The Check button is a manual override to see whether or not the charger address has been published to the Car's UI. A Successful transaction has been depicted in the figure below:

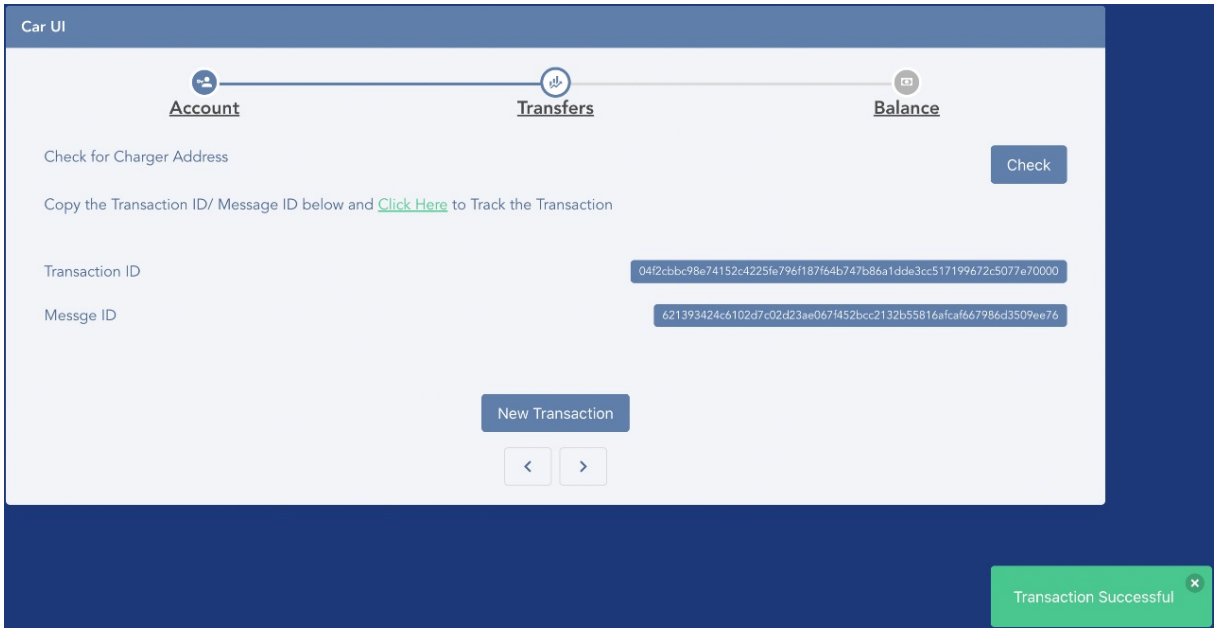

Figure 12: Successful transaction in the car UI

Clicking on New Transaction Button on the section in Figure 12 takes you back to the main Transfers page (Figure 11).

If the Charger Address is not published the following error can be seen.

