A Security Analysis of the Emerging P2P-based Personal Cloud Platform MaidSafe

Florian Jacob, Jens Mittag, Hannes Hartenstein
Institute of Telematics & Steinbuch Centre of Computing
Karlsruhe Institute of Technology, Germany
florian.jacob@student.kit.edu, {jens.mittag, hannes.hartenstein}@kit.edu

***Abstract*—The emergence of decentralized crypto currencies such as Bitcoin and the success of the anonymizing network TOR lead to an increased interest in peer-to-peer based technologies lately – not only due to the prevalent deployment of mass network surveillance technologies by authorities around the globe. While today's application services typically employ centralized client/server architectures that require the user to trust the service provider, new decentralized platforms that eliminate this need of trust are on their rise. In this paper we critically analyze a fully decentralized alternative to today's digital ecosystem – MaidSafe – that drops most of the commonly applied principles. The MaidSafe network implements a fully decentralized personal data storage platform on which user applications can be built. The network is made up by individual users who contribute storage, computing power and bandwidth. All communication between network nodes is encrypted, yet users only have to remember a username and password. To guarantee these objectives, MaidSafe combines mechanisms such as Self-Authentication, Self-Encryption, and a P2P-based public key infrastructure. This paper provides a condensed description of MaidSafe's key protocol mechanisms, derives the underlying identity and access management architecture, and evaluates it with respect to security and privacy aspects.**

I. Introduction

Despite the fact that the Internet was initially designed as a de-centralized network of services, the ecosystem of today's Internet looks different on most layers that reside above the transport layer. When looking at services such as Google Docs, Office 365, Facebook, and Paypal – to name only a few – this difference has two key implications: (1) the introduction of a trust requirement into the service provider, and (2) dependency on a central system. Furthermore, these systems are, e.g., due to business related reasons, not open anymore: users cannot freely choose to run parts of the service locally or choose a different service provider while remain being connected with users that are still in contract with the old service provider. When looking at the shutdown of the encrypted webmail service Lavabit in 2013, which discontinued its service as the operating company did not want to cooperate with the U.S. government, the weakness of such centralized services against authorities or individual powerful parties becomes also visible.

In parallel to the increased awareness w.r.t. these issues, and in reply to the decrease of trust into individual centralized service providers, new platforms and networks developed during the past years. Bitcoin [1] for instance is a peer-to-peer based digital currency system and allows to perform transactions without the need of intermediates such as banks. Bitcoin uses a public distributed database called the Blockchain which contains records of all transactions executed in the past. Ethereum [2], a platform for decentralized applications, also uses a blockchain in order to implement a cryptographically secure, transaction-based state machine. It can be considered as a generalization of Bitcoin which allows to implement other services such as voting systems, domain name registries, self-enforcing contracts and agreements, smart property, and distributed autonomous organizations. A similar but rather storage-oriented approach is the project called MaidSafe [3], which calls itself "the new decentralized Internet". Tor [4], a peer-to-peer based anonymizer that protects its users against traffic analysis, a common form of Internet surveillance, is also a very prominent example of the de-centralization trend.

The success of these emerging platforms convinced several companies around the world to interconnect with them or to adopt their principles. Dell and Expedia, amongst others, are now accepting payments in Bitcoin. IBM goes even further in an executive report of their Device Democracy Program [5] and states that the Internet of Things (IoT) can only survive the end of trust and scale to billions of devices if the currently applied principles are re-thought at their foundation. Together with Samsung they recently show-cased a completely de-centralized IoT prototype platform called ADEPT (Autonomous Decentralized Peer-To-Peer Telemetry) [6] which applies the principle of privacy-by-design.

It may appear that de-centralized approaches will solve many of the security and privacy issues we are facing in today's Internet. They promise to reduce the power of service providers and to eliminate the need of trust into them. However, it is unclear whether this reduction and elimination is exclusively a gain or maybe also a loss from a user's perspective. For instance, it may be difficult to guarantee availability of stored data. The limitation and termination of a security or privacy attack is also non-trivial if the underlying platform runs completely de-centralized. If everything is distributed, security and privacy properties are based completely on cryptographic algorithms, which eventually might be broken in the future.

In this paper, we provide a condensed description of the emerging P2P-based personal cloud platform MaidSafe and its core protocol mechanisms. We then derive the underlying trust model, critically analyze MaidSafe w.r.t security and privacy aspects and discuss the inherent drawbacks of such a decentralized architecture. The rest of the paper is structured as follows: Section II reviews related work, Section III presents a condensed description of MaidSafe, followed by the analysis and evaluation of MaidSafe's trust model and architecture in

 doi:10.1109/Trustcom.2015.538

Section IV. Our conclusion is given in Section V.

II. Related Work

Security considerations for P2P-based distributed hash tables (DHTs) were already discussed by Sit et al. in 2002 [7]. The authors compiled a list of potential network attacks (e.g. routing attacks or data storage and data retrieval attacks) and provided general guidelines how these attacks could be defended. They identified one key issue which they termed *verification of node keys* and outlined an authentication solution which is adopted by MaidSafe. Similar privacy-preserving authentication approaches can be found in [8] and [9].

