Analysis of uPort Open, an identity management blockchain-based solution

Andreea-Elena Panait[1][0000−0003−0591−3135], Ruxandra F. Olimid[1,2,3][0000−0003−3563−9851], and Alin Stefanescu[1,2][0000−0002−8418−2643]

[1] Department of Computer Science, University of Bucharest, Romania
[2] The Research Institute of the University of Bucharest (ICUB), Romania
[3] Department of Information Security and Communication Technology, NTNU - Norwegian University of Science and Technology, Trondheim, Norway
andreea-elena.panait@drd.unibuc.ro, {ruxandra.olimid,alin}@fmi.unibuc.ro

Abstract. Recently, academics and professionals proposed a substantial number of identity management solutions based on blockchain technology. Among them, uPort appeared promising, being considered for both personal and administrative operations. This paper analyzes the open-source version uPort Open in terms of risk delegation and private key recovery of the mobile application, smart contract security of the uPort identity registry, and usage and on-chain transaction analytics.

Keywords: identity management, blockchain, self-sovereign, uPort

1 Introduction

Technological development in the last years led to an increased number of digital entities that must be identified in an interoperable and efficient way. With the adaptation of 5G and mostly Internet of Things (IoT), identity and access management became a difficult issue to address. Papers like [10] propose ways to leverage blockchain for solving such problems. A recent identity management model, called the *Self-Sovereign Identity (SSI) model* makes the user be his/her own identity provider and fully responsible for his/her identity, thus eliminating the need of an external identity provider [13]. The blockchain technology has been considered to be a candidate solution to accommodate identity management while decreasing risks such as identity fraud or data leaks [13]. Among the identity-management solutions built on blockchain, uPort is currently one of the most known, targeting both personal and administrative operations [21]. uPort comes in two versions, one for public networks, called uPort Open, which is open-source, and one for closed ecosystems and consortia, called uPort Serto.

Contribution. Motivated to find the current status of open-source identity-management solutions, the paper analyses the security and usage of uPort Open, as a representative open-source solution. For simplicity, unless otherwise specified, we will further refer to uPort Open by simply uPort. We conduct our analysis in three directions: (1) a brief assessment concerning risk delegation

and private key recovery, (2) an automatic code analysis of the identity registry smart contract, and (3) uPort usage and on-chain transaction analytics. By looking into terms, conditions, and privacy policies, we found that for the mobile application, the risks are delegated to the user, with (almost) no responsibility on the uPort side. Concerning the code analysis, the used tools found possible vulnerabilities, which we have notified to the uPort team. Later in the paper, we will refer to the uPort team response to our notice. Their feedback assisted us in understanding better the purpose of their proposed solution[4]. Finally, we developed an application to collect and analyze the blockchain transactions made to the uPort identity registry, *EthereumDIDRegistry*. We found that the on-chain usage of the uPort identity is rather low, the identities did not make significantly usage of the offered features (i.e., changing owner, setting attributes) through its *EthereumDIDRegistry*. Moreover, advanced features such as delegation (i.e., granting the right to perform identity-related actions for a certain period to other addresses) are even more rarely used. However, this on-chain usage is somehow expected since the uPort identities are intended to be mainly used without any call to the *EthereumDIDRegistry*. In fact, changes in the owner or attributes and delegation are only possible using libraries, and not via the mobile application.

Related Work. Except uPort, several other identity management solutions have been proposed so far. More comprehensive lists are available at [8, 13]. From those, we chose to investigate uPort because it is one of the most visible open-source solution[5] both in the academic literature and in the online setting. uPort is theoretically described and briefly analysed in several publications, including [2, 4, 13, 14]. Nevertheless, they mostly analyze uPort only on general aspects. To the best of our knowledge, no similar results in terms of smart contract security and on-chain usage and transaction analytics exist.