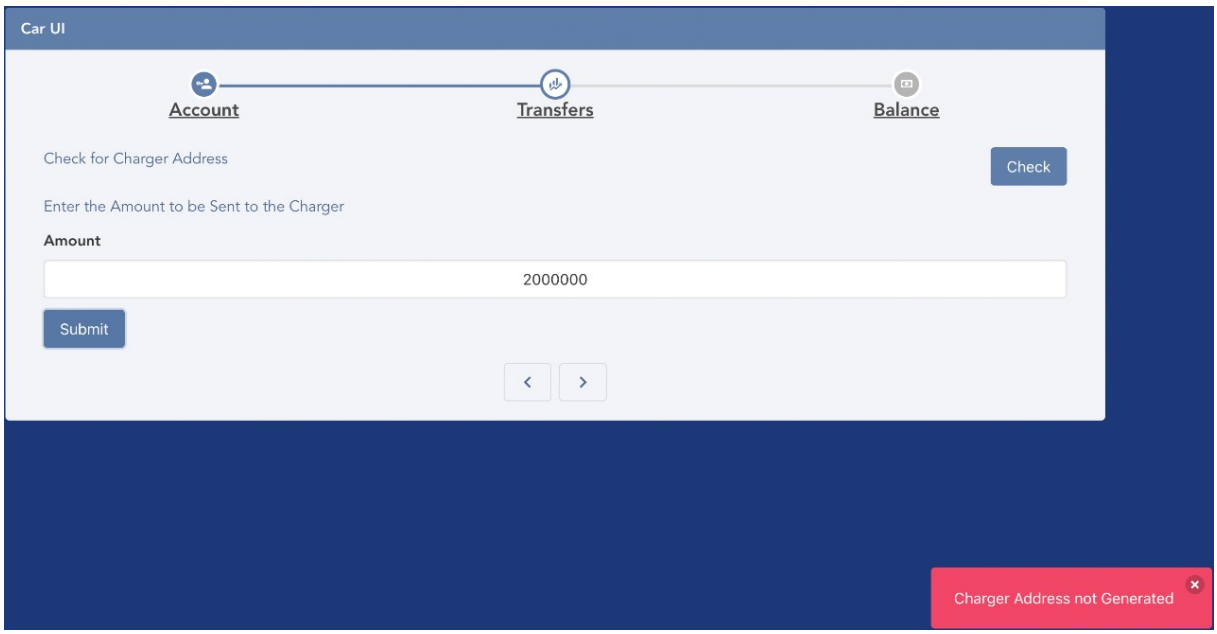

Figure 13: Charger address is not generated

If the amount is less than 100000, the following error is seen.

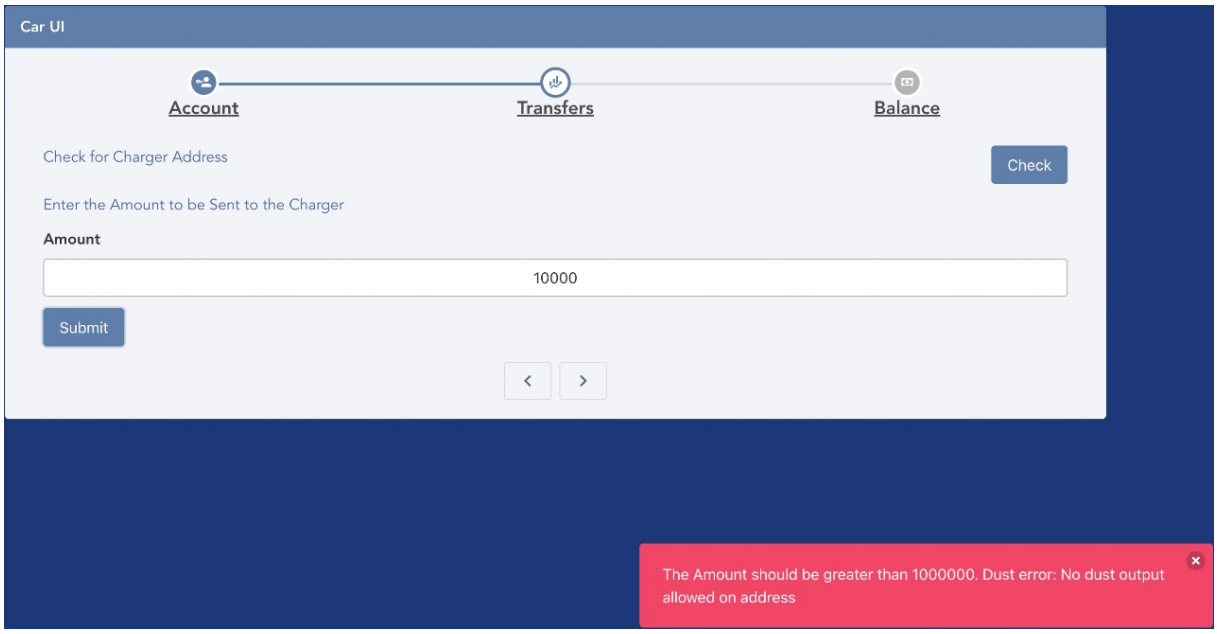

Figure 14: The entered amount is less than minimum

Balances

Hit the Right Chevron Arrow to Enter the Balances Section of the UI and the latest Balance will be fetched. You should see a notification as in the figure below.