At the same time, Douceur et al. identified the well known *Sybil Attack* [10] which de-centralized P2P networks are vulnerable to. In a sybil attack, an adversary creates a large number of pseudonymous identities to gain a large influence on the overall network with the objective to control operation. A classification of sybil attacks and methods to avoid them were later provided by Dinger et al. in [11]. The aspects of privacy, security and trust were previously discussed by Mondal et al. in [12], however, with a focus on the applicability of the P2P paradigm on domains beyond traditional file-sharing applications.

A security analysis that specifically focuses on data confidentiality, data integrity and data availability within the MaidSafe network is provided by the authors of MaidSafe [13]. Yet, the provided analysis is limited to the mechanisms that implement MaidSafe's distributed storage system.

III. MaidSafe

The SAFE network (or MaidSafe) refers to *Secure Access For Everyone* and is described by their authors as a fully distributed data management service that offers secure data storage and secure data retrieval. The following subsections provide a condensed summary of its design goals, the implementation of its core features, and its underlying identity and access control management (IAM) architecture.

A. Design Goals & Assumptions

MaidSafe aims to satisfy the following design goals. The mechanisms which implement the design goals are described in the following subsections.

1) Autonomous network operation without reliance on centralized servers or components.
2) No need to trust single entities or operators.
3) Ease of use from the perspective of its users. Users need to remember only their username and password. All other data shall be stored in the network.
4) Privacy by design: nobody has access to anyone else's data unless specifically shared.
5) Availability and integrity: stored data shall be available at all times and integrity protected.
6) Protection against targeted and mass surveillance of users.

To facilitate comprehension in the subsequent subsections, we prepend several design principles and assumptions: (1) MaidSafe stores all data in a distributed hash table (DHT), i.e. a common address space which is shared over all nodes of the MaidSafe network; (2) all nodes keep an open connection to at least four of their closest neighbors; (3) all communication between nodes is authenticated, hence nodes are identifiable by an asymmetric public key accessible through a P2P public key infrastructure [14]; (4) all communication between nodes is encrypted.

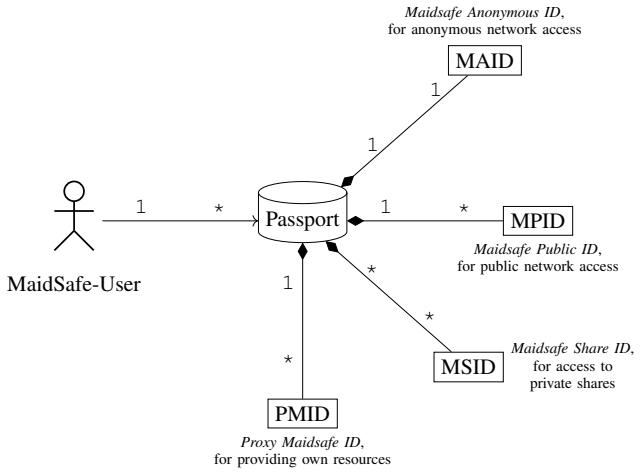

Fig. 1. Overview of the various access methods of a user to the MaidSafe network

B. Trust & Identity Management

From a user's point of view, their passport is the central concept in the MaidSafe network. As a user can have more than one passport it roughly corresponds to a user account as with traditional services today. With each passport, a user has several views or access possibilities to the network, depending on the aspect of the network they want to use currently. For each of these aspects, different identities (IDs) are used to communicate with the network. All of these identities are stored in the passport, see Figure 1, and are defined as follows:

- for anonymous network usage like storing or retrieving the user's own personal data, the *MaidSafe Anonymous ID (MAID)* is used.
- *MaidSafe Public IDs (MPIDs)* correspond to usernames or email addresses and are used to send messages between users, or to store public data such as a personal website. Each user can own an arbitrary number of MPIDs.
- in case the data to store is not public, but should still be shared with a limited user group, the user can create a private share. Each share is identified by a *Maidsafe Share ID (MSID)*, which is also used to store and retrieve the share's data. Therefore, each user that has access to the share knows the MSID.
- in case the user wants to contribute resources like storage of his own machines to the network, the last type of IDs are used: the user starts one or more DHT nodes per machine, and each node gets its own *Proxy MaidSafe ID (PMID)* to communicate with the network – independent from the user's own activity.

The main purpose of this separation is anonymization, as there is no direct way to link different identities to a user.

All IDs consist of two public key pairs: k_1^{priv}, k_1^{pub} and k_2^{priv}, k_2^{pub}. The first pair k_1 is used to encrypt and sign all communication with the network and other users, as well as to encrypt data before storing it. k_2 is only used to validate the identity in case k_1 gets compromised and has to be revoked. Using these pairs, the ID fingerprint (usually referred to as ID as well) is then computed as

$$ID = h(k_1^{pub} || s_{k_2^{priv}}(k_1^{pub})) \quad (1)$$

where h equals SHA-512, $||$ refers to string concatenation and $s_{k_2^{priv}}(x)$ corresponds to the signature of x using k_2^{priv}.

In case the ID fingerprint represents a PMID, it is also used as the node's position in the DHT. This is a key design decision, as the security of MaidSafe requires that one can not choose a specific desired position easily. This requirement is satisfied as long as the hash function h in Equation 1 is not broken (which is the case for SHA-512 at the time of writing) and if assumption 3 (all communication is authenticated) applies. Authentication in this context means that a node proves that he knows the public/private key pairs that yield his position in the DHT.