2 Background

Identity Management on Blockchain. A *blockchain* is a type of decentralized distributed database with cryptographic enhancements. It is *distributed* because the data is stored on multiple *nodes* (each node stores a complete or partial copy of the blockchain), and it is *descentralized* because the storage decision is a result of a *consensus* protocol between the nodes [13]. Data is stored in *blocks*, which are basically collections of *transactions*. A user authenticates a transaction by signing it with the private key. The paired public key represents the user's address in the blockchain. A blockchain-based identity management solution implements selective storing of identities in the blockchain, where the identities are *attested* by authorities or other entities in the blockchain, usually by *verifiable claims* [13]. A *self-sovereign* solution eliminates the need for an external authority by transferring the full responsibility for creating and managing the digital identity and associated attributes to the user.

[4] We have notified the uPort team to their official address, *info@uport.me*.
[5] Maybe together with Sovrin [15].

Ethereum is a particular implementation of a blockchain, which is capable to store *smart contracts*. A smart contract is identified and referred to by an address (a public key) and provides a function-based interface that allows it to be executed. To execute a smart contract (changing its state), a user performs a transaction with the function signature and the input parameters as the data payload and the smart contract's address as the destination [19]. We differentiate between *direct* transactions, which directly call the smart contract and *indirect* transactions, which result from an execution of a smart contract that executes another stand-alone transaction[6].

uPort Open. uPort Open is an open-source identity management system built on Ethereum that claims to satisfy the self-sovereign property [21]. It has three main components: (1) the mobile application, which stores the user's public-private keys pair, and is used to perform different actions (e.g., create and manage the digital identity, authenticate to third parties, sign transactions), (2) the Ethereum smart contracts, and (3) protocols for signed messages and message flows, as well as libraries and interfaces necessary for integrating uPort with third-party applications [19, 20]. The uPort identity has a Decentralized Identifier (DID) and a pair of public-private keys [20]. The management of the DIDs is done by the *EthereumDIDRegistry* smart contract that allows the *owner* of an identity to update the customized *attributes*. The *owner* is, in fact, an Ethereum address that has full control over the identity but accepts *delegates*. A delegate is an address that is delegated for some specific time to perform actions on behalf of an identity [20]. The *EthereumDIDRegistry* smart contract is deployed both on mainnet Ethereum blockchains (Mainnet, RSK, and Alastria Telsius) and on testnets (Ropsten, Rinkeby, Kovan, RSK Testnet, and Goerli) [20]. It is written in the Solidity programming language and provides several functions to manage identities: change identity ownership, set or revoke attributes, set or revoke delegates [20]. Some of these functions emit events that change the state of the contract. The event arguments are stored in a blockchain special data structure called *transaction log* that can be listen to through the Remote Procedure Call (RPC) interface of an Ethereum client. The *EthereumDIDRegistry* functions related to identity ownership emit the *DIDOwnerChanged* event, the functions related to identity attributes emit the *DIDAttributeChanged* event, and ones related to delegates emit the *DIDDelegateChanged* [20].

Analysis Tools. To check for vulnerabilities in the uPort registry smart contract *EthereumDIDRegistry* we looked into the static and dynamic tools listed in [3]. To analyze uPort transactions, we used *Etherscan.io*, which is a stand-alone web-based Ethereum blockchain explorer, but also provides access to the data stored in the blockchain by APIs. The APIs are provided as a community service and require the creation of a free *API-Key Token* [5]. We developed an application to extract data and store it in a SQL database. For this, we used *Swagger*, an open-source editor dedicated to OpenAPI Specification (OAS)-based APIs that help developers to make use of RESTful web services [16].

[6] For clarity, we avoid to refer to *normal* and *internal* transaction.

3 Results and Discussion

We further present and discuss our results. Since uPort is a solution under continuous development, note that our analysis is conducted based on the source code and transactions prior to Dec. 13th, 2019.

3.1 Risk Responsibility

The uPort mobile application is a credential wallet that does not manage all aspects of an decentralized identity, features that are included in the uPort libraries (e.g., *eth-did-registry* and *ethr-did* that uses [6]). The application can be used, for example, to register in decentralized applications.

We first look into how the uPort mobile application manages risks responsibility and private key recovery. For this, we have installed and inspected the uPort application that is freely available in Google Play and tested the application ourselves. To create an account, the user must accept the *Terms and Conditions* and the *Privacy Policy*, which delegate strong responsibility to the user and assume usage on the user's own risk. The agreements refer to risks in terms of cryptography and platform security, and also absolve uPort from bugs, errors, and defects by delegating the responsibility to the user. Although this might be seen as natural for an open-source solution, the users must fully understand the responsibility and risks in using the solution.

