E-Mine 1: Token Creator to Incentivize e-waste Collection and Recycling

FuturICT 2.0 Challenge 1 - Token Creator
Blockchain and Internet of Things School (BIOTS) 2018

ETH Zürich, 30.04.2018

Team E-Mine 1 members:
Vikas Patil
Andrin Bertschi
Nenad Katanic
Matija Piskorec
Julia Füreder

All contributors contributed equally to this report.

The software code, which is part of this report, is open source and available at the URLs listed below.

This project report was written as part of the spring 2018 course 'Blockchain And the Internet of Things (851-0591-01L)' run by M. Dapp, S. Klauser, and D. Helbing.

This report is licensed under a CreativeCommons licence CC BY-SA v4.0.

Code repositories:
https://github.com/e-mine1/e-mine-web
https://github.com/e-mine1/GenesisApp

Outline of the documentation

This document summarizes the contributions of the team “E-Mine 1” for a hackathon, organized as part of the Blockchain and Internet of Things School (BIOTS) 2018. Major contributions are:

- Development of the E-Mine Token generator [1] - a web service for creation and deployment of Ethereum smart contracts.
- Extension of the GenesisApp [2] - an Android application for easy generation of custom token contracts. We extended the GenesisApp with several standard token templates, and implemented the compilation and deployment of tokens by requesting it through the E-Mine Token Generator web service.

In order to showcase the utility of our E-Mine Token generator we used it as a part of the token-based solution for incentivizing e-waste recycling. This use case was developed in collaboration with the “E-Mine 2” team [3] which participated in the FuturICT 2.0 Challenge 2 - Token Obtainer.

[1] E-Mine Token Generator, https://github.com/e-mine1/e-mine-web
[2] Fork of the GenesisApp, https://github.com/e-mine1/GenesisApp
[3] E-mine2 Github organization, https://github.com/e-mine2

1. Introduction

The e-waste challenge. Electronic waste, or e-waste, is a term used for electrical and electronic equipment that has reached its end of useful life or has become obsolete over time. Fast-changing market demands, coupled with aggressive marketing and "planned obsolescence" by manufacturers of these devices has generated a huge volume of e-waste. According to a report by the United Nations University[4], an estimated 44.7 million metric tonnes of e-waste was generated globally in the year 2016 (equivalent to 6.1 kilogram per inhabitant). Barely 20% of this was recycled through appropriate channels. The global annual e-waste generation is expected to rise to 52.2 million metric tonnes by 2021.

There are various types of e-waste, as given in Table 1[5].

Table 1. E-waste categories and examples

E-waste category	Examples
Information and communication	Computer, tablet, mobile phone
Large household appliances	Refrigerator, air conditioner, washing machine
Small household appliances	Iron, dryer, rice cooker
Lighting equipment	Household luminary, outdoor lighting, automotive lighting
Electrical and electronic tools	Volt–ohm–milliammeter, soldering iron
Toys, leisure and sports equipment	Coin slot machines, car racing set
Automatic dispensers	Water dispenser, coffee machine
Medical equipment	Ultrasound machine, heart–lung machine

[4] Baldé, C.P., Forti V., Gray, V., Kuehr, R., Stegmann,P. : The Global E-waste Monitor – 2017, United Nations University (UNU), International Telecommunication Union (ITU) & International Solid Waste Association (ISWA), Bonn/Geneva/Vienna.

[5] Wang, X. V., Ijomah, W., Wang, L., & Li, J. (2015). A Smart Cloud-Based System for the WEEE Recovery/Recycling. *Journal of Manufacturing Science and Engineering*, *137*(6), 061010.

While it is possible to repair and recondition certain e-waste products for use in secondary markets, they would eventually reach the end of their life and end in unprocessed waste streams (e.g. landfills or private storage in consumers' homes) or get recycled. It is challenging yet important to manage e-waste recycling due to its peculiar characteristics:

- E-waste collection requires special logistics, over and above the conventional waste collection infrastructure for municipal solid waste and wastewater streams.
- E-waste contains substances hazardous for human and environmental health. Therefore, its disposal and recycling requires careful treatment.
- Despite the hazardous substances, precious metals such as gold, silver and copper can be recovered from e-waste and put back into the material cycle. In fact, this is the primary incentive for existing recycling efforts, whether formal or informal.

