# Sigmoid(x):
# Secure Distributed Network Storage

David Irvine, Sigmoid Solutions Ltd;
James Irvine, University of Strathclyde;
Swee Keow Goo, University of Abertay

Secure data storage is a serious problem for computer users today, particularly in enterprise environments. As data requirements grow, traditional approaches of secured silos are showing their limitations. They represent a single – or at least, limited – point of failure, and require significant, and increasing, maintenance and overhead. Such solutions are totally unsuitable for consumers, who want a 'plug and play' secure solution for their increasing datasets – something with the ubiquity of access of Facebook or webmail. Network providers can provide centralised solutions, but that returns us to the first problem. Sigmoid(x) takes a completely different approach – a scalable, distributed, secure storage mechanism which shares data storage between the users themselves.

## 1. Background

The recent rise of portable personal computing devices (netbooks, tablets, and smartphones) has shown that the long-standing goal of anywhere, anytime access to data is valued by consumers. The ease of use, accessibility and instant-on nature of these consumer devices has changed how people think about computing. The netbook paradigm works extremely well for network-based data, such as email, social networking sites or streaming video, but it has thrown into sharp focus the missing link in the current computing schemes – secure ubiquitous access to personal data and the reliability of the local storage device] As prosumers generate increasing volumes of content, they will demand the same anytime, anywhere access to their own data. Today's standards may have prosumers backing up data to tapes, off site servers (that ultimately goes on tape) and other devices. This is generally carried out in a manner that will effectively copy the same data at least 10 times and store exact copies of data at least ten times.

Currently there are two approaches which can allow this to happen. The first is to store the data on the network. Current examples are social media sites such as Facebook or YouTube, or file sharing sites such as Dropbox and other similar 'cloud based' storage solutions, such as Apple's recently announced iCloud. While such solutions do provide anytime, anywhere access, there are inherent limitations with such centralized storage systems. Most provide only a few gigabytes of storage, whereas a terabyte of local storage retails for less than $100. Also, there are security concerns. If users