Security of the MaidSafe Vault Network

Greig Paul, *University of Strathclyde,* Fraser Hutchison, *MaidSafe.net,* James Irvine, *University of Strathclyde*

***Abstract*—The MaidSafe network is an open-source, decentralised, autonomous network for data storage and retrieval by end user applications. All data is stored on the network within vaults, which are member nodes of a self-managed network resembling a distributed hash table (DHT). We explore the design of the vault network, including the self-managing nature of both vaults and data, and attack vectors worth consideration and further research.**

I. Introduction

THE MaidSafe network aims to provide a decentralised, autonomous network for the secure storage and retrieval of user data. Within this network, there are no centrally operated servers - instead, users provide storage capacity to the network via *vaults*, which are ordinary computers making available their resources to the network. This network of vaults builds upon the fundamentals of the Sigmoid distributed network concept [1], and presents itself as a distributed network of independent network nodes, each capable of storing data in key-value pairs, as a distributed hash table (DHT). Like in a DHT, each vault has its own unique address, which is derived from a cryptographic hash, ensuring vaults are spread evenly throughout the available address space.

Within the network, we shall use the term *node* to refer generically to any client or vault which is participating in the network, on behalf of a user. It is possible for a user to have no clients, no vaults, or any combination of clients and vaults, and these need not necessarily be on the same computer, or local network. We shall consider a *client* to be running software which implements the MaidSafe API, capable of making requests to the network on behalf of the user, for the purposes of storing, retrieving, or mutating data (where permitted). We shall consider a *vault* to have long-term storage (such as hard disks), as well as running the vault software. Vaults on the network store the chunks provided by clients, and return them when they are requested. Vaults on the network also serve an important role as supervisor nodes (only vaults are used when selecting the nearest 4 neighbours for the purposes of management, since clients are likely to go online and offline regularly, in the course of using the network).

Each vault on the network is managed by its 4 nearest neighbour vaults, with proximity determined by node address rather than geographical location. As the node address is derived from a cryptographic hash, it is not practical to select the nodes managed by any given node on the network. As nodes join and leave the network, the managers of a particular node will change if a nearer neighbour joins the network.

A vault's nearest neighbours can be identified through a simple bitwise XOR function, based upon the vault addresses (hashes) - for any given node address, there can logically be no other network node at the same distance as any other node (as the numerical significance of each bit is considered in the distance calculation). The ability to determine if two nodes are nearest neighbours is provided by the DHT, and ensures the resulting network can accurately determine if a given node is, as it claims, the nearest neighbour of another node.

II. Network Overview

The network's DHT is used to locate and store a multitude of different types of data within the MaidSafe network - for the purposes of simplicity, we shall refer to a node as being located at a given address in the table - nodes join the network with identifiers obtained by hashing a randomly generated keypair, and storing their identity data within this location on the network. When locations in the network are used, such as those of proximity or location, these locations are logical - two logically adjacent nodes (by address), have no increased probability of being geographically close. As a result, on average across the network, each node's nearest neighbours should be uniformly selected from the network. To prevent an attacker from carrying out an offline attack (i.e. using brute force techniques to generate a keypair which will hash to a desired location in the network), the actual address used is influenced by contributions from neighbouring nodes at the point of connecting to the network.

III. Vault Overview

As with all peer-to-peer storage networks, the long-term health and sustainability of the network is dependent upon users contributing resources, rather than simply freeloading, as discussed in [2] (within the context of peer to peer file sharing networks). Each data chunk is replicated to 4 separate locations on the network, in order to provide geographical redundancy, as well as to allow for vaults going offline to undergo maintenance, or in the event of network outages. While this means that each chunk uses a total of 4 times its capacity to be stored on the network, the network-level deduplication achieved through convergent encryption offsets this. Given that 75% of user data is believed to be non-unique [3], where two or more users hold the same file, the overhead of replicating the data on the network is quickly balanced by the deduplication of user data.