In some countries, collection, recycling, disposal and monitoring of e-waste is an organized system, whereas in many others, it has evolved into informal sectors with questionable health and environmental impacts. At the moment, there is little incentive for a consumer to repair a device (before deciding to replace it by buying a brand-new one) or to dispose it responsibly.

Existing blockchain solutions for recycling. The use of blockchain for facilitating recycling is an emerging area, with few solutions implemented, which have been described briefly in this section.

The RecycleToCoin is a recycling initiative launched by the Blockchain Development Company (BCDC), which is a member of the Enterprise Ethereum Alliance[6]. Single use plastic bottles and aluminum cans can be exchanged at physical machines and designated collection points in the UK for BCDC tokens via a blockchain-based mobile app. The token owners can then privately finance renewable energy projects. Recycling is incentivized and the public is rewarded by allowing them to access investments in the sustainability domain. However, this approach may appeal only to those people who are interested to actively make sustainable investments, and not for everyone at large.

Every year, about 8 million metric tons of plastic finds its way into our oceans. In developing countries, it is estimated that about 80% of the plastic refuse comes from areas with high levels of poverty and lack of effective waste management systems[7]. The Plastic Bank is a Vancouver-based non-profit broker for recycling companies in developing countries that collects discarded plastic bottles, shreds the plastic and sells it back to manufacturers as an ethically-sourced raw material. Plastic Bank's philosophy is to make waste plastic too valuable to be discarded into the oceans. Many of the people collecting the waste plastic have no bank accounts and cash transactions may turn dangerous due to corruption and crime. However, almost everyone owns a mobile phone that supports digital transactions. A blockchain reward

[6] Whitepaper - Blockchain Development Company, https://www.bcdc.online/pdf/BCDC_WhitePaper.pdf
[7] *Plastic Bank deploys blockchain to reduce ocean plastic*, IBM IT Infrastructure Blog, https://www.ibm.com/blogs/systems/plastic-bank-deploys-blockchain-to-reduce-ocean-plastic/

system, co-developed by IBM and Cognition Foundry, enables people in the developing regions to safely earn Plastic Bank digital tokens by collecting and bringing plastic waste to recycling centers. The tokens can then be spent on vital goods such as food, schooling of children and phone top-ups. In this way, the Plastic Bank aims to address the dual objectives of keeping the oceans clean while lifting millions out of poverty globally at the same time.

Circularise is a Dutch startup that offers a blockchain platform to store product information to help certified recyclers identify components and materials in waste products that can be harvested for recycling or reuse[8]. Two advantages are claimed for this solution. First, permanence of information. Once stored on the blockchain, the information about a product and its material contents cannot be changed later. This can help validate the manufacturer's claims about toxins in their product at a later stage. Second, confidentiality of information. Circularise claims that their solution is able to disclose information selectively, i.e. only to those stakeholders in the value chain who need to know it via a "smart questioning" function. For example, a battery manufacturer can keep their proprietary ingredient information hidden from the smartphone manufacturer, but make it available to the certified recycler to enable appropriate processing of the battery. Circularise issues tokens for the products as its own revenue model.

Structure of this report. This report is divided into five chapters, including this Introduction chapter. Chapter 2 describes the blockchain-based solution that we propose to address the e-waste return and recycling challenge. Chapter 3 elaborates on the technical design of the the token generator solution and evaluates its features and bugs. Chapter 4 provides brief conclusions and outlook for continuing this work in the future. Chapter 5 directs the reader to the code repository for detailed instructions on using the code.

[8] https://www.circularise.com/

2. A Blockchain-based Solution

The current project aims for a solution to incentivize tracking of electronic devices and increase their recycling. This is attempted by implementing a token-based incentive system on top of a blockchain network, as described below and in Figure 1.