With respect to trust, MaidSafe relies on the condition that any group of k nodes that are closest to an arbitrarily chosen ID (within the DHT address space) does not cooperate in order to execute an attack. The group is considered to cooperate if the majority of nodes in the group cooperates. Hence, the nodes are expected to mutually check and validate their operation.

C. Self-Authentication

MaidSafe employs a mechanism which is termed *Self-Authentication* by their authors. It allows to authenticate users without communicating to a central, trusted identity (or service) provider, while guaranteeing that the credentials of the users never leave their machines [3]. To satisfy this design goal, the process is divided into two phases: (1) determining the DHT location and subsequent retrieval of a so called *access packet*, and (2) retrieval of the user's passport from the location that is calculated with the help of the information stored within the previously fetched access packet [15]. Once the passport is retrieved (and decrypted) the user is considered to be logged in.

To clarify how the process works in detail let us focus on phase one first: retrieval of the information stored within the access packet. It consist of the following sequence of actions:

1) Calculation of the location of the access packet as $h(u||s)$ where h is the hash function SHA-512, u the username, s a salt that is derived from u and p (p being the password of the user), and $||$ the string concatenation function
2) Retrieval of the (encrypted) access packet ap using the DHT location calculated in the previous step
3) Calculation of the encryption key k_a that was used to symmetrically encrypt ap, as $k_a = k(u, s)$, whereas k refers to the password-based key derivation function in version 2 as defined by RFC 2898[16] and u or s to the username and salt as above
4) Decryption of ap using k_a in order to receive the random value v, which will be used in the second phase.

Obviously, the username and password never leave the user's device in plain text. However, they are used as input to the salt creation function (not clearly specified in the specification), which in turn is used to derive the location of the access packet. As username and password are used to derive the salt, two users with the same username will not run into an identity collision as long as their passwords are not equal as well.

The sequence of actions in the second phase, i.e. retrieval of the passport, is then as follows (re-using the notation introduced above):

1) Calculation of the location of the passport as $h(u||s||v)$, with h being SHA-512 again, u the username, v being the result value of phase one, and $||$ the string concatenation function
2) Retrieval of the passport pp from the DHT location calculated in the previous step
3) Calculation of the encryption key k_p, that was used to symmetrically encrypt pp, as $k_p = k(p, s)$, with k being defined as before, p being the password of the user, and s being the salt
4) Decryption of pp using k_p in order to receive the passport of the user

As can be seen, phase two is very similar to phase one, with the difference that the calculation of the passport location uses the value stored within the access packet, and that the password-based key derivation function k takes the user password p and not the username u as input.

The MaidSafe documentation mentions two enhancements that are meant to improve the security of the Self-Authentication mechanism. The first enhancement suggests to move the passport to a different location whenever the user logs out (and consequently update the value v stored within the access packet), and the second one suggests to move the access packet as well. The second enhancement could be implemented by using the current date or week during the calculation of the access packet's location. The access packet would then move to a different location as well during user logout. The implications of both enhancements will be discussed in Section IV.

D. Secure Distributed Storage

The actual storage of data in MaidSafe happens in a DHT based on Kademlia [17], but significantly extended and enhanced to suit the needs of MaidSafe [18].

1) Storing Data in the Network: Storing data in the MaidSafe network is split up to five different kinds of nodes, called *Personas* in MaidSafe. Each Persona has simple tasks, which in collaboration result in anonymous, distributed and secure data storage. The personas will now be explained by means of an example, following a `PUT` request through the network. This request is depicted step by step in Figure 2, which is based on [19].

