2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering

DecentID: Decentralized and Privacy-preserving Identity Storage System using Smart Contracts

Sebastian Friebe
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
friebe@kit.edu

Ingo Sobik[1]
esentri AG
Ettlingen, Germany
ingo.sobik@esentri.com

Martina Zitterbart
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
zitterbart@kit.edu

***Abstract*—Many Internet services require the registration of an account before permitting use of their services. Over time, many Internet users end up with a multitude of accounts with separated identities. A solution to this problem is offered by single-sign-on (SSO) providers, where a user can create a single identity and use this identity for multiple services. However it requires the user to trust the SSO provider. When the provider blocks access to the identities the users lose access to their subscribed services. To avoid this problem, we propose *DecentID*, a completely decentralized identity storage system that does not require a centralized trusted third party. Instead, a public blockchain is used as trust anchor. Identities can be created and used for different services. Each service can only read the identity attributes disclosed for it without being able to read attributes the user wants to keep secret.**

***Index Terms*—identity, decentralized, privacy, blockchain, sybil**

I. Introduction

Many current Internet services, for example interactive websites, require their users to create an account before granting access to the offered services. As a consequence, users tend to use the same credentials for multiple services to make memorizing them easier. While this simplifies the handling of their identities it poses a potential security risk.

Another simplification for users is the introduction of Single-Sign-On identity providers. They allow users to create a single account and use this account to log in on different services. This makes the management of their identities much easier, but users have to trust their identity provider. The provider has full knowledge of the stored identity data and has the possibility to track the actions of users as well as denying access to other services. Additionally, the identity provider is a single point of failure and the stored data poses a tempting target for attackers.

A similar approach are federated identity providers. Users have to register on only one of the providers. When they log in on a cooperating website later on, they are forwarded to their selected identity provider to enter their credentials. This way users have the convenience of using Single-Sign-On but can still choose an identity provider they trust.

However, all these approaches share a common problem with regard to trust: The user has no direct control over the stored identity data but a dishonest identity provider can modify the stored data in any way it wants. A possible solution to this problem can be a decentralized system where users are not required to place trust into a single instance.

Unfortunately, a frequent problem in fully decentralized systems is the possibility of Sybil attacks [1]. In these, a single attacker creates multiple identities, violating the usual "one identity per user" assumption of many systems. In decentralized systems, there is no single registration server that can act as trust anchor and mitigate this problem. A solution for this can be the utilization of smart contracts in blockchains. Everyone can verify the correct execution of smart contracts, making it possible to create a decentralized trust anchor.

In this paper, we propose *DecentID* a decentralized identity storage system based on Ethereum smart contracts. DecentID avoids a single trust anchor, protects the privacy of the users identity data and allows the user to assemble multiple identities out of subsets of their stored attributes, e.g., addresses, photos, or videos. While being decentralized, our system still allows the user to manage their attributes and identities in one place. This way, the user only has to trust a verifiable system instead of trusting into the benignancy of an identity provider. To protect the privacy of the user's identity data, the access to identities and attributes by services is restricted and has to be granted by the user. Due to this, users stay in control of their own identity data.

Using blockchains as data storage only makes sense for small amounts of data since each stored byte has to be paid for. For this reason, we combine a blockchain with an external storage organized as key-value store, e.g., a distributed hash table (DHT). By using cryptographic data stored in the blockchain, we can ensure the integrity of the identity data in the external storage.

Our specific contributions are the following:

- *DecentID*, a novel decentralized storage system for identity data using smart contracts without the necessity of any centralized trust anchor.
- Privacy preserving storage and user-controlled disclosure of identity attributes.
- Support of large attributes by linking an external storage.

Section II provides a high-level overview and an example use case. In section III, technical details are explained. An evaluation is presented in section IV. Section V covers related work, section VI concludes the paper.

[1] Work performed while at KIT

2324-9013/18/31.00 ©2018 IEEE
DOI 10.1109/TrustCom/BigDataSE.2018.00016

II. Basic terminology and example use-case

Within this section we introduce some basic terminology of DecentID and provide an introductory example. A rough overview of the components of the system is given in figure 1.