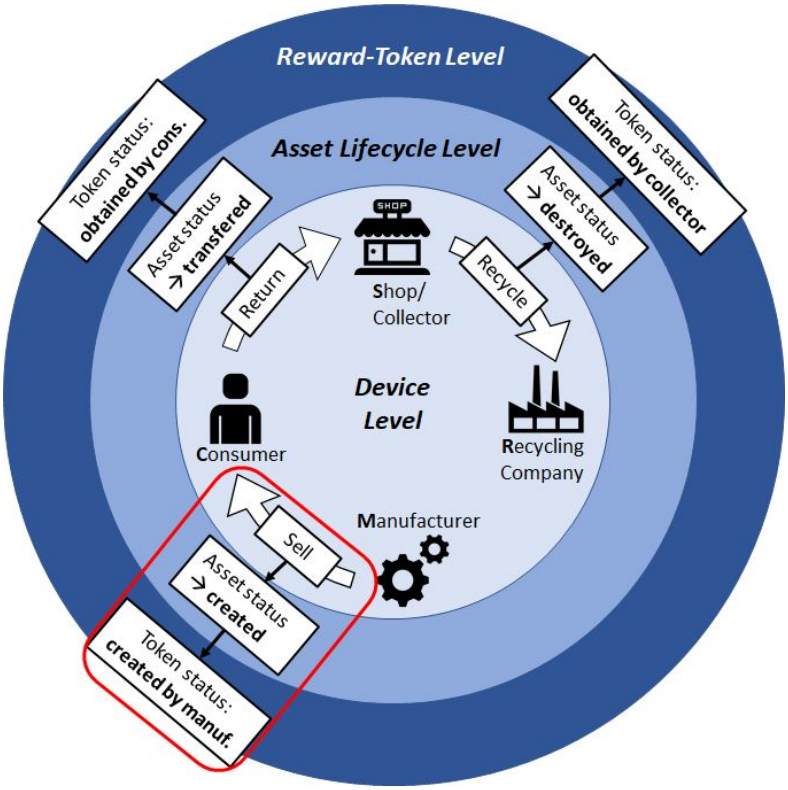

Figure 1. Schematic of the proposed blockchain solution for incentivizing e-waste return and recycling (focus of this project is highlighted in red)

To incentivize various stakeholders to track and recycle electronic devices, a token-based incentive system on top of a blockchain network that rewards tokens is implemented. First in line are manufacturers of electronic devices which, we believe, would be motivated to participate in the token-based incentive system, so that they can track the lifecycle history of their devices. The lifecycle of each device would be permanently stored on the blockchain and, as such, easily available to various environmental agencies, manufacturer company's shareholders, and other stakeholders interested in environmental impact of electronic devices. In exchange for this tracking information, the manufacturer can finance reward tokens. A full description of how our token-based solution incentivizes various stakeholders is as follows:

1. When a device is manufactured, an asset as well as a token is created.

2. This asset is tracked by the blockchain as it traverses through the system, from manufacturer to distributor to consumer to collector and ultimately to the recycler.
3. When the consumer returns the device to a collection center (or shop authorized to collect returned devices), a reward token is obtained by the consumer.
4. When the collector hands over the device to the recycler, another reward token is also granted to the shop.
5. When the recycler proves that the device was recycled, this info is transferred to the manufacturer.
6. In reality, there is no direct material link between the recyclers and the manufacturers because they are usually based in different continents, or the quality of recycled raw material may not be acceptable for the manufacturer.
7. So in effect, the current solution creates incentives in the system to track and return devices for recycling in the form of tokens and valuable lifecycle information.

Tokens. Tokens are commonly used to represent a particular asset or utility. They usually reside on top of another cryptocurrency blockchain, which in our case is Ethereum. Tokens can represent basically any assets that are fungible and tradable: commodities, loyalty points and even other cryptocurrencies. When creating tokens we do not have to modify the underlying code (logic) from a particular protocol/blockchain or create another blockchain from scratch as it is the case with alternative cryptocurrencies (also known as altcoins). All we have to do is to conform to a standard template on the existing blockchain – such as on the Ethereum platform – that allows us to create our own tokens. This token creation functionality is made possible through the use of *smart contracts.*

Smart contracts. A smart contract is, in general, a piece of code that executes on top of a blockchain network so that it changes its state. Smart contracts that manage tokens are called token contracts, and their primary function is to map account addresses to their token balances, which reside as state variables on top of a blockchain network. Balance is actually a value defined by contract creator (owner) and can represent different assets (physical objects, reputation, monetary value etc.). The unit of this balance is called a token.