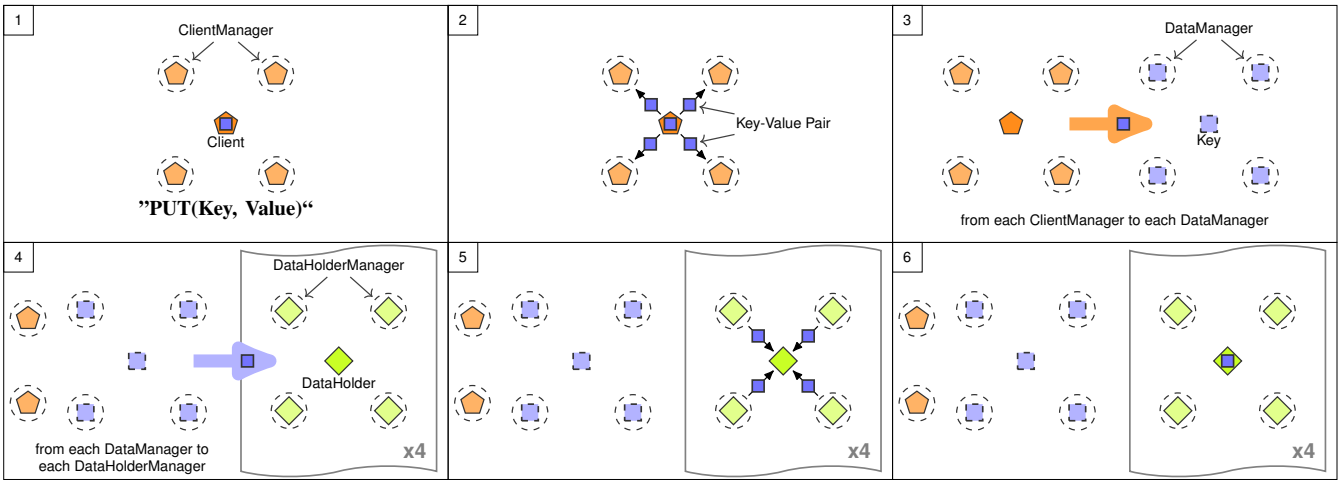

Fig. 2. **Storage of Key-Value Pairs in the network via PUT: 1)** The Client uses a MAID or MPID to connect to its ClientManagers, the 4 nearest nodes to the used ID. **2)** The key-value pair to store is sent to the ClientManagers to hide the Client's IP. **3)** The DataManagers are the 4 nearest nodes to the key. Each ClientManager forwards the request to each DataManager, which take the majority of agreeing ClientManager requests as valid. **4)** To ensure replication and integrity, the DataManagers agree on 4 random DataHolders to store the data. The 4 nearest nodes to each DataHolder are his DataHolderManagers. Each of the 16 in total DataHolderManagers are contacted by each DataManager individually, and each takes the majority of DataManager requests as valid. **5)** The DataHolderManagers forward the request of the majority to their respective DataHolder, which then processes the request received by the majority of his DataHolderManagers, i.e. stores the data. **6)** DataHolderManagers monitor the availability of DataHolders and report the loss of a DataHolder back to the DataManagers, which then agree on additional DataHolders to ensure availability. To ensure integrity, DataManagers perform a „Proof of Resource": They check whether all DataHolders still store the same data (not depicted here), and then, the trusted majority of DataHolders receives Safecoin generated by the DataManagers for their efforts.

Step 1) First, the Client can be either an anonymous MAID or a public MPID of the user. It is connected to the group of four nodes that are closest to its ID, which are the ClientManagers. They are used as proxies by the client and relay all network-level traffic from and to the client. Their purpose is to hide the client's IP address.

Step 2) To store a key-value pair via a `PUT` request, the Client just forwards the request to each of his ClientManagers and isn't involved in anything else.

Step 3) The ClientManagers then independently identify the four DataManagers, which are the four nearest nodes to the key of the key-value pair. Each ClientManager forwards the request to each DataManager, so that each DataManager receives the request independently from four ClientManagers. The DataManagers first check whether the ClientManagers are in fact the nearest nodes to the ID that initiated the request and then take the majority of agreeing ClientManager requests as valid, based on the assumption that the majority of ClientManagers is not compromised.

Step 4) The DataManager's purpose is availability and integrity management, not the actual storing of the data. They agree on four random DataHolders, whose sole purpose is to provide the actual storage. The four nearest nodes to each DataHolder are his DataHolderManagers, therefore, 16 DataHolderManagers exist. Each DataManager forwards the request to each DataHolderManager, so that each DataHolderManager receives one storage request from all DataManagers. The DataHolderManagers validate the four requests by checking whether the DataManagers actually are the four nearest nodes to the key to store and again take the request of the majority of agreeing DataManagers as valid.

Step 5) Each DataHolderManager forwards the request to their respective DataHolder. Again, the DataHolders validate the request by checking whether the DataHolderManagers are the nearest nodes to him and processes the request that the majority of DataHolderManagers agrees on, i.e. stores the actual key-value pair.

Step 6) The DataHolderManagers' purpose is to observe their DataHolder as they keep direct connections to them established and therefore notice the sudden absence of a DataHolder very fast. Additionally, they provide anonymization to the DataHolder, as they are, like the ClientManagers, the only nodes that know the IP address of the DataHolder in this context. As soon as they detect that the DataHolder is gone, they report back to the DataManagers, which then agree on new, additional DataHolders to ensure the replication and therefore the availability of the data. This is how churn handling[1] is performed in MaidSafe.

To ensure integrity, the DataManagers perform a periodic check that is called *Proof of Resource* (PoR) in MaidSafe. The resemblance of the term to Bitcoin's proof of work is intended, as Safecoins are generated through this PoR. For the PoR, the DataManagers send a random value to each DataHolder. The DataHolders answer with the hash of the value plus the stored data, so that the actual data has not to be transmitted. The DataManagers then compare the hashes received from the DataHolders and have to trust the majority, as they don't store the data themselves. The DataHolders that comprise the majority are then rewarded for their provided resources in Safecoin generated by the DataManagers to incentivise correct behaviour, while the DataHolders in the minority (which are none, usually, if all DataHolders returned the same hash) are punished and will receive less data to store and to earn Safecoin through. The Safecoins issued by the DataManagers are

[1] i.e. the handling of nodes that stored data leaving the network and the subsequent redistribution to new storage nodes

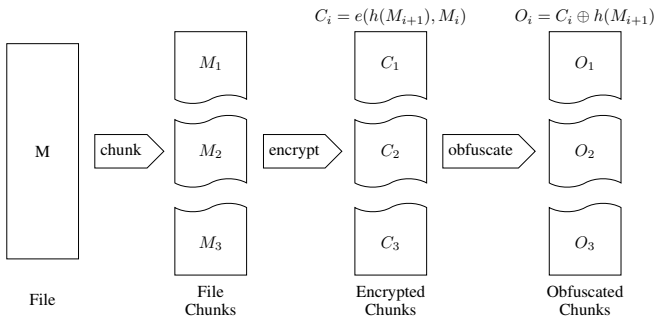

Fig. 3. Overview of the Self-Encryption process, where $e(key, text)$ is the symmetric encryption function AES and $h(key)$ is the hash function $SHA512$. $h(M_i)$ and $h(O_i)$ are stored in the DataMap, which is sufficient to retrieve the data and then reverse the process.