A. Basic terminology

Identities are associated with human users or Internet services. They are represented by an asymmetric key pair. Access to an identity is granted through a so-called *shared identity contract* (SIC) in the blockchain. This way, referenced identity *attributes* can be shared between the *issuer* of the attribute (who created it) and authorized *owners* (with access to it, e.g., web services).

Selected attributes can be associated with an identity. These attributes, however, might require significant storage capacity. Therefore, they are not stored in the blockchain itself, but are outsourced to the accompanying DHT. Since one goal of DecentID is the flexible provisioning of fine-grained access control to single attributes, each attribute is stored as a separate DHT entry. *Attribute contracts* (AC) are used to reference the attribute data in the DHT and to ensure its integrity. References to the ACs are stored in *attribute locator files* (ALF) to link an identity with a specific instantiation of attributes.

By creating attributes and linking them from shared identity contracts, the user is able to assemble multiple identities for different contexts. These identities can be composed to only contain the attributes necessary for a service. Unwanted disclosure of any attributes the user wants to keep secret can be avoided.

For an easier management of identities, a *root identity contract* (RIC) contains a link to a private *identity locator file* (ILF) listing all SICs of the user. To create such a contract, the user has to submit their public key to a global *registry contract* (RC) in the blockchain.

B. Example use-case

In this section an exemplary use of DecentID is described. A user wants to register in a web forum, which implies the creation of a SIC. It is assumed that the web forum is already registered in DecentID.

1) Registering to DecentID: The user registers to DecentID by providing their public key *pkPrimary* and paying a registration fee within Ethereum to the logically centralized registration contract. When doing so, the RC creates a new RIC and stores *pkPrimary* in the registry and in the new contract.

2) Creating a Shared Identity Contract: The user can now create a new SIC. The public key $pkIdent_I$ of a new key pair created by the user is stored as issuer and owner in the SIC. This new key pair is subsequently used for all activities regarding the new shared identity. Afterwards, the user can create attributes – consisting of some *attribute data* (AD) stored in the DHT – which are indirectly referenced by the SIC. Values of the attributes are encrypted with the symmetric key *kAttr*. An example for such an attribute would be the intended user name for the web forum.

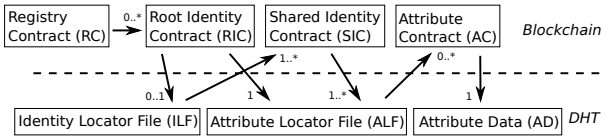

Fig. 1. Interaction of blockchain and external storage (DHT). The arrows mark which components reference which and how many other components

3) Registering to the web forum: The user grants the web forum access to the SIC and their attributes by adding the public key $pkIdent_O$ of the web forum as an owner to the SIC. Thereby, the web forum indirectly gains access to the key *kAttr* and is now able to read the encrypted attributes referenced by the SIC. Additionally, the web forum is now able to add own attributes to the SIC. In this scenario, such an attribute could be a confirmation of the user's rank in the web forum.

When joining another web forum later on, the user can use the granted attribute to confirm the forum rank achieved in the first web forum. This could, e.g., enable the user to start directly as a moderator in the second forum. Since the attributes are stored decentralized, this is even possible if the first forum no longer exists or is temporarily not available.

III. Component Details

DecentID is based on a blockchain as well as on a distributed hash table. Each identity is distributed between both storages and is linked together by cryptographic hashes. The following paragraphs describe the functionality and interactions of the various components of DecentID. The cryptographic keys used are described in section III-E. For an overview of the used cryptographic keys, see figure 2.

A. Registry Contracts

The purpose of the *registry contract* (RC) is to restrict the possibility of creating a lot of Sybil-identities. This contract, depicted in figure 3, provides for each registered user a linkage between a public key and a RIC. A new user registers by providing their public key *pkPrimary* and paying a fee. On registration, a new RIC is automatically created for the user.