Total supply of tokens can be:

- Increased - by *minting* new tokens (producing them for the first time)
- Decreased - by *burning* tokens

ERC-20 standard [9] came about as an attempt to provide a common set of features and interfaces for token contracts in Ethereum [10]. Using a smart contract standard such as ERC-20 has many benefits, for example allowing wallets to provide token balances for hundreds of different tokens and creating a means for exchanges to list more tokens by providing nothing more than the address of the token's contract. Also, using a recognized and time-tested contract standard such as ERC-20 ensures that essential security practices are followed. Currently most of the active token contracts on Ethereum blockchain are ERC-20 compliant tokens. An ERC-20 token contract is defined by the *contract's address* and the *total supply* of tokens available to it, but has a number of optional items that are usually provided as well to provide more detail to users. These are the *token's name*, its *symbol*, and the *number of decimals*.

3. E-Mine Token Generator web service

The problem. Creation of smart contracts is still a laborious task requiring specialized knowledge. Technically, it involves - 1) specifying a smart contract in Solidity or some other appropriate language, 2) compiling it into bytecode and 3) deploying it on the Ethereum network. Properties which smart contracts have to satisfy can be specified on any convenient user interface such as a mobile app or a web page. An actual smart contract can then be prepared by filling a predetermined template of a smart contract with the properties provided by the user. On the other hand, compiling and deployment of smart contracts is not readily available on all platforms. For example, currently there no Android based library for compiling the Solidity code [11], making it impossible to compile and deploy the smart contract directly through the Android mobile application such as GenesisApp [12].

Our solution. We decided to build E-Mine Token Generator [13] web service through which users can submit requests for creation and deployment of their smart contracts. We built a web service using Flask [14]- a widely used Python [15] based web framework. The web service can be accessed through a REST API which allows user to programatically make a request for creation and deployment of a smart contract. Properties of desired smart contract are specified by setting the relevant parameters exposed through the REST API. The web service then prepares an actual smart contract by filling a template of Solidity code, compiles it and broadcasts it on Ethereum network. e-mine-web provides compilation and deployment of several standard tokens which are available in OpenZeppelin framework [16] (Table 2.). For compilation and

---

[9] https://theethereum.wiki/w/index.php/ERC20_Token_Standard
[10] https://medium.com/@jgm.orinoco/understanding-erc-20-token-contracts-a809a7310aa5
[11] As of February 2018.
[12] GenesisApp, https://github.com/FuturICT2/GenesisApp, https://github.com/FuturICT2/Genesis
[13] e-mine-web , https://github.com/e-mine1/e-mine-web
[14] Flask, http://flask.pocoo.org/
[15] Python, https://www.python.org/
[16] OpenZeppelin, https://openzeppelin.org/

deployment on the server side we used Truffle [17] - a Node.js [18] based development framework for Ethereum smart contracts. In order to test our system we set up a local Ethereum network using Ganache [19], a tool which is a part of the Truffle framework. Ganache features an internal Javascript implementation of Ethereum blockchain, as well as built-in blockchain explorer that allows easy testing and debugging of distributed applications. We extended GenesisApp - an Android mobile application for creation of smart contracts, so that it can send and receive requests to and from E-Mine Token Generator web service. In this way it effectively serves as an user interface for creation and deployment of smart contracts on the Ethereum blockchain, although in principle any other REST capable user interface would do. The overall system architecture of the E-Mine Token Generator is displayed on Figure 2.