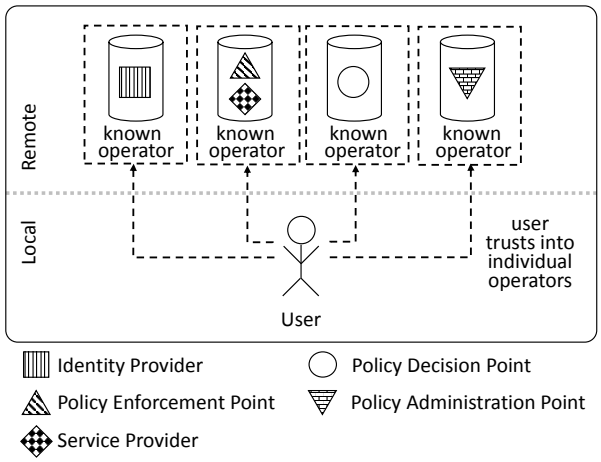

Fig. 4. Traditional identity and access control management architecture (IAM) and existing trust relationships from the perspective of an end-user. The user trusts the (potentially distinct) operators that implement the identity provider, the policy administration point, the policy decision point, and the policy enforcement point at the service provider, because he knows the parties that run the IAM components.

new Safecoins that didn't exist before, analogous to Bitcoin's mining process. It is the only occasion where new Safecoins can be created.

2) Self-Encryption: The mechanism that MaidSafe uses to split and reconstruct files to and from encrypted key-value pairs is called *Self-Encryption*. The only input of the mechanism is the file itself, hence the name. The output consists of multiple AES-encrypted chunks that can be stored in the DHT, as well as a *DataMap* that contains the pre- and post-encryption hashes of the chunks. The process is depicted in Figure 3.

First, the file is split into groups of three equally sized chunks. Instead of using a file fingerprinting approach (such as Rabin Karp) the chunks adhere to a fixed size of at least $kMinChunkSize$ bytes and at most $kMaxChunkSize$ bytes[2]. At the time of writing, $kMinChunkSize$ equals to 1 KB and $kMaxChunkSize$ to 1 MB. In the following, each chunk triple is handled independently.

For each chunk, the pre-encryption hash is computed and used to encrypt another chunk in the triple to form a circle. Then, the encrypted chunks are additionally obfuscated by XOR-ing them with their encryption key. This additional step serves as additional protection in case the encryption is attacked. The obfuscated chunks are finally stored in the DHT (with their post-encryption hashes as key). Those post encryption hashes are combined with the pre-encryption hashes to form the DataMap. For data retrieval the process is simply reverted: first, the encrypted chunks are fetched from the DHT using the post-encryption hashes as the key, followed by the de-obfuscation and decryption with the pre-encryption hash. As the DataMap is key and reference to the file, it is encrypted using the private key of the user and stored in their passport.

Working this way, Self-Encryption also allows deduplication: If two stored files share a group of three equal chunks, they map to the same cipher text at the same position and will be stored only once in the network.

E. Resulting Identity and Access Control Management Architecture & Trust Model

In this section we abstract from the mechanisms applied by MaidSafe and extract the underlying Identity and Access Control Management (IAM) architecture from the perspective of the end-user. Due to its wide acceptance we use the nomenclature of the eXtensible Access Control Markup Language (XACML) specification [20] and first begin with the description of the typical IAM architecture of today's Internet ecosystem, as shown in Figure 4. Following the nomenclature of [20], an IAM architecture comprises several components[3]: an Identity Provider (IdP), a Policy Administration Point (PAP), a Policy Decision Point (PDP), and a Policy Enforcement Point (PEP). These component may all be operated by a single provider, but it is in general possible that they are operated by distinct parties. This is the case when federated identity management principles are applied, for instance if a service provider uses services such as Facebook Connect to offload the operation of the Identity Provider functionality. Nevertheless, in either case, the end-user puts trust into the components, in particular, he trusts the

1) *Identity Provider* that he returns data only to subjects that know username and password, and that he implements security monitoring to detect and block identity attacks (e.g. by limiting the number of login trials).
2) *Policy Administration Point* that access control rules are stored such that they reflect the intentions of the end-user 1:1.
3) *Policy Decision Point* that the defined access control rules are evaluated properly and authorization are generated only to matching subjects.
4) *Service Provider* that he implements a *Policy Enforcement Point* which grants access only to authenticated and authorized subjects.