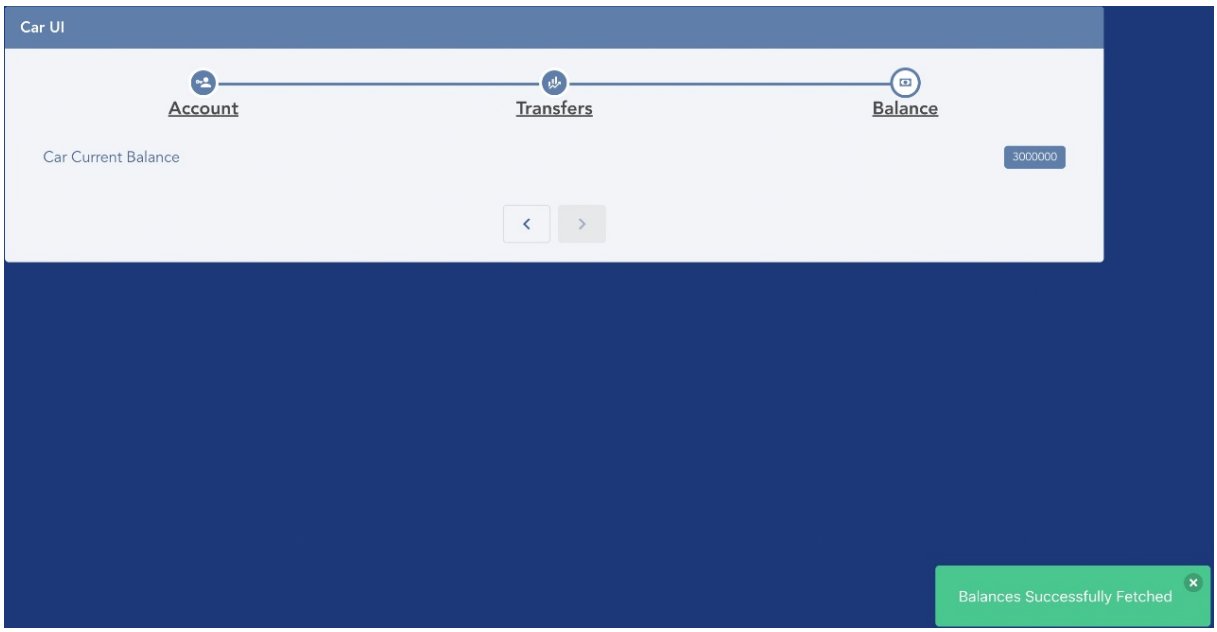

Figure 15: Check the balance in the car GUI

8.1.2 Charger UI

Setup

- Setup the Car UI under the repository iota-asl-gui through the README.md file.
- Run the Compile command 'npm run serve' (Also mentioned in the README.md)
- Open the localhost link on a browser.

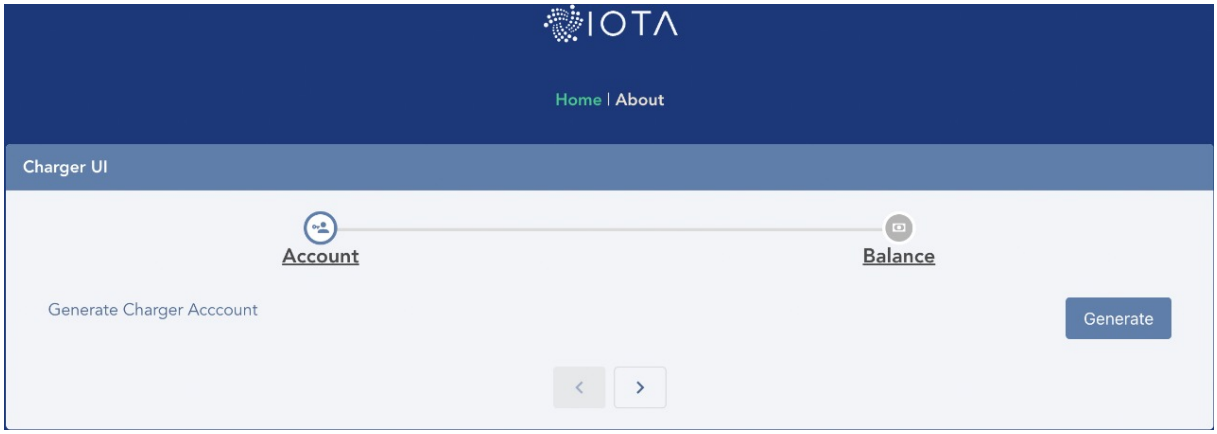

Figure 16: Main page in the Charger GUI

Generate Account

To generate the account and address for the Generate hit the Generate Button under the Account tab of the UI as seen in Figure 16. You should see a notification as in the figure below.