Payment of a fee is enforced to provide resilience against Sybil attacks. The idea is that a Sybil attacker has to invest a substantial amount of money to create its Sybil identities

Asymmetric keys

Name	Stored at	Used to
pkPrimary	user, RC, RIC	- encrypt kILF in RIC
skPrimary	user	- sign ALFs
pkIdent	user, SIC, ACs	- encrypt kALF in SIC
skIdent	user	- sign ALFs

Symmetric keys

Name	Stored at	Used to
kILF	RIC	- encrypt ILF
kALF	ILF, SIC	- encrypt ALFs
kAttr	ALF	- encrypt AC - encrypt AD

Fig. 2. Types and uses of cryptographic keys in DecentID

List of tuples of:
- User: $pkPrimary$
- Blockchain address of RIC

Fig. 3. Registry contract (RC)

while a single honest identity is easily affordable. Instead of enforcing the payment of a fee, another registration requirement could also be used.

B. Root Identity Contracts

Owner: $pkPrimary$
Blockchain address of SIC
Enc_{kILF}(DHT address of ILF)
Enc_{kILF}(DHT address of ALF)
$\text{Enc}_{pkPrimary}$(kILF)

Fig. 4. Root identity contract (RIC)

For each registered user one *root identity contract* (RIC, see figure 4) exists. It contains the user's key *pkPrimary* registered in the RC. For its owner, it provides a single entrance point to manage their SICs and their unused ACs, i.e., attributes which have been created but are currently not linked to any SIC. For this, the RIC links to an ALF which in turn links to the unused ACs. Additionally, an *identity locator file* (ILF) links to all SICs of the user. The structure of this file is similar to the ALFs, except that it contains references to SICs instead of ACs. Both of these links are encrypted with the key *kILF* stored in the RIC, with *kILF* itself being encrypted with the public key *pkPrimary* of its owner.

As a special identity, one SIC can be linked as a public identity profile. The data in this profile is readable by all users and can, e.g., be used to link to a public website.

C. Shared Identity Contracts

Issuer: $pkIdent_I$
List of Tuples of:
- Owner: $pkIdent_O$
- $\text{Enc}_{pkIdent_O}$(kALF)
- Enc_{kALF}(DHT address of ALF)

Fig. 5. Shared identity contract (SIC)

Identities in DecentID are represented by *shared identity contracts* (SICs, cf. figure 5). These identities are shared between a user and one or multiple services to allow access to the linked attributes. SICs contain an issuer and one or more entries for owners. The issuer generates the contract while the owners are users with access rights to it. Each owner entry consists of the public key *pkIdent* of the respective owner, the encrypted storage key *kALF* and a reference to an *attribute locator file* (ALF).

An owner can link some of their attributes to a SIC. These attributes are referenced in an ALF which is encrypted with the symmetric key *kALF*. While all owner entries contain the same storage key *kALF*, each copy of *kALF* is encrypted with

Completely encrypted with $kALF$
List of Tuples of:
- $kAttr$
- Blockchain address of AC
Signature $skPrimary$
Signature $skIdent$

Fig. 6. Attribute locator file (ALF)

the public key of the respective owner. Thus, all owners have access to this key and can read the shared ALFs.

As depicted in figure 6, each ALF consists of a list of references to ACs as well as the keys required to decrypt the AC and the AD itself. For each created attribute, a new key *kAttr* is used. Additionally, the ALF contains signatures of the issuer of a SIC and the owner of a RIC. The signature of the issuer of the SIC is used by other owners to verify the authenticity of the ALF and to check whether the user presenting the ALF is really its issuer. Additionally, the signature ensures that only the issuer can modify the ALF. Out of the other signature, the public key *pkPrimary* can be recovered that should be one of the public keys stored in the registry contract, proving the correct registration of the issuer. Due to their size, the ALFs are stored in the DHT and not in the blockchain.