The *Terms and Conditions* specify that generation of the private key is possible from a potential leaked twelve-word seed and the impossibility of recovery of the private key or the twelve-word seed by other means but by user knowledge. After installation, uPort displays two options: *Get Started* and *Recover Identity*. While creating the account (*Get Started*), the user sets a twelve-word Recovery Seed Phrase that can be used for recovering the account: the words are selected from a list that becomes available after inputting two letters from a word. This restricts the available words to be part of a dictionary - BIP39 standard with 2048 English words [11]. The number of possible combinations is $2048^{12} = 2^{132}$, so 132 bits of security, which can be considered rather secure. The twelve-words passphrase is a widely used practice in applications such as crypto wallets. We note that while selecting the twelve words, the *Next* button remains inactivated. It becomes active only for a valid twelve-word phrase.

3.2 Code Analysis

We further present our findings after analyzing the *EthereumDIDRegistry* smart contract with several tools from [3]. The used tools and the found vulnerabilities are summarized in Table 1, together with suggested tool mitigation techniques and our notes. We looked into other tools from the above-mentioned list, but we were not able to use them due to various reasons.

The uPort Response. Regarding the upgrade to a 0.5.x version of Solidity, uPort admitted that the next contract deployment will be upgraded to a recent

Table 1. Code analysis: vulnerabilities exposed by different tools

SmartCheck [17]
(V1) Incorrect compiler version *Mitigation*: Specify the exact compiler version used for test *Note*: Not applicable for a consumption contract, such as the one analyzed [3]
(V2) Old Solidity compiler version *Mitigation*: Upgrade to Solidity 0.5.x
(V3) Function *keccak256(arg)* will become deprecated *Mitigation*: Change the function call to *keccak256(abi.encodePacked(arg))*
(V4) Data location for function parameters is not explicit *Mitigation*: Specify data location *Note*: This vulnerability is applicable after smart contract update to Solidity 0.5.x
Securify [18]
(V1) Recursive calls after a method call that is followed by a state change *Note*: Not accurate, state changes are made after calling the predefined Solidity function *ecrecover* by the identity owner, otherwise no state change occurs
(V2) Insecure coding patterns when unrestricted write to storage *Note*: Writing is restricted to identity owners only
(V3) Unexpected Ether flow (lock of Ether) *Note*: The smart contract does not receive or deposit any Ether
MythX [9]
(V1) Possible overflow at the binary addition of current time with a *uint* validity *Mitigation*: Use an assertion to catch the overflow
(V2) *Block number* considered a weak source of randomness *Note*: Tool warning. The block number is used for logging triggered events, not as a source of randomness
(V3) Potential Denial-of-Service to block gas limit when using *keccak256* function *Note*: Tool warning
ContractGuard [23]
(V1) *Block number* dependency *Mitigation*: Not recommended to use, it can be manipulated by attackers
(V2) Timestamp dependency *Mitigation*: Not recommended to use the current time *Note*: Triggered by an event having a date that is an addition between the current time and a validity parameter of type *uint*
(V3) Using *require* assert without reason string attached *Mitigation*: Suggested to add a reason string *Note*: Tool warning
(V4) Misplaced order of smart contract functions *Mitigation*: Recommended order: constructor, fallback, external, external const, public, public, const, internal, and private functions *Note*: Tool warning

Solidity version, but even though the *keccak256* function is deprecated in the recent versions, the function behaves as expected in the previous deployments. For the possible overflow of the binary addition of the current time with a validity of type *uint* (within the *addDelegate* and *setAttribute* functions) for future deployments, one can indeed use the assert statement. A possible overflow will only result in adding a delegate or an attribute that is already expired, but no other off-chain or on-chain changes will occur. Therefore, this would not be a problem from the uPort team's point of view. For the possible DoS attack that might occur because the first argument of the *keccak256* hash function might grow unbounded, the worst-case scenario would be that the transaction fails.