The above trust relationships are commonly based on the fundamental trust that the applied cryptographic algorithms

[2] see https://github.com/maidsafe/MaidSafe-Encrypt/blob/master/src/maidsafe/encrypt/self_encryptor.cc for further details.

[3] Additional components may exist, e.g. a Policy Information Point, but they are neglected here as their relevance to this analysis is insignificant.

(e.g. to ensure confidentiality, integrity, authenticity, etc.) are secure and not broken.

The IAM architecture of MaidSafe looks less trivial, cf. Figure 5, as the components are distributed over the SAFE network. Instead of putting trust into an easily understandable set of operators, the user has to trust the collective network of unknown MaidSafe users (which are effectively operators in this case) and the applied algorithms. The Self-Authentication mechanism implements the functionality of an identity provider, which is distributed between the nodes that store the encrypted passport and the MaidSafe Client which decrypts it. Policy administration is implemented only in the client software as it selects the encryption key that only the target audience knows and also provides the public key used for signature-based write protection. Policy decision and enforcement for read operations is implicitly performed at the client application through data encryption algorithms. In case of write operations, policy decision and enforcement is additionally handled by the network as data managers and data holders perform majority-based evaluations and signature-checks. All nodes involved in a user's storage request are of course considered as service providers. To summarize, when using MaidSafe the user trusts the

1) *Client software* that it does not include a backdoor and applies the cryptographic algorithms in the proper way.
2) *The SAFE network* that
   a) a successful Self-Authentication is only possible with known username and password, and that online password guessing attacks are effectively infeasible.
   b) data holders and data managers apply the given rules correctly, i.e. they do not ignore a signature-based write protection or simply replace the corresponding public-key with a different one.
   c) nodes cannot choose their network position freely, that neighboring nodes do not collaborate, and that the view on the network is equal for for all nodes[4].

The main principle of MaidSafe is therefore best described by 'Do not trust individual operators, but trust the collective of all operators as well as the security of cryptographic algorithms!'.

IV. Security Analysis

A. MaidSafe-Specific Analysis

In this section, MaidSafe's security is analyzed w.r.t. the classical IT security goals: authenticity, integrity, confidentiality, availability and anonymity. Non-repudiation, i.e. of Safecoin transactions, is neglected, as not enough information is available regarding the details of the transaction mechanisms. The analysis uses the terms of MaidSafe's IAM architecture we developed in subsection III-E.

The internal security concept of the MaidSafe network is almost entirely based on the mutual-control of neighboring

[4]Equal in the sense that all nodes share the same topology information, i.e. the group of N closest nodes to a given key is equal (or at least almost equal) for all network members.

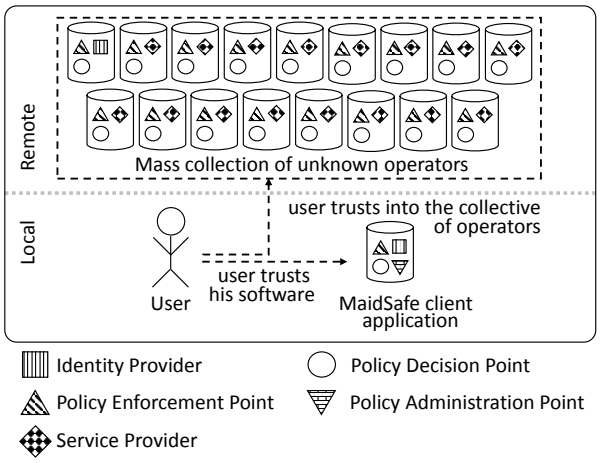

Fig. 5. Illustration of how MaidSafe distributes the different IAM components within its network. Instead of putting trust into individual operators that he does not know, a user puts trust into the collective network and his client software.

nodes within the DHT. This is assumed to be the case due to the non-manipulable position within the DHT and the P2P-PKI based detection of such manipulation attempts. An attack on this property, i.e. if nodes could select specific positions, would kill the security architecture of MaidSafe completely and thereby kill most of the above security goals. Consequently, defeating mutual control among neighboring nodes should be very expensive for an attacker, and in fact requires that he either performs a Sybil attack on the MaidSafe network (which is easier the smaller the MaidSafe network), or that he manages to successfully break the hash function to generate IDs close to a desired position. The costs of a successful Sybil attack can of course be reduced by routing table poisoning [11], which facilitates the presence of malicious nodes within the routing tables of well-behaving nodes. Due to these threats the MaidSafe developers recently published plans to increase the size of the group of nodes that control and check each others operation (i.e. the group of data managers, data holders, etc.) from 4 to 32, and to trust the result provided by the majority of these nodes, whereas majority is achieved by 28 out of 32. However, it is not clear why the MaidSafe developers choose exactly 28 out of 32 and how significant it increases the level of security.

We want to note that the distributed storage process of MaidSafe contains one step where randomness is eliminated: the step in which data holders are selected. In fact, they are chosen by the data managers, and the exact procedure of this selection (or agreement) is undocumented so far. Nevertheless, we don't see a general security risk here as a very simple agreement could be that each data manager suggests one data holder. This does not help in gaining a majority.