D. Attribute Contracts and Attribute Data

Issuer: $pkIdent_I$
Owner: $pkIdent_O$
Enc_{kAttr}(DHT address of AD)

Fig. 7. Attribute contract (AC)

The *attribute data* (AD) itself is stored as unstructured but encrypted data in the DHT. The AD is referenced by an *attribute contract* (AC), a smart contract which stores the encrypted DHT address of the AD. The structure of the ACs is depicted in figure 7. This indirection is necessary since modifying the data stored in a DHT also changes the associated DHT address. Since smart contracts can modify their stored data without changing their blockchain address, a fixed blockchain address for the attribute is available even when the DHT address changes. Note that the key to decrypt the stored AD and its address is not stored in the AC but in the referencing ALF.

The public keys of the issuer and the owner of the attribute are stored in the AC. The used smart contract ensures that only the registered issuer is able to modify the stored DHT address. However, it can be useful to transfer ownership of the attribute to another identity, permitting the stored owner to prove that the attribute was granted. A possible use-case for this would be a university which certifies the status of its students. In this case the university would be the issuer of the AC while the student would be its owner. After ownership of the attribute has been transfered to the student, the student can use the attribute as a proof to, e.g., receive a student discount in online shops.

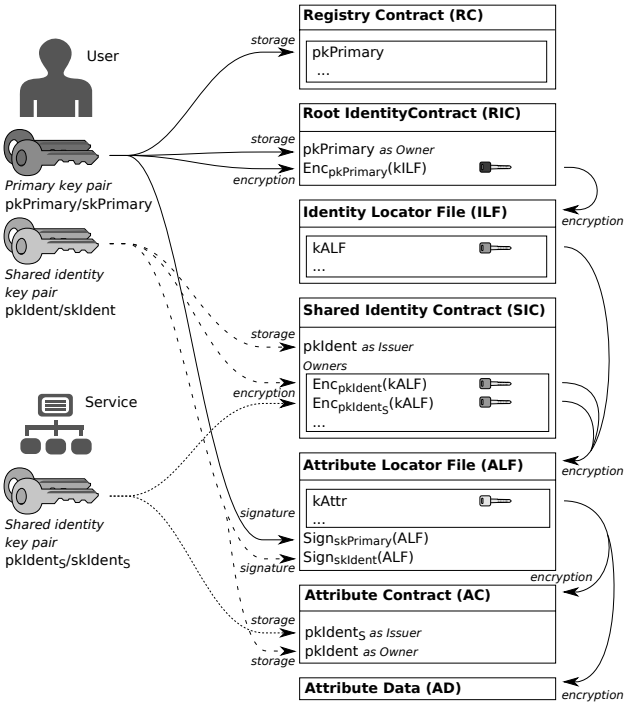

Fig. 8. The cryptographic keys used in DecentID. In the depicted scenario the service shares a SIC with the user and granted an attribute to the user

E. Cryptographic keys in DecentID

Overviews which keys exists, where they are stored and how they are used are given in figures 2 and 8.

Multiple asymmetric key pairs are used by each user. The public key *pkPrimary* of the primary key pair is registered in the RIC when paying the initial fee. A digital signature of the associated private key *skPrimary* is later on written to all ALFs of the user. This allows services to verify that the user is registered.

For each SIC, an additional key pair is used. Its public key *pkIdent* is stored in a SIC and a signature is stored in the ALF to verify the linkage to the SIC. Additionally, it is used inside the SIC to encrypt the symmetric key *kALF* for the ALF. *kALF*, created by the issuer of a SIC, is shared between all owners and used to encrypt all ALFs linked to this SIC. Since all owners have access to *kALF*, they can read the shared attributes of all other owners.

Within the encrypted ALF a list of references to ACs is stored. Together with each reference a symmetric key *kAttr* is stored that is only used to encrypt the data in a single AC and the linked AD itself. Since each attribute has its own key created by its issuer, it is not possible to decrypt other attributes that have not been explicitly shared.

IV. Evaluation

The evaluation of DecentID is structured into three parts: An analysis of the privacy protection provided by the system, performance measurements based on a prototype implementation, and an analysis regarding the practicability.

A. Privacy protection

When evaluating the privacy protection of DecentID the privacy of separate attributes can be differentiated from the unlinkability of multiple SICs.