In order for the network to be sustainable, it is expected that each user contributes to the network by providing at least one vault. It is possible for the user to delegate this task (for example, by paying someone else to provide resources on their behalf), or carry this out by themself (by running vault software on their own computers or servers). By offering storage space to the network, a user is entitled to store data on the network, through clients they may run.

IV. Network Architecture

Every node on the network is managed by 4 vaults, in a constantly overlapping supervision scheme, such that every node is under the management of a group of its peers. To ensure there is a motivation to follow the rules of the network, each node is assigned a reputation by the remainder of the network. The 4 managers of a node are responsible for ensuring that the node under their supervision follows the network rules. In the event of a node acting maliciously (or otherwise in accordance with the rules of the network), this is reported to the rest of the network, who can verify this if required. For example, if a node which stored a given data chunk has lost the chunk, or has corrupted it, this can be verified by any other node on the network, by simply requesting the chunk. If a node's reputation is reduced sufficiently, it will be quarantined from the network, in order to prevent damage or detriment to the rest of the network.

A client on the network sends all its requests to the network via its 4 nearest neighbour vaults (which are its managers). These managers are not selected by the client (as discussed in section II), and therefore are able to act impartially with regard to the client. When a client wishes to store data, the client's managers contact the 4 vaults closest to the address of the chunk being stored, and request they act as chunk information managers for this chunk. The chunk information managers are responsible for distributing the actual chunks to other vaults for holding, and recording where these chunks are stored. In the event of these chunks going offline, the chunk information managers are responsible for detecting this, and (if necessary) replicating the chunks to other vaults, and keeping a record of this. The chunk information managers identify 4 vaults to hold the actual chunk, acting as chunk holders, and request that the 4 managers of each of these vaults (the chunk holder managers)ensure that the vault in question holds the requested chunk.

This is illustrated in figure 1, where a selection of vaults (each with its own address, not illustrated in the figure) are responsible for managing nodes to whom they are closest. It is worth noting that clients, vaults, chunks, and chunk information, are all held within the same address space in the DHT.

V. Self-Monitoring Network

The MaidSafe network is self-monitoring, in the sense that there is no central entity responsible for ensuring and enforcing policies upon other nodes on the network. Each node on the network has an identifier, which is a cryptographic hash, and the node forms part of the MaidSafe DHT. Each node is managed by 4 other nodes, where proximity is derived from the separation of nodes by their identifier, as per the Kademlia DHT. [4]

A. Request Authentication

In order to prevent the need for every request to be signed, client requests are authenticated by their identity - a node can only connect using an identity it holds the private key

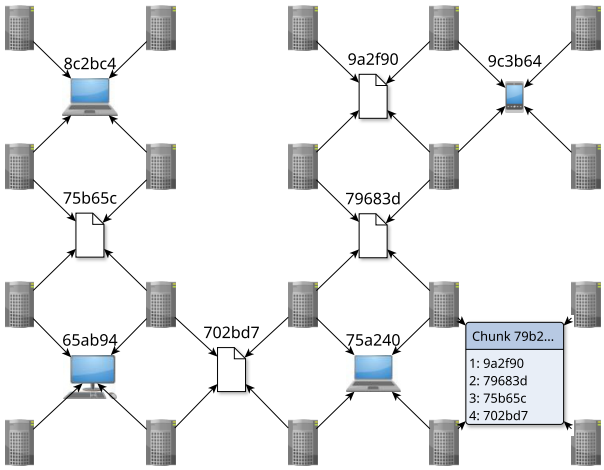

Fig. 1. A representation of a small area of the network, with clients, vaults, chunks and chunk information within the address space

for (in order to present itself to the network), and at this point establishes a mutually authenticated connection with its 4 nearest neighbours. After this connection is established, further communications will be sent using the negotiated connection, thus validating the origin of the request was the client itself. The client's identity is protected, as its IP address is scrubbed by its managers prior to any request being processed.