Authenticity is the domain of the Identity Provider, which is between the client application and the network. Successful authenticated is performed if a valid username and password combination is provided. No additional security features to distinguish the user from an attacker is present. An adversary can therefore execute an online password guessing attack by trying out any number of username/password combinations, being only limited by his network connection and the performance

of the MaidSafe network. MaidSafe employs no mechanism to limit his login attempts in any way or even to detect what a single node is doing. With prior knowledge about typically used passwords or password structures, the adversary has a significant chance of success to impersonate the accounts of random users that use mediocre passwords. The network itself cannot defend such an attack, but users can protect themselves by using good passwords that are only used for MaidSafe and changed regularly.

In order to perform an offline password guessing attack, the attacker needs access to either an access packet, or the passport itself. This can be the case if the access packet (or passport) of a user is stored on a node that is contributed by an attacker. The attacker then only needs to determine whether data that is stored on his node reflects an access packet or passport or not, which might be possible through traffic analysis. Another possibility is the interception of access packets or the passport itself if the attacker runs a node that serves as a client manager to neighboring nodes.

Decrypting access packet and passport only yields their current state, as access packet and passport are moved by the client on each logout. With a moving passport, the position of the passport has to be re-determined again and again, hence, a successful attack is limited in time and the gain restricted to the information that was present in the cracked but potentially old passport. The same conclusion applies to moving access packets.

After successful authentication, the Identity Provider supplies the user with full access to his IDs. The possession of matching private keys to the IDs of a user is what Service Providers and Policy Enforcement Points use to validate that the user was authenticated by the Identity Provider. Hence, authentication can additionally be attacked by circumventing the Identity Provider and attacking MaidSafe's hash function in order to generate IDs with the same fingerprint or the actually to generate exact same ID. This can be done offline: the attacker only has to acquire the IDs of the targeted user, which is easy in case of public IDs, but needs an attack on anonymity in case of the MAID.

Especially the loss of account credentials in MaidSafe is highly critical as they allow access to the entirety of the user's data, activities and assets, without the possibility to suspend the account. In addition, all data can be lost if the user has no local backup of his data and the attacker chooses to delete it within the MaidSafe network. This means that with an successful attack on authenticity, other security goals can be easily attacked subsequently.

Integrity is protected by Service Providers and Policy Enforcement Points, and relies on authenticity checking by those components. This check can be attacked by fooling the components into believing the attacker was correctly authenticated, which in turn requires a successful attack on authenticity as described above. The check can also be circumvented by controlling the remote Policy Enforcement Point, i.e. the majority of data managers, and ignore update and delete operations or to allow write operations by nodes that are not authorized by the data owner. This also works if the the majority of data holders or data holder managers is owned by the attacker.

With a majority of client managers an attacker can also simply deny service by not forwarding the user requests to the data managers. The client cannot detect this kind of attack as he cannot verify whether the data was actually stored. Yet, the attacker cannot delete data that was already stored previously.

Confidentiality is provided by transport layer encryption between the components of the architecture and end-to-end encryption of stored data or messages sent to other users. Transport layer encryption can be circumvented by being a Service Provider that gets in contact with the unencrypted data.

End-to-end-encryption is provided by the Self-Encryption mechanism, which can be attacked by attacking authenticity and using the data maps stored in the passport, or by a direct attack on the encryption algorithm. If the attacker does not want to attempt decryption of random data, he needs to know the position of the target data, but can then request it and perform an offline attack. The attacker can also use the fact that Self-Encryption is not indistinguishable under a chosen-plaintext attack (not IND-CPA).

To attack end-to-end encryption of messages between users, the attacker can again choose between an attack on authenticity or a direct attack on the cryptography. As MaidSafe does not employ Perfect Forward Secrecy (PFS), an attack on authenticity combined with the logging of prior messages sent between the users yields the additional benefit of being able to decrypt former communication for free.

Availability is a property of the Service Providers or the remote part of the Identity Provider. If the adversary controls the majority of neighboring nodes w.r.t. a given key, the adversary can either isolate users by dropping all communication, or attack the integrity of the user's data, i.e., deleting the data, which also attacks MaidSafe's data availability guarantee.

Anonymity can be provided as long as the attacker cannot learn sufficient amount of information about the user in order to de-anonymize him. Service Providers inevitably learn something about the user that uses them. An attack on anonymity does not require majorities in groups, but only a single node, as all nodes receive the same communication. With one client manager or data holder manager, the adversary is able to establish a mapping between the MAID, MPID or PMID and the IP address of the user, which he can then use to de-anonymize the user with information from outside of the scope of the MaidSafe network. This threat can be circumvented easily and cheaply by using a VPN access or TOR to separate MaidSafe and non-MaidSafe data traffic, though.

B. Generic Security Analysis

The two biggest problems found with MaidSafe are the full dependence of the security on username and password in combination with the non-existence of security monitoring capabilities. Through these problems, successful attacks always lead to total loss or, respectively, full disclosure of all user data – in case the attack is actually noticed. Once the encrypted data is spread out in the MaidSafe network and therefore distributed globally, there is no way to limit potential damage whenever cryptographic keys are compromised. This problem is inherent in a completely distributed approach such as the one employed by MaidSafe. The user does not know where and by whom his

data is actually stored. With centralized or federated systems, the user knows who is in charge of storing his data, which enables damage limitation and continuous security monitoring in cooperation with the provider.