3.3 Advanced Usage and Transactions Analytics

Finally, we investigate the usage of uPort solution. We start by finding the number of mobile application downloads but then proceed to a more in-depth analysis concerning on-chain performed transactions.

Downloads. The first point in our numerical analysis addresses the number of downloads of the uPort mobile application. Although this cannot be a precise indication for the number of users (as for example the same user can download the application on several devices, or individuals can download the application but never create a digital identity), we consider this to be an acceptable indication of the interest towards the uPort mobile solutions. At the time of the writing, in Google Play there were 10,000+ downloads and only seven reviews [7]. This seems to indicate a low usage of the application, with many users that downloaded the application just by curiosity. Similarly, within the AppStore there were only three reviews, indicating the same low usage [1].

Transactions. Secondly, we focused our analytics concerning the analyzed smart contract's transactions on different networks. The address of the *EthereumDIDRegistry* smart contract in different blockchains can be found at [20]. At this step, we have used these addresses and found the number of transactions directly from the web interface. Table 2 shows the number of direct transactions made to the uPort smart contract for each blockchain network where it was deployed, together with other details. Note that some of these transactions can be with error (all transactions are included in the web interface), and the number of transactions always includes one transaction used for the contract creation. We deliberately exclude the Alastria Telsius testnet from our analysis, as it presents errors and does not correctly display the information extracted.

Transaction types and on-chain digital identities. We further performed a more in-depth analysis concerning the networks with a significant number of transactions (Mainnet, Rinkeby, and Ropsten). This time, we extracted the transactions using the *Ethereum Developer APIs* [5]. To fulfill our goal, we developed a .NET Core simple web application for making the API requests through a Swagger frontend interface [16] and saved the transactions in a SQL Server Database. The sample project can be found at [12]. In order to find the number of uPort created identities that changed their owner, set identity

Table 2. *EthereumDIDRegistry* direct transactions (from the web-platform)

Blockchain Network	Blockchain Type	Contract Creation	First Transaction	Last Transaction	Direct Trans.
Mainnet	Mainnet	Jun-15-2018 01:27:27 AM	Jan-11-2019 07:56:32 PM	Nov-09-2019 05:56:40 PM	2079
Rinkeby	Testnet	Jun-15-2018 01:05:09 AM	Aug-02-2018 01:28:52 PM	Dec-06-2019 11:24:32 AM	4106
Ropsten	Testnet	Jun-15-2018 01:07:19 AM	Oct-01-2018 08:45:27 PM	Nov-22-2019 02:41:49 AM	681
Kovan	Testnet	Jun-15-2018 01:01:44 AM	Dec-04-2019 02:27:44 PM	Dec-04-2019 02:27:44 PM	2
RSK	Mainnet	Jun-06-2019 04:41:33 AM	-	-	0
RSK Testnet	Testnet	Jul-24-2019 05:24:34 AM	-	-	0
Görli	Testnet	Dec-02-2019 01:57:27 PM	-	-	0

Table 3. Types of transactions and number of unique addresses grouped by network

Blockchain	Successful Trans.	Direct Trans. with Error	Direct Successful Trans.	Indirect Successful Trans.	Unique Addresses	Unique Addr. with > 1 trans.
Mainnet	2844	68	2010	834	2004	431
Rinkeby	7570	89	4016	3554	3805	3570
Ropsten	634	46	634	0	434	19

attributes and/or revoked identity delegates, we used the *getLogs* API method. This is an Etherscan API method that provides the event type triggered when a specific uPort registry function is executed (for the event types, see Section 2). To exemplify, we give next a Rinkeby request for transactions belonging to blocks in a given interval (fromBlock - toBlock), where the address could correspond to the *EthereumDIDRegistry* smart contract and `YOUR-API-KEY` is the Api-Key Token: `https://api-rinkeby.etherscan.io/api?module=logs&action=getLogs&fromBlock=2463641&toBlock=3224895&address=YOUR-ADDRESS&apikey=YOUR-API-KEY`.