B. Node Loss Handling

The MaidSafe network anticipates that in a real-world network, nodes will become available and unavailable continually, whether as a result of network connectivity issues, or simply reboots to install system updates. The act of a vault becoming unavailable will be detected by that vault's managers, who will inform the data managers of each chunk previously held on that node. These data managers may request an additional copy of the data chunk to be made (using one of the other copies on the network).

Given a vault will also act as a manager for other nodes on the network, the act of a node becoming inaccessible will trigger a process of identifying the next manager for the vault under the oversight of the now-offline vault, based on the proximity of that vault to other nodes on the network.

Nodes on the MaidSafe network are ranked by the network, based on the principle of each node taking care of its own interests. A node is ranked based on the amount of data it can store, and the amount of data it loses (through corruption, deletion, being offline for extended periods of time, or simply responding slowly to requests).

C. Opportunistic Data Caching

In order to improve the performance of the network, opportunistic caching [5] is implemented by nodes. When a request to retrieve data is received, each node in the chain will add cacheable (i.e. immutable) data to its own cache,

to speed up future requests for this data. This makes denial of service attacks (both single-source and distributed) much more difficult, as requested data will be delivered from the nearest neighbour to the requesting node, reducing the load on the network. The caching process also makes popular data faster to access for every node on the network (as more popular data will likely be located near to the nodes dealing with the requests, due to their cache). This means that the MaidSafe network can act as a content delivery network, with dynamic caching of requested content throughout the network, enhancing availability and redundancy, and reducing transport delays.

D. Network Requirements

A key part of the network, currently undergoing development, is the resource management logic of the network. As briefly covered in section III, a significant aspect of any decentralised storage network is its resource management policies. In particular, the need for regular users of the network to contribute towards its upkeep, through the provision of storage and bandwidth, is of great importance, in order to ensure the network remains truly decentralised.

If there is insufficient motivation for users to contribute storage capacity to the network, and the bandwidth necessary for such storage to be used, storage provision will ultimately become centralised around those nodes (and their operators), who are willing to provide storage to the network. If this group is insufficiently diverse, it would result in a network which was almost entirely reliant upon one small group to provide the vast majority of its resources.

Existing research has explored the behaviour of users on previous peer-to-peer file sharing networks, indicating it is a very real concern that the majority of users may choose not to provide any resources to the network. For example, on the FastTrack P2P network, 1% of connected users provided 73% of the bandwidth needed to share files. [6] Likewise, on the Gnutella network, it was found that around 70% of users shared no files, and the top 1% of users were providing around 50% of all responses to requests for data on the network. [7]

VI. Implementation of Services

The network is currently designed predominantly as a data storage and retrieval system. Data on the network is stored within the DHT, where the 'key' is typically the hash of the chunk (for immutable data), and the 'value' is the chunk data itself. To this effect, the main functions of the network are *GET* and *PUT* requests, which retrieve the data held under a given chunk address, or store data under a given chunk address. In order to demonstrate how a more complex service can be constructed upon the network, we shall describe the process of logging in and authenticating to the network, in a decentralised manner, without relying on any server to verify credentials, or have any position of trust. We shall then explore the ability for this network to be used to create a service capable of storing and retrieving a user's files from their client device.

A. Implementation of Decentralised Login

In order to connect to the network, a random, one-use identity is generated, allowing the client device to connect to the DHT, and make requests for data. As data on the network can be retrieved by any user (given the data is itself protected from unauthorised access via convergent encryption and obfuscation), as discussed in section VI-B, the client is at this point able to request existing data chunks, but not create any new ones.

The client software first requests the authentication credentials from the end-user, through its user interface, and combines the identity information (username, password and PIN). The hash of the combined identity information is used as the user's *identifier* - this is the key within the DHT, where the user's identity information is located. A key is derived from the authentication credentials, using the PBKDF2 key derivation function. This key is used to decrypt the data stored on the network, at the determined identity address within the DHT.