A solution to this problem would be to combine the completely centralized and completely decentralized approaches into a single solution. A certain subset of a user's data, e.g. very sensitive data like MaidSafe's passport, could be stored on trusted entities, while other less sensitive data is stored on nodes operated by random entities.

V. Conclusion

In this paper we analyzed and evaluated the security of the MaidSafe architecture. MaidSafe is an emerging personal cloud platform that runs completely decentralized and aims to replace today's centralized service approach by dropping its key design principles. To this ends, it employs and combines a set of mechanisms, protocols and algorithms such as Self-authentication, Self-encryption, and P2P-based PKIs. We first provided a condensed description of these key mechanisms and extracted the only implicitly given identity and access control management architecture (IAM) as well as the corresponding trust relationships from the perspective of an end-user. We then analyzed how these IAM components and trust relationships can be attacked and also which type of knowledge or resources would be required to perform such attacks. Our analysis shows that MaidSafe has potential to fulfill its design goals, and we believe that MaidSafe provides sufficient protection against mass surveillance attacks. Yet, attacks that target individual users are possible if the attacker has enough resources or knowledge in order to break the Self-authentication mechanism for a given user. As a more generalized result, we come to the conclusion that it is not yet clear whether a completely decentralized approach such as MaidSafe (which stores all user data in the public network) really reduces the risks: the probability of a successful attack can be reduced with MaidSafe, but at the cost that the damage is more severe if an attack is successful, due to the lack of containment features.

In our next steps we want to analyze and quantify how routing table poisoning attacks can reduce the required resources when trying to dominate the network. We also want to evaluate how additional counter-measures which enable distributed security monitoring, e.g. limiting the number of login trials or detecting attacks, have to be designed and integrated.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[2] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," April 2014. [Online]. Available: http://gavwood.com/Paper.pdf

[3] N. Lambert and B. Bollen, "The SAFE Network: a New, Decentralised Internet." [Online]. Available: http://maidsafe.net/docs/SAFEnetwork.pdf

[4] Tor Project: Anonymity Online. [Online]. Available: https://www.torproject.org/

[5] P. Brody and V. Pureswaran, "Device Democracy: Saving the future of the Internet of Things," September 2014. [Online]. Available: http://public.dhe.ibm.com/common/ssi/ecm/gb/en/gbe03620usen/GBE03620USEN.PDF

[6] IBM, "ADEPT: An IoT Practitioner Perspective," January 2015. [Online]. Available: http://de.scribd.com/doc/252917347/IBM-ADEPT-Practictioner-Perspective-Pre-Publication-Draft-7-Jan-2015

[7] E. Sit and R. Morris, "Security Considerations for Peer-to-Peer Distributed Hash Tables," in *Revised Papers from the 1st International Workshop on Peer-to-Peer Systems*. London, UK, UK: Springer-Verlag, 2002, pp. 261–269. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687810

[8] L. Lu, J. Han, Y. Liu, L. Hu, J. Huai, L. Ni, and J. Ma, "Pseudo Trust: Zero-Knowledge Authentication in Anonymous P2Ps," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 10, pp. 1325–1337, Oct 2008.

[9] K. Butler, S. Ryu, P. Traynor, and P. McDaniel, "Leveraging Identity-Based Cryptography for Node ID Assignment in Structured P2P Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 12, pp. 1803–1815, Dec 2009.

[10] J. Douceur, "The Sybil Attack," in *1st International Workshop on Peer-to-Peer Systems*. Springer, 2002.

[11] J. Dinger and H. Hartenstein, "Defending the Sybil attack in P2P networks: Taxonomy, Challenges, and a Proposal for Self-Registration," in *The 1st International Conference on Availability, Reliability and Security*, April 2006.

[12] A. Mondal and M. Kitsuregawa, "Privacy, Security and Trust in P2P environments: A Perspective," in *17th International Workshop on Database and Expert Systems Applications*, 2006, pp. 682–686.

[13] G. Paul, F. Hutchison, and J. Irvine, "Security of the MaidSafe Vault Network." [Online]. Available: http://pure.strath.ac.uk/portal/files/34898763/Paul_etal_wwrf32_vault_network.pdf

[14] D. Irvine, "Peer to Peer Public Key Infrastructure," 2011. [Online]. Available: https://github.com/maidsafe/MaidSafe/wiki/unpublished_papers/PeerToPeerPublicKeyInfrastructure.pdf_not_yet?raw=true

[15] D. Irvine, "Self-Authentication," 2010. [Online]. Available: http://maidsafe.net/Whitepapers/pdf/SelfAuthentication.pdf

[16] B. Kaliski, "RFC 2898; PKCS# 5: Password-Based Cryptography Specification Version 2.0," 2000.

[17] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System based on the XOR metric," in *Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[18] D. Irvine, "MaidSafe Distributed Hash Table," 2010. [Online]. Available: http://maidsafe.net/Whitepapers/pdf/MaidSafeDistributedHashTable.pdf

[19] D. Irvine, "Autonomous Network," 2010. [Online]. Available: http://maidsafe.net/Whitepapers/pdf/AutonomousNetwork.pdf

[20] B. Parducci, "extensible access control markup language (xacml) specification," 2005.