Table 3 shows the number of successful transactions, the number of error transactions, the number of direct and indirect successful transactions, the number of unique addresses that made transactions, and the number of addresses that made multiple transactions. Note that this time, the number of transactions excludes one successful transaction, which is the contract creation transaction. In Mainnet, the number of indirect transactions represents 26.93% from the total number of successful transactions, while for Rinkeby it represents 45.78%. This suggests that, on tests, the developers are more likely to send transactions to intermediar smart contracts before sending transactions to the uPort registry. In Mainnet, only 21.50% of the unique addresses made more than 1 transaction,

Table 4. Transactions and unique addresses grouped by event type and year

Blockchain	Year	uPort Event	No. Trans	Unique Addresses.	Unique Addr. with > 1 trans.
Mainnet	2019	Owner	2002	2001	1
		Attribute	839	431	381
		Delegate	3	2	1
Rinkeby	2018	Owner	7105	3573	3525
		Attribute	137	71	21
		Delegate	70	10	5
	2019	Owner	2	2	0
		Attribute	225	170	18
		Delegate	31	8	4
Ropsten	2018	Owner	93	93	0
		Attribute	103	85	12
		Delegate	15	7	1
	2019	Owner	12	5	1
		Attribute	276	250	4
		Delegate	135	1	1

for Rinkeby network 93.82%, whereas for Ropsten only 4.37%. The number of unique addresses is important as it reveals the number of identities in the network that made changes to their initial values, while the number of addresses with more than one transaction implies that some identities made several changes to their initial identity. The transactions with errors were not returned by the API requests because for an error transaction the events are not triggered. Hence, for each network of interest, we computed their number by the difference between the number of transactions in Table 2 and the number of direct transactions returned by API request. A high percentage of errors usually indicate a problem or an immature solution. We found a percentage of 2,39% errors in Mainnet. Overall, the numbers indicate that on-chain, the solution is mostly used for performing tests (Rinkeby), with no significant usage in real applications.

Digital identity-related events. Table 4 splits the transactions on year and event type. For Mainnet, there are no transactions in 2018. The data shows that only one identity changed the owner more than once. 88.4% of the addresses triggered more than a single attribute event. This appears normal, as an identity should naturally set and change attributes corresponding to its identity. The delegation was mostly unused: with only 3 *DIDDelegateChanged* events, it is clear that the this was found to be an uninteresting feature. With respect to the test networks, we notice a significant number of creating or changing owners in 2018. This is natural for testing purposes. Moreover, we observed intense testing in owner change on Rinkeby (98.65% of the identities made at least a change in the owner). Concerning the total number of unique addresses per blokchain network, from Table 3 and Table 4, as well as from querying the database, it results that: for Mainnet 429 addresses made owner-related and

attribute-related transactions, and 1 address made owner-related and delegate-related transactions; for Rinkeby 18 addresses made owner and attribute-related transactions, 6 addresses made owner and delegate-related transactions, and 5 addresses made attribute and delegate-related transactions. From this, 1 address made all three types of transactions; for Ropsten there were 7 addresses that made transactions related to owner and attributes.

Discussion. The measure of downloads and on-chain interaction with the uPort *EthereumDIDRegistry* does not entirely reflect the usage of this solution (uPort self-sovereign libraries), and we can not make a correlation between them, as initially thought: the uPort mobile application is a credential wallet and does not interact on-chain with the registry, whereas the on-chain interaction does not reflect the entire uPort usage. Identities can act themselves as public keys by using the *eth-did-resolver* library, which enables the Ethereum addresses to be self-managed Decentralized Identifiers [22]. This means that they need not to previous register to produce valid signatures (and prove their identity). Moreover, as the uPort team mentioned in their feedback, the identities are more likely not to use the changing owner, setting attributes and delegate features due to reasons of scalability, cost and in order to enhance their privacy. Lastly, uPort mentioned that the attribute and delegate changes are expected to be performed mostly by organizations that require multiple entities to have the power of signature without sharing the private keys with each entity. The ownership changes are to be performed even rarely in extreme events or when upgrading the identity model from a key pair to a multi-signature contract, the team specified.

4 Conclusions

Being quite visible online and in the academic world, we were interested in researching whether uPort Open is popular among users. Our analysis concluded that the on-chain usage of the uPort identities is not significant, but the solution is expected to be more used for off-chain interaction and valid signature signing. However, off-chain usage is difficult to be measured due to the privacy-by-design nature of the solution. The possible vulnerabilities encountered while analyzing the security of the *EthereumDIDRegistry* smart contract were reported to the uPort team, which responded to all our points. The uPort mobile application is a credential wallet and does not make use of advanced identity features. Finally, when using the mobile application, responsibility is fully delegated to the user.

Acknowledgement. This work was partially supported by a grant of Romanian Ministry of Research and Innovation project no. 17PCCDI/2018.

We thank Mircea Nistor, from the uPort team, for his valuable support.

References

1. Apple Store: uPortID (2020), https://apps.apple.com/us/app/uport-id/id1123434510#?platform=iphone

2. van Bokkem, D., Hageman, R., Koning, G., Nguyen, L., Zarin, N.: Self-sovereign identity solutions: The necessity of blockchain technology. arXiv preprint arXiv:1904.12816 (2019)
3. Consensys: Ethereum Smart Contract Best Practices - Security Tools (2020), https://consensys.github.io/smart-contract-best-practices
4. Dunphy, P., Petitcolas, F.A.: A first look at identity management schemes on the blockchain. IEEE Security & Privacy **16**(4), 20–29 (2018)
5. Etherscan: Ethereum Developer APIs (2020), https://etherscan.io/apis
6. Foundation, D.I.: DID resolver for Ethereum Addresses with support for key management (2020), https://github.com/decentralized-identity/ethr-did-resolver
7. Google Play: uPort (2020), https://play.google.com/store/apps/details?id=com.uportMobile
8. Mire, S.: Blockchain For Identity Management: 33 Startups To Watch In 2019 (2020), https://www.disruptordaily.com/blockchain-startups-identity-management
9. MythX: MythX User and Developer Guide (2020), https://docs.mythx.io/en/latest
10. Nuss, M., Puchta, A., Kunz, M.: Towards Blockchain-Based Identity and Access Management for Internet of Things in Enterprises: 15th International Conference, TrustBus 2018, Regensburg, Germany, September 5–6, 2018, Proceedings, pp. 167–181 (01 2018). https://doi.org/10.1007/978-3-319-98385-1_12
11. Palatinus, M., Rusnak, P., Voisine, A.: Bitcoin bip39 (2020), https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki
12. Panait, A.E.: uPort etherscan transactions (2020), https://github.com/apanait/EtherscanTransactions
13. Panait, A.E., Olimid, R.F., Stefanescu, A.: Identity management on blockchain–privacy and security aspects. Proceedings of the Romanian Academy, Series A **21**(1), 45–52 (2020)
14. Roos, J.: Identity management on the blockchain. Network **105** (2018)
15. Sovrin: Sovrin: A protocol and token for self-sovereign identity and decentralized trust (2020), https://sovrin.org/wp-content/uploads/Sovrin-Protocol-and-Token-White-Paper.pdf
16. Swagger: API development for everyone (2020), https://swagger.io
17. Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E., Alexandrov, Y.: Smartcheck: Static analysis of ethereum smart contracts. In: Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. pp. 9–16 (2018)
18. Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F., Vechev, M.: Securify: Practical security analysis of smart contracts. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 67–82 (2018)
19. uPort: A platform for self-sovereign identity draft version (2016-10-20) (2016), https://blockchainlab.com/pdf/uPort_whitepaper_DRAFT20161020.pdf
20. uPort: Ethereum registry for ERC-1056 ethr did methods (2020), https://github.com/uport-project
21. uPort: uPort (2020), https://www.uport.me
22. W3C: Decentralized Identifiers (DIDs) v1.0 (2020), https://w3c.github.io/did-core
23. Wang, X., He, J., Xie, Z., Zhao, G., Cheung, S.: Contractguard: Defend ethereum smart contracts with embedded intrusion detection. CoRR **abs/1911.10472** (2019), http://arxiv.org/abs/1911.10472 [7]

[7] All links were last accessed March 2020