Decryption of the user's identity data results in a random key, which is combined with the authentication information, in order to yield the address of the user's data atlas on the MaidSafe network. The data atlas is a mutable chunk (or set of chunks) containing the addresses of each data map the user has access to (a data map is a list of the information needed to access a stored file via its constituent chunks, including the hashes of each plaintext chunk, to permit decryption of the stored data). Having access to this data atlas, the client may now locate any chunks it requires (such as one containing its authentication keys for connecting to the network as an authenticated user), request these, and reverse the convergent encryption to obtain the original data. At this point, the client would connect to the network using its regular keys, thus restoring it to its original location within the network, and permitting it to authenticate to its 4 nearest neighbours, allowing modification of existing, or storage of new, data.

B. Implementation of a File Storage Network

1) Data Storage: An obvious use of a decentralised network such as this is to regard it as a decentralised filesystem, which the user can access from anywhere they require. By providing a software-layer implementation of the MaidSafe network, it would be possible to mount or bind a user's data to a virtual location on the device they are using to connect to the network. We shall consider only the network-level implementation of such a system, and not how the storage would be mapped to a location in the user's operating system (and refer the interested reader to software implementations of other user-mode filesystems such as FUSE).

In order to store a file on the network, it is first split into a minimum of three chunks, each a maximum of 1 Megabyte, and a minimum of 1 Kilobyte. For files of trivial size which cannot be split into at least 3 chunks of 1 Kilobyte, the file is stored, encrypted, directly within the data map, after undergoing convergent-encryption - we shall therefore consider the storage of a non-trivial file to demonstrate the process fully.

Each chunk of the file is hashed using the SHA-512 hash algorithm, and each chunk is then encrypted with an AES-256

key derived from the hash of that chunk, and the two previous chunks, in a cyclical manner, such that the first chunk will use the two last chunks for key derivation. This is the process of convergent encryption [8], [9], whereby the ciphertext of any given plaintext shall be the same for any user carrying out the encryption. This provides network-level deduplication [8], [10], as discussed in section III.

Following convergent encryption, each chunk is obfuscated, by repeatedly XOR'ing the contents of the chunk with the unused bytes of the key derivation stage above. This process is deterministic , based upon the contents of the file, and therefore does not inhibit file-level deduplication. The resulting chunk can be uploaded to, and stored on, the DHT (if it does not already exist). If the chunk already exists on the network, another user (or the same user) has already uploaded it, and there is no need to store another copy.

In order to permit the client to later retrieve their files, it is necessary for the client to have knowledge of, or be able to retrieve;

- the address of each encoded chunk making up the file
- the position of each encoded chunk within the file
- the original hash of each plaintext chunk, prior to convergent encryption

Naturally, in order to make this storage system usable on other locations of the network, it is necessary for this information to itself be stored on the network (to allow a user to access their data by logging in from any client). As such, a data map is created for every file on the network, where these three pieces of information can be stored. In order to prevent another user from discovering the data map for a given file, and accessing it, the data map is protected through the same convergent encryption process as discussed above for data. The requisite information for the data map is stored within the user's data atlas, which, as described in section VI-A, is directly decrypted using the user's authentication information.

2) Data Retrieval: To retrieve a given piece of data, a user must log into a client (the login process will obtain the user's data atlas in the process), and identify which file they wish to retrieve - the information from the data atlas can be used to present a listing of files, perhaps in a mounted filesystem, and allow the user to select their desired file. At this point, the information obtained from the data atlas is used to request the retrieval of the data map of the desired file. This data map provides the addresses of chunks to be retrieved to recover the file itself. Following receipt of these chunks from the network, the client reverses the XOR-obfuscation process, using the original hashes of the chunked plaintext, and removes the convergent encryption from the chunks, to restore the user's original file.

VII. Security Analysis

Within the MaidSafe network, we have identified some key areas from which security, in the conventional sense of confidentiality, integrity and availability, is derived, and shall now review these.