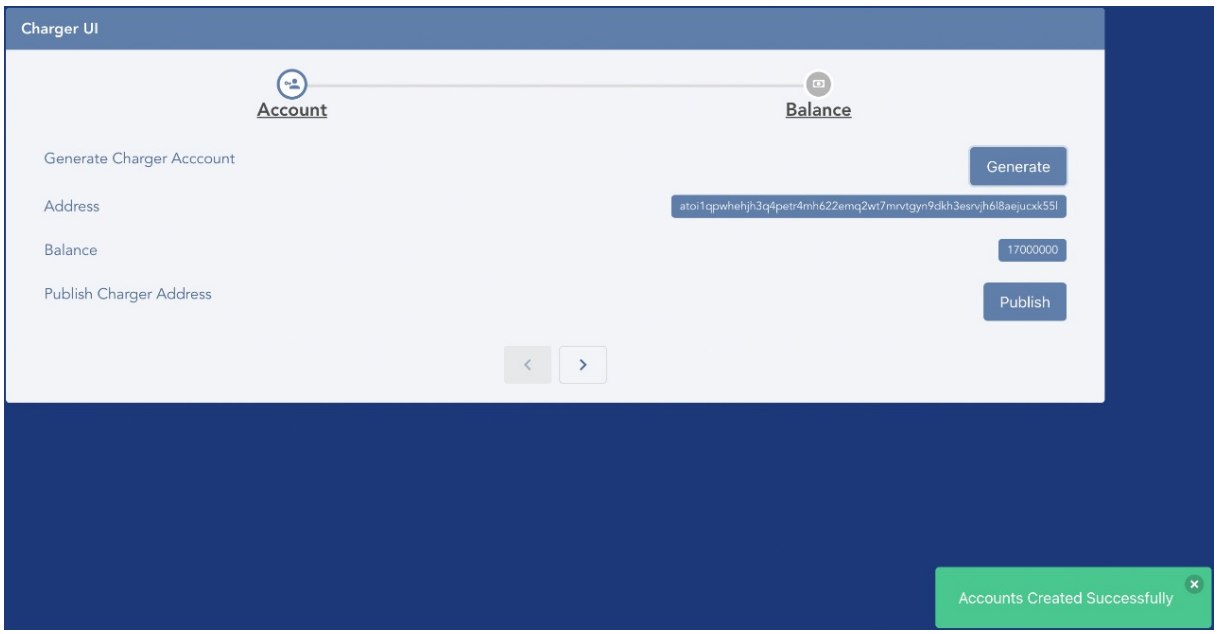

Figure 17: Generate account and address for the charger

Publish

After generating the account, hit publish so the Charger's Address can be published through the IOTA tangle using Streams protocol such that it can be fetched from the GUI of the Car. You should see a notification as in the figure below.

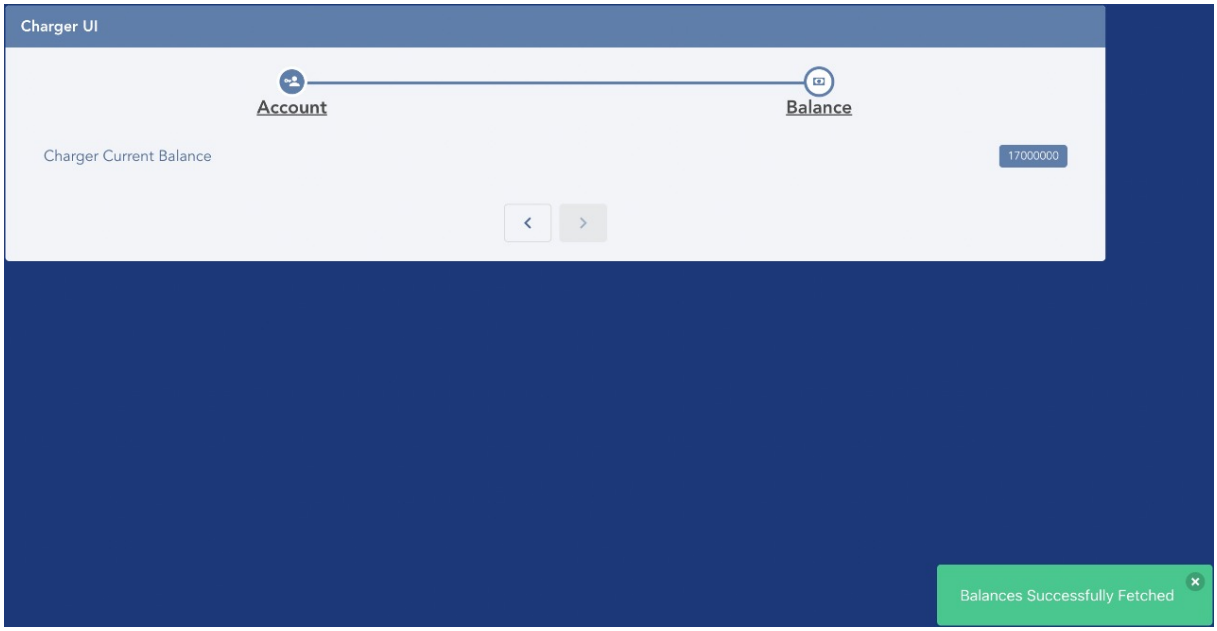

Figure 18: Publish address to the tangle using streams protocal