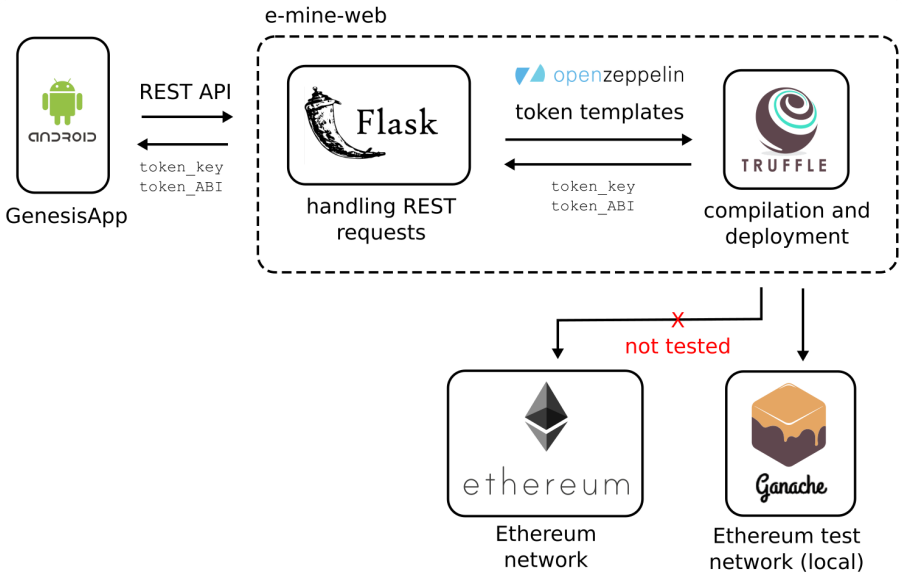

Figure 2. Architecture of the E-Mine Token Generator web service for compiling and deploying the smart contracts to Ethereum network.

Security concerns. We should emphasise that our solution is currently in a prototype phase and should not be used in a production environment. Largest security concern, which we do not address at all, is the management of public and private keys. Currently, the client sends its private key directly to web service and the service is responsible for compilation and deployment in the name of the client. In addition, the communication between the user and the web service is not secure. This exposes clients to the potential fraud by the web service provider, as well as by any malevolent eavesdropper in the communication channel. Key management is the most crucial functionality which should be implemented before the system could be considered production-ready. For more details, see section Future work.

[17] Truffle, http://truffleframework.com/
[18] Node.js, https://nodejs.org/en/
[19] Ganache, http://truffleframework.com/ganache/

Smart contract templates. As a basis for token creation we use OpenZeppelin, a framework to build secure smart contracts on Ethereum. We provide templats for many standard token types, implemented as an extended class of corresponding OpenZeppelin token implementation. Following token templates are provided:

Table 2. Token templates available on our web service. All token templates have the following properties exposed: name, symbol, decimals, initial supply.

Token Standard	Token	Description
ERC20	Basic Token	Implementation of the basic standard token
	Burnable Token	Standard token with no allowances
	Capped Token	Standard token modified with pausable transfers
	Mintable Token	A Simple ERC20 with mintable token creation
	Pausable Token	Mintable token with a token cap
	Standard Token	Token that can be irreversibly burned (destroyed)
ERC721	Generic ERC721 Token	Generic implementation for the required functionality of the ERC721 standard
ERC827	Generic ERC827 Token	ERC827 implementation - ERC20 standard with extra methods to transfer value and data and execute calls in transfers and approvals

Each template contains placeholders for custom user parameters provided by the client application. Currently, the following parameters are supported:

- *Token name* - Name of a token.
- *Token symbol* - Symbol representing the token.
- *Decimals* - Divisibility of a token, i.e. number of decimal places when displaying token value.
- *Initial supply* - Initial supply of tokens.

Each template can be easily extended to introduce new smart contract (token) parameters and functions according to the specific user requirements.

Prior to integrating manually written templates with the rest of the E-Mine Token Generator system, we tested token creation (for all token types supported by our generator) using the

Remix Solidity IDE[20]. We made sure that newly introduced parameters hold and return correct values after successful creation.

As a final step, we remove hardcoded values, and introduce the corresponding placeholders that will be updated with real values provided by the Token Generator Web service during the creation of a new token:

```
string public constant name = "%token_name%";
string public constant symbol = "%token_symbol%";
uint8 public constant decimals = %token_decimals%;
uint256 public constant INITIAL_SUPPLY = %token_initial_supply%;
```

We then used the same approach to create templates for all other token types listed in Table 2.

REST API. E-Mine Token Generator exposes a REST API to compile and deploy token contracts. As of initial writing of the source code, we did not find an official way to compile Solidity source code on the Android platform. Some workarounds and solution ideas are further discussed in the Future work section. Within the time scope of the hackathon, the decision was made to delegate compilation and deployment to a backend service. The backend service exposes the following web interface to a client.

Table 3. REST API endpoints for the E-Mine Token Generator web service.