A. Confidentiality

Within the MaidSafe network, data confidentiality is achieved through three steps, all part of the data storage process, as detailed in section VI-B1. Data undergoes convergent encryption via AES-256, where the keys are derived from the cryptographic hash of 3 adjacent data chunks. This introduces the first security consideration.

1) Known Plaintext Attacks: Since the key for a given plaintext is derived from the hash (which is a function of the plaintext input), it stands to reason that convergent encryption is vulnerable to known plaintext attacks. This is not a new finding, and is documented elsewhere, but is nonetheless relevant. [11] In this case, we can somewhat limit the extent of this attack, since despite the use of file chunking, the use of adjacent plaintext chunks in key derivation ensures that two coincidentally identical chunks (as part of different files) will not yield to the same ciphertext (and thus cannot be detected as being the same). It is necessary for the two previous chunks in the file to also be identical, in order to obtain the same ciphertext output. Likewise, during the obfuscation step, when the encrypted data is repeatedly XOR'd with bytes derived from the unused keying data, it is clear that the same ciphertext output will not be obtained for two identical chunks, unless the two preceding chunks were also identical.

As this attack is a known plaintext attack however, it does not compromise the confidentiality of data - it is necessary to know the precise contents of three chunks, in order to determine that one chunk of a file is common with a *target* file. If an application on the network wished to mitigate against this, it could make use of regular symmetric cryptography with a non-deterministic key, which would remove global deduplication but remove the possibility of known plaintext attack, as discussed in [11].

2) Usage of Asymmetric Cryptography: Within the MaidSafe network, usage of asymmetric cryptography is limited to signatures (for the purposes of authentication), and in securing the UDP connections between nodes on the network. The confidentiality of data chunks held on the network is not dependent on the security of any asymmetric cryptography, since regular AES encryption is used for all decryption operations, from the data atlas itself, down to individual file chunks.

B. Integrity

In general, chunk integrity on the network is handled in two ways - firstly through the use of digital signatures, and secondly through the use of self-validating data. When immutable data is stored on the network, there is no need for it to be signed, since it is not possible for any entity on the network to later update it. Since an immutable chunk is stored on the network at the location of its hash, any node can verify that the data is correct. In the event of the data being incorrect, the managers of the node holding the corrupted data will reduce the reputation of that node. The fact the data is invalid is able to be checked, as any user on the network can validate an immutable chunk by comparing its hash with the ciphertext contents of the chunk.

In the case of mutable data, each ciphertext chunk is signed by the owner of the chunk, at the point of it being stored on the network. In order to update that chunk in future (i.e. to mutate its contents), this signature will be verified by each node prior to modifying the stored data. In the event of RSA signatures being forgeable (for example through quantum computing), the integrity of this mutable data can no longer be guaranteed, as an attacker may be able to introduce fraudulent update requests to the network. The contents of such data will remain confidential though.

C. Availability

1) Offline Nodes: In order for a chunk to remain available within the network, it is necessary for at least one vault holding a valid copy of the chunk to be online, at the time of the request, with sufficient resources (i.e. bandwidth) to respond to the request. To mitigate this risk, the managers of chunks are responsible for ensuring that in the event of only 2 live copies of the chunk being available, a further 4 copies will be made (resulting in 6 live copies of the chunks being available). In the event of a chunk becoming permanently unavailable (i.e. all of the nodes which held it go offline simultaneously, and do not reconnect to the network) it will not be possible to recover that individual chunk. As such, it is important for data availability that the network is truly decentralised, such that no failure of an individual or group results in permanent loss of data on the network, if they are taken offline.

Availability concerns on the network are mostly mitigated through the duplication of chunks - if one node loses (or refuses to retrieve) a chunk, the same chunk should be accessible in between 3 and 6 locations elsewhere on the network (depending on how many nodes holding the chunk are online at the time). The reputation features of the network help to mitigate this further, by ensuring that intermediary nodes do not act maliciously on the network, as discussed in section IV.