The privacy of single attributes is protected by their encryption. Only when an attribute is explicitly shared with another participant it is possible to read the value of the attribute. Changing the value of an attribute is impossible for all but the issuer of it. The issuer of an attribute can revoke access by removing it from the SIC. As soon as the cryptographic key to the attribute changes it becomes impossible to read its value for the other owners of the SIC. What can not be avoided is that the other owners can store local copies of the previously read attribute value. This problem generally can not be avoided when the attribute is readable even once. Similar considerations apply when removing owners from a SIC.

For a malicious service provider it can be interesting which other SICs a user created. However, it is not possible to determine this by only using the data stored in the blockchain and the DHT. The RIC of a user can be found by accessing the RC of DecentID. SICs of the user cannot be discovered. While a smart contract on the blockchain can be identified to be a SIC, the only reference linking a SIC to the RIC of the same user is the signature of their *skPrimary* stored within the ALF in the DHT. Since the complete ALF is encrypted with a key unknown to all but the owners of the SIC, no linkage is possible for others.

B. Performance measurements

To evaluate DecentID, a prototype has been implemented and used to measure the performance of retrieving attributes.

1) Test setup: The prototype implements all mentioned smart contracts but due to restrictions of the *web3.js* framework of Ethereum, the asymmetric encryption is future work. While a method for creating digital signatures using the key pair of an Ethereum account exists, there are currently no methods available for asymmetric encryption and decryption.

For conducting performance measurements, three options exist: The Ethereum mainnet, the testnet or an own, local network. For the evaluation of DecentID a local network has been used to avoid measurement artifacts that may occur on the testnet. Our evaluation setup consists of one bootstrap node and three full nodes virtualized in Docker containers. One of the full nodes is a miner. In addition to the Ethereum network a distributed hash table is required for DecentID. Since in the future it might become possible for Ethereum smart contracts to interact with it directly, *Swarm* [2] has been selected as DHT and is run in the containers of the full nodes. Alternatively, any other key-value store could be used instead of Swarm.

2) Retrieval performance: The retrieval times of attributes were evaluated for different sizes of attributes. The retrieval times are separated into retrieval from the blockchain, local cryptographic operations and retrieval from the Swarm DHT. The resulting times are shown in figure 9.

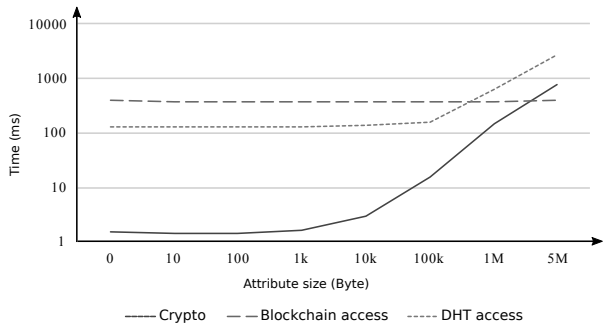

Fig. 9. Time of attribute retrieval for different attribute sizes

a) Blockchain access: As the measurements show, the time periods of interaction with the blockchain remain constant over all attribute sizes. This behavior is expected since the amount of retrieved data is always the same, independent of the attribute size. Even when the attribute data in the DHT becomes larger, the cryptographic keys and DHT indices stored in the blockchain are of a fixed size.

b) DHT access: With attribute sizes between 100 KByte and 5 MByte the retrieval time of attributes increases proportionally. In contrast to that, smaller attribute sizes show nearly constant retrieval times. This can be explained with the overhead of the Swarm DHT. Even when the local testing network only consists of two nodes, a certain overhead is required to locate and retrieve the requested data. Compared to this, the actual transmission of a small attribute is negligible.

c) Cryptographic operations: The third measured time interval is the time needed for cryptographic operations. Similar to the times for retrieving data from the DHT, calculation times for small attributes remain mostly constant. As long as the size of the attribute is less than 1 KByte, calculations for size independent parts of the system, e.g., the signature of the attribute locator file, dominate the required overall time.

C. Conceptual analysis

Some properties of DecentID are unsuited for evaluation in experiments as they depend on the current state of the blockchain network. For example, the writing speed of attributes depends heavily on the activity of other blockchain users. Hence, these properties are evaluated conceptionally in the following paragraphs.