#	API URL	Method	Required/Optional variables	Expected response
1	/api/requests/<key>	GET	key=[integer]	same response as in /api/tokens/create
2	/api/tokens/types	GET		{ "types": ["MyBasicToken", "MyBurnableToken", "MyCappedToken", "MyERC721Token", "MyERC827Token", "MyMintableToken", "MyPausableToken", "MyStandardToken"] }
3	/api/tokens/create	POST	tokenName=[string] symbol=[string] maxSupply=[integer] decimals=[integer] genesisSupply=[integer]	{ "created": [datetime], "key": [uuid], "status": "success" or "pending" or "failure", "token_abi": [base64 encoded JSON value]

[20] https://remix.ethereum.org/, Remix is an IDE for the smart contract programming language Solidity and has an integrated debugger and testing environment.

				"token_addr": "0xaba7902442c5739c6f0c182691d48d63d06a212e", "updated": [datetime], "version": 1}

With the endpoint `/api/tokens/create` (3) a client can form a POST request to compile and deploy a token contract. This process may take a while which is why a call to (3) immediately returns with a internal key. This key can be used as a request parameter in a GET call to `/api/requests/<key>` (1) to obtain status results. Opon deployment success, the response of (1) contains the smart contract address (referred as token_addr) and the application binary interface (token_abi) of the deployed contract. With these pieces of information, a client is able to interact with the smart contract on the Ethereum network.

```
{
       "created": [datetime],
       "key": [uuid],
       "status": "success" or "pending" or "failure",
       "token_abi": [base64 encoded JSON value ]
       "token_addr": "0xaba7902442c5739c6f0c182691d48d63d06a212e",
       "updated": [datetime],
       "version": 1
}
```

Figure 3. Response of POST /api/tokens/create and GET /api/requests/<key>. Once status is success, the token_abi and token_addr are available

As described in section "Smart contract templates", various standardized contracts can be deployed. A list of supported templates can be obtained with a GET request to the endpoint `/api/tokens/types` (2).

The REST API follows the HTTP status code convention that codes for a successful request are always code 200. If a status code is not 200, the JSON response contains a field "error" with an error message.

The most common return codes are the following:

Table 4. Common HTTP Return codes

Code	Reason
200	Successful request
400	Missing/ Invalid required fields in request.
500	Server error, POST requests without valid JSON body

Source Files Backend. The essential parts of the backend are implemented within three python source files.

Table 5. Source files of the backend

#	Name	Responsibility
1	`./emine_web.py`	`REST Backend`
2	`./solc_compile_deploy.py`	`Truffle compile & deploy`
3	`./db.py`	`Request database`

`./emine_web.py` (1) implements REST Endpoints (see REST-API section) and uses the Flask framework.

Incoming requests are validated and scheduled for compilation using `./solc_compile_deploy.py` (2). A new Solidity source file is created with respect to a selected smart token template and the the command-line interface of Truffle is launched. `./solc_compile_deploy.py` (2) spawns a new process and instructs Truffle to compile the Soldity source file and to deploy it into a test network. `db.py` (3) runs a local instance of a database (TinyDB[21]). Since complication and deployment may take up a while, `db.py` (3) stores an internal request key with which status information can be obtained.

Mobile Client. To demonstrate the effectiveness of our solution we extended the GenesisApp so that it can communicate through the REST API with our web service. The app implements a simple graphical user interface through which users can select one of the supported smart contract templates. For demonstration purposes, we extended the given Genesis app by an additional Android Activity. The activity delegates requests to HTTP wrappers built around okhttp[22], a An HTTP+HTTP/2 client for Android applications. For each REST endpoint implemented in the backend, a corresponding client wrapper exists. Upon selection of all required fields in the smart contract template, the Android Activity delegates the user input via okhttp to Emine Token Generator backend. The compilation and deployment process is visualized with a spinning circle animation.