2) Network Flooding Attacks: As the network uses a reputation-based system to attempt to detect malicious behaviour by nodes, we identified a possible attack, where an attacker could obtain network resources for a period of time (such as via a botnet, or rented cloud computing), and flood the network with nodes they controlled, in order to use the birthday paradox to surround some nodes on the network, gaining the ability to reach and control consensus on their actions (or spoofed actions).

VIII. Considered Attacks

With a distributed network such as MaidSafe, where nodes are responsible for the management and supervision of other nodes, a number of attacks should be considered, whereby nodes could potentially cause harm to other nodes. The following main types of attack have been identified to pose a risk, specifically with relation to the vault network.

A. Dishonest Vault Attacks

We firstly propose the dishonest vault attack, whereby a vault falsely claims to offer more storage capacity to the network than it is capable of holding. While the MaidSafe reputation algorithm takes into account only storage capacity offered by a vault which is actually used, a client is permitted to store as much data on the network as they themselves claim to offer the network.

This could lead to situations where a user could create a vault, offer a large quantity of storage space, then upload a large quantity of data from an associated client. They could then remove their vault from the network permanently, deleting any data stored on it, and their data would remain on the MaidSafe network, despite not making any contribution of storage capacity to the network.

This attack could be mitigated by only permitting clients to store data on the network, to a maximum capacity of the data they have been verified to be holding. This introduces a potential conflict, whereby a new user cannot store data on the network via their client (due to nobody storing data on their vault), and no other user will store data on their vault (due to their vault not being trusted). To alleviate this, it would be possible to permit users to store their own data on their own vault, in order to establish a reputation on the network.

B. Traffic Amplification Attack

A possible traffic amplification attack has been identified, in light of the role of vault managers, which are responsible for notifying data managers if a vault is no longer accessible, and is believed to be offline. In the event of a vault going offline (resulting in no introduction of new traffic into the network by an attacker), the 4 manager nodes of the offline vault are required to contact the data managers (4 nearest neighbours) of every data chunk held by the now-offline node.

As such, a node can generate a significant volume of signalling traffic which will be uniformly distributed throughout the network (due to the uniform distribution of data throughout the network). This serves as an effectively unbounded traffic amplification attack, which could potentially lead to a denial of service attack, particularly if a number of nodes were taken offline simultaneously.

Possible mitigations for this include having managers throttle the rate at which they inform data managers about lost vaults, and rate limiting how frequently a vault's state will be reported throughout the network, to prevent repeated joining and leaving the network from generating large volumes of traffic being sent throughout the whole network.

C. Read-Only Fallback Attack

Given client requests are processed by the client's 4 managers, which must achieve a quorum of 3 nodes in agreement for a request to be valid and accepted by the network, it is possible that an attack which compromised over 25% of the nodes on the network could result in portions of the network becoming effectively read-only, as two manager nodes would be able to block a legitimate user's write or delete operation by declining to approve any requests.

Given nodes joining the network are evenly distributed throughout the network, an attacker would not require the

ability to join particular areas of the network, rather simply connect many nodes to the network. The possibility of a botnet of compromised host devices to launch such an attack is equally possible.

Possible mitigations for this attack include legitimate nodes on the network being instructed to reduce the required quorum level required for a request to be accepted. Whether this attack is viable, and the point at which it would become viable, is a topic for further research.

D. Network-wide Denial of Service

Further research into the number of malicious nodes necessary to make the network effectively unavailable is also likely necessary. Theoretically, if a botnet or other group of malicious nodes were to make up around 75% of the network, they could achieve false quorum over invalid requests, and thus request erase data, or modify mutable data.

It is more likely that an attacker obtaining control of around 75% of the network would pose more significant threats though, such as simply destroying user trust in the network, such as the so-called *50% attack often seen in cryptographic currencies*.

E. Compromised Host Attacks

We identified through both simulation, and mathematical prediction, that it would be possible to carry out a 'birthday paradox' attack on the network, whereby an attacker simply wishing to cause harm or disruption could flood the network with nodes it controlled, knowing they only need to surround a single address with 3 or more malicious nodes, in order to exert control over that node. While as previously discussed, it is not possible to deliberately position these nodes around a desired point in the network, we identified that with around 0.8% of the network's nodes under the (temporary) control of an attacker, it was likely the attacker would have at least one node surrounded on the network, allowing it to exert control over that node as managers, reaching quorum on false actions. By way of example, we proposed a proof-of-concept attack, whereby an attacker would request deletion of any chunks it could, by acting as the chunk information managers (as defined in section IV), thereby causing the chunk holders to delete the chunks in response to a legitimate request, preventing access to that data for legitimate users. If the attacker were inclined to act out of financial motivation, they could request a copy of the chunk prior to its deletion, and request the user pay a ransom before uploading it to the network again.

We reported this possible attack vector to MaidSafe, who responded by proposing modifications which would include not permitting the deletion of immutable data by anyone, including its owner, and requiring all requests be processed by two groups of nodes rather than one. Under this proposed modification, a client would pass its request to its 4 managers, who verify the request based on the client's signature, then pass this request to a deterministically selected group of 4 other nodes, which would also verify the request based on its signature. By deterministically selecting the second group of managers, the birthday paradox no longer holds true for the network, since it would not be possible for the attacker to gain control over a node by simply surrounding it - the attacker would require the ability to surround specific nodes in the network, which is believed to be a difficult task which would require being able to effectively generate different values which, when hashed with SHA-512, result in close hashes around one particular point.

IX. Conclusion

The MaidSafe vault network offers a truly decentralised architecture for data storage throughout a network, with no single point of failure, and no central authority with control over a network. All nodes present on the network are managed by their peers, selected through a deterministic and uniform process based upon their unique identifier on the network. Through processes such as opportunistic caching, frequently accessed data is distributed throughout the network in an optimal manner, reducing the ability for malicious users to conduct (distributed) denial of service attacks. A number of other attacks against the network have been discussed, along with possible mitigation strategies.

X. Acknowledgements

This work was funded by EPSRC Doctoral Training Grant EP/K503174/1, and MaidSafe.net.

References

[1] D. Irvine, J. Irvine, and S. K. Goo, "Sigmoid (x): Secure distributed network storage," *WWRF*, 2011.

[2] P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge, "Incentives for sharing in peer-to-peer networks," in *Electronic Commerce*. Springer, 2001, pp. 75–87.

[3] J. Gantz and D. Reinsel, "The digital universe decade - are you ready?" EMC Corporation, Tech. Rep., 2010.

[4] P. Maymounkov and M. David, "Kademlia: A peer-to-peer information system based on the xor metric," in *1st International Workshop on Peer-to-Peer Systems*, 2002.

[5] S. T. Kouyoumdjieva, S. Chupisanyarote, O. Helgason, and G. Karlsson, "Caching strategies in opportunistic networks," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*. IEEE, 2012, pp. 1–6.

[6] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," *IEEE/ACM Transactions on Networking (ToN)*, vol. 12, no. 2, pp. 219–232, 2004.

[7] E. Adar and B. A. Huberman, "Free riding on gnutella," *First Monday*, vol. 5, no. 10, 2000.

[8] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*. IEEE, 2002, pp. 617–624.

[9] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," *Advances in Cryptology ...*, pp. 1–29, 2013. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-38348-9_18

[10] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller, "Secure data deduplication," *Proceedings of the 4th ACM international workshop on Storage security and survivability - StorageSS '08*, p. 1, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1456469.1456471

[11] Z. Wilcox-O'Hearn, D. Perttula, and B. Warner. Drew perttula and attacks on convergent encryption. [Online]. Available: https://tahoe-lafs.org/hacktahoelafs/drew_perttula.html