1) Writing attributes: The performance of write-operations on the blockchain depends mainly on two factors: The number of active users of the blockchain and the block time.

The activity of Ethereum users affects the number of transactions that are in newly created blocks. The time until attributes or contracts are written increases with the number of transactions that are competing for inclusion in the blockchain. While the Ethereum blockchain is able to adapt to increasing amounts of transactions, this process is relatively slow and can be a limit for future growth.

The block time, i.e., the delay between two blocks, is currently at around 15 seconds per block in Ethereum. Thus, new attributes could theoretically be added without much delay. However, compared to traditional storage systems like databases this is not competitive. Consequently, DecentID is not suitable for applications which require timely updates of attributes. Frequent reading of attributes does not pose a problem, since it is done on a local copy of the blockchain.

2) Scalability: The scalability of DecentID depends on two aspects: The Swarm DHT and the Ethereum blockchain. Since Swarm is still in an early stage of development, reliable scalability information about it is not yet available. However, the DHT used in DecentID can be replaced easily if Swarm turns out to be an inadequate choice.

The reading capability of the Ethereum blockchain can be scaled easily. Since each full node stores a copy of the complete state of the blockchain, reading data is done on a local full node without requiring communication with other nodes. Due to this, the reading performance can easily be improved by deploying supplemental local Ethereum nodes.

Additionally to the aforementioned restrictions, the amount of data that can be written is also constrained by the size of transactions. While the number of transactions per block is not directly limited, each transaction costs *gas* and each block has a maximum amount of gas it can use. Due to this, only a restricted number of transactions can be stored in one block. However, interacting with smart contracts costs more gas than conducting financial transactions, especially when storing data in the blockchain. As a result, less DecentID transactions can be contained in a block than normal financial transactions.

3) Monetary costs: To send transactions to the Ethereum network, gas has to be provided. This gas can be bought by paying with Ether which in turn has to be bought in advance. This means that using DecentID incurs a small fee for modifications like storing attributes.

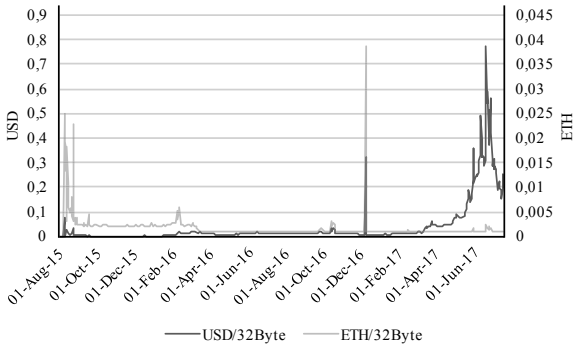

Fig. 10. Financial costs for storing one hash in the blockchain Ethereum

Based on the statistics in figure 10 it can be seen that the price per byte is highly volative. This influences the costs of storing the cryptographic hashes used by DecentID in the blockchain. In any case, the costs of storing attributes will be much higher than using a traditional storage system, e.g., a database or cloud storage. Whether the price is acceptable depends on the use case of the system.

V. Related Work

This work combines two independent areas of research: Blockchains, which are used as a building block in this work, and identity management, which this work contributes to.

A. Identity Management

Initially, many independent centralized identity management systems were used. With those, each service provider stored its own set of user accounts with completely separated identities. A disadvantage of this approach is that users have to keep their data up to date and synchronized at all providers. As a result Single-Sign-On providers emerged to simplify the log in procedure for users. With these, a user can create a single identity and use it for multiple services. A protocol implementing this design is the Security Assertion Markup Language (SAML) [3]. In the context of local networks Kerberos [4] can be used as a Single-Sign-On provider.

However, these providers are (logically) centralized and the users have to trust them to use connected services. Federated identity providers like OpenID [5] aim to mitigate this problem. Users register an identity on one of the servers of the federation and use that identity everywhere where support for the federation is available.