[21] TinyDB https://tinydb.readthedocs.io/en/latest/
[22] https://github.com/square/okhttp

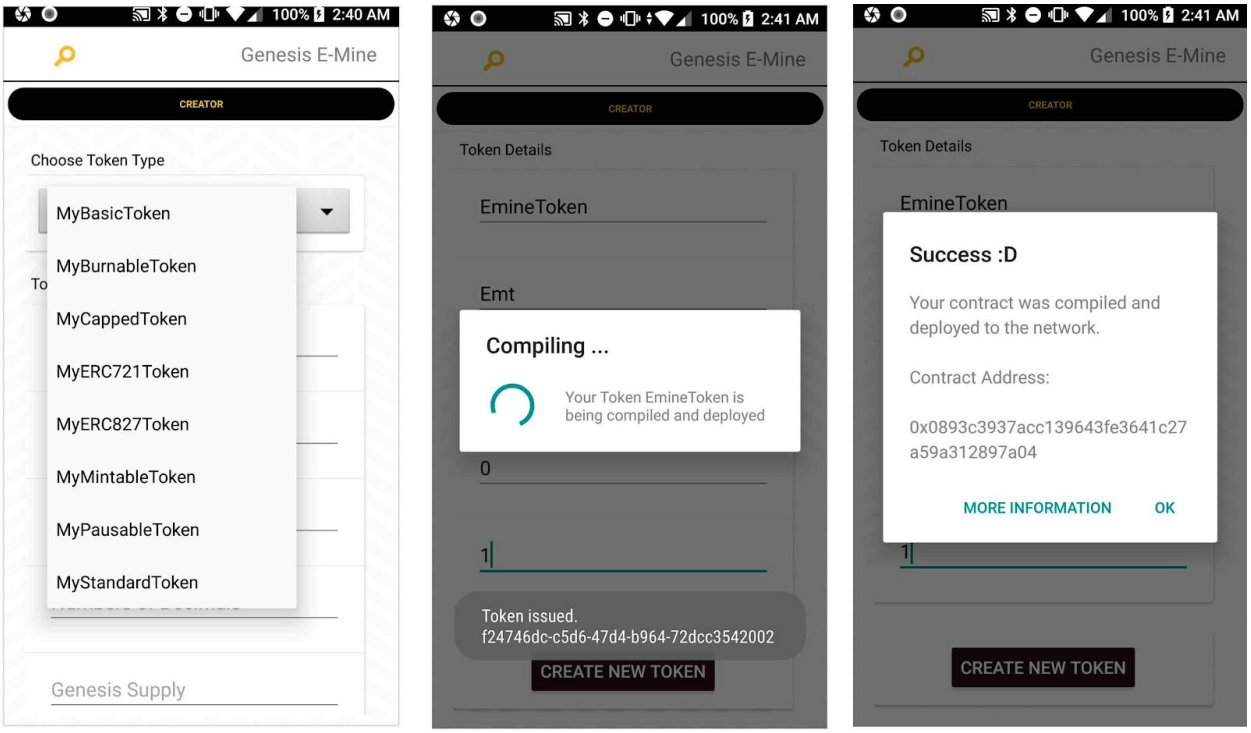

Figure 4. The GenesisApp app implements a user interface through which users can submit requests to compile and deploy smart contracts. A user can fill in the smart contract template with additional attributes (see smart contract templates) and can issue a request to the backend web service. The requests are sent through the REST API to the web service which compiles and deploys smart contracts.

The mobile app calls the REST API `/api/tokens/create` with filled in details (left image) and uses the obtained internal key to call `/api/requests/<key>` periodically to obtain status information (middle image). The key is a v4 UUID (Universally unique identifier) and is generated and stored on the backend. Upon compilation and deployment success, the ABI and contract address can be viewed (right image).

With the ABI, Application Binary Interface, and the address of the contract, all major pieces are at hand to interact with the contract. A library such as web3j[23] can then connect to the contract and interact with it. Web3j is a lightweight Java and Android library for integration with Ethereum clients. It's the equivalent of web3.js but written for the Java Platform. Integration of web3j was not implemented within the time scope of the Hackathon (see future work section).

4. Conclusions and Outlook

Conclusions. In the time scope of the hackathon we built a working prototype to compile and deploy smart tokens on a Android client to incentivize people to collect and recycle e-waste. The

[23]web3j: https://github.com/web3j/web3j

prototype consists of two tiers, a Android based user interface and E-Mine Token Generator backend which serves for compilation and deployment of smart contracts.

Outlook. There are several directions for future work and potential upgrades of the E-Mine Token Generator.

Compilation and deployment of smart contracts on client side. When interacting with smart contracts, we faced two challenges: (i) compilation of smart contracts and (ii) deployment into the ethereum blockchain. One way to interact with the blockchain is to download the whole blockchain and keep a local node synchronized. As of writing of this report, a ethereum blockchain node with geth client requires more than 50GB of storage[24]. This size is not reasonable for smartphones which is why we used a service such as Infura[25] to deploy contracts. Infura runs an entire ethereum node and provides services to remote connect and to execute transactions without to worry about maintenance and synchronization. Infura does not sign transactions and therefore does not impose a security risk. The only concern with Infura is availability. Since it is broadly used by the MetaMask we believe that availability is a minor concern.

With Infura and a framework such as web3j, a compiled smart contract can be deployed into the blockchain. Web3j is available on Android[26], however, in order to compile a smart contract, a Solidity compiler is needed. The official Solidity compiler - solc - has not yet been ported to Android. Porting a compiler to Android was not possible in the scope of the hackathon which is why we built E-Mine Token Generator backend. The source code of solc and all libraries used by solc need to be recompiled for Android (ARM target) and a Android NDK[27] wrapper library needs to be built to delegate compilation requests to solc.

An alternative way to compile Solidity code is to use the web-assembly port of the solc compiler: solc-js[28]. Solc-js can possibly be executed in a Webview on Android or within an alternative Javascript engine. So an alternative to the approach to port the solc compiler is to investigate the use solc-js on Android. This was, however, not further considered in the hackathon.

Due to the lack of a Solidity compiler, our mobile client sends a request to the backend to compile and deploy a new smart contract. The client currently does not transmit private keys and those being used are hard coded in the backend.

Key management. With a way to compile Solidity code on Android, all major pieces are available to compile and deploy smart contracts securely on Android. In order to handle keys properly, a wallet integration should be considered. For a smart contract to be deployed, a user can import a key from his wallet into the E-Mine Android app. For this to work, the user's wallet

---

[24] geth, ethereum client in go, sync mode FAST https://etherscan.io/chart2/chaindatasizefast
[25] https://infura.io/
[26] https://github.com/web3j/web3j
[27] Native Development Kit
[28] https://github.com/ethereum/solc-js

needs to support a key-share/export feature. We don't intend to reimplement a wallet as there are already sophisticated wallet apps for Android available. If a way to compile Solidity code has been ported to Android, the imported key stays local on the phone and is used to sign transactions for deployment of smart contracts. If no such way continues to exist, the key is sent to E-Mine backend where it is used for deployment. Currently, no such key exchange is performed and statically defined keys are used in E-Mine backend. We suggest that future work of this project focuses on a solc port so that no keys must be exchanged. Even if E-Mine Token Generator backend is released as open source code and every entity interested in this project can host their own backend, private keys are sensitive data and an exchange of keys increases the sum of different points where an unauthorized user can try to extract data. We suggest that private keys stay on the client device where they are used to sign transactions.

Deployment on Ropsten test network. E-Mine Token Generator currently deploys contracts into a local network generated by Ganache. Further extensions of this project cover deployments into other networks such as the Ethereum blockchain or Ropsten test network. Truffle is used for deployment and deployment targets are defined by configuration files in E-Mine Token Generator backend. For deployment into other networks with the currently implemented approach one needs to extend Truffle configuration files.

Web-based user interface. Many challenges with Solidity compilation can be avoided if a frontend with web technologies is built. If the Android functionality is ported to a website, Javascript can be used to compile contracts. Similar to Remix Solidity IDE, solc-js wrappers can compile Solidity code. For desktop browsers, MetaMask can store keys and be used to sign and submit transactions. A web3[29] Javascript app can be built without the need of functionality currently implemented in E-Mine Token Generator backend.

5. Installation instructions

To run the backend, Python3+, pip3, nodejs, npm, Truffle and Ganache are required. Python package dependencies are available in the ./requirements.txt file in the source directory of the project. The Truffle binary search path may be changed using the constant TRUFFLE_BIN in solc_compile_deploy.py.

Further installation instructions can be found on the Github websites of the project:
https://github.com/e-mine1/e-mine-web (backend),
https://github.com/e-mine1/GenesisApp (frontend)

[29] web3: https://github.com/ethereum/web3.js/, javascript API for Ethereum