B. Blockchains

Ethereum [6] was developed to support abilities of the scripting system. With it, Turing complete scripts can be created with the programming language Solidity [7]. These scripts are called smart contracts and improve heavily upon the simple payment scripts possible within Bitcoin [8]. Smart contracts offer decentralized computation and storage, making them a versatile tool for developing distributed systems.

To store data in blockchains, the Namecoin [9] network was started. Its purpose is to provide a distributed, robust and censor-resistant database, with a replacement of the centralized DNS system as primary use-case. In [10] an approach for private data storage based on a blockchain and a DHT is presented. Similar to our work, the focus is on privacy preserving use and sharing of data. However, their approach uses its own blockchain, effectively linking the security of the system to only the nodes that participate in the system itself.

Blockstack [11] is similar to Namecoin since key-value pairs can be stored. A difference to Namecoin is that no separate blockchain is used but the system is based on Bitcoin for security reasons. Furthermore, data is not directly stored in the blockchain but instead an external storage is referenced. Blockstack also provides user management and a platform for decentralized applications, but is not designed for assembling and privacy preserving storage of identities.

Another platform focused on identities is uPort [12]. Based on the Ethereum blockchain, uPort allows to create identities and their attributes. Attributes are stored in a freely chosen external storage while the blockchain ensures their integrity. Privacy protection does not seem to be a design goal of uPort. Disclosing only individual attributes is not possible. Only a complete block of attributes can be linked to an identity.

VI. Conclusion

With DecentID we demonstrate the possibility of a completely decentralized identity storage system by combining the integrity of the blockchain Ethereum with the storage of a DHT. DecentID can be used to store attributes, assemble separate identities from them and use these identities for different services. Services are then able to grant the user further attributes that can later be used with third-party services, e.g., to prove membership in an organization.

Compared to currently deployed Single-Sign-On providers no trusted third party is required. Additionally, the privacy of the user identity data is ensured since no instance is able to read attributes of a user without their explicit consent.

In its current version, DecentID requires the linkage of identities to a logically centralized registration contract. While this is required to thwart Sybil attacks it allows cooperating services to discern which shared identities belong to a user. In future work we will develop a concept to allow for shared identities which cannot be linked to each other. Additionally, the initial registration fee will be replaced by some other, more secure, Sybil-resistance mechanism, e.g., based on trust relationships embedded in social networks.

Acknowledgment

This work was supported by the German Federal Ministry of Education and Research within the framework of the project KASTEL_ISE in the Competence Center for Applied Security Technology (KASTEL).

References

[1] J. R. Douceur, "The sybil attack," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK: Springer-Verlag, 2002, pp. 251–260.

[2] Swarm docs. http://swarm-guide.readthedocs.io/en/latest/index.html. Accessed: 2017-07-01.

[3] S. Cantor, J. Kemp, R. Philpott, and E. Maler, "Assertions and protocols for the oasis security assertion markup language (saml) v2.0," OASIS, Tech. Rep., March 2015. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[4] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The kerberos network authentication service (v5)," Internet Requests for Comments, RFC Editor, RFC 4120, July 2005. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4120.txt

[5] D. Recordon and D. Reed, "Openid 2.0: A platform for user-centric identity management," in *Proceedings of the Second ACM Workshop on Digital Identity Management*, ser. DIM '06. New York, NY, USA: ACM, 2006, pp. 11–16. [Online]. Available: http://doi.acm.org/10.1145/1179529.1179532

[6] V. Buterin, "Ethereum white paper: a next generation smart contract & decentralized application platform," 2013.

[7] Solidity documentation. https://solidity.readthedocs.io/en/develop/. Accessed: 2017-07-01.

[8] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[9] Namecoin.org. https://namecoin.org/. Accessed: 2017-05-08.

[10] G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *Security and Privacy Workshops (SPW), 2015 IEEE*, 2015, pp. 180–184.

[11] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, 2016, pp. 181–194.

[12] D. C. Lundkvist, R. Heck, J. Torstensson, Z. Mitton, and M. Sena. (2017) [whitepaper] uport: A platform for self-sovereign identity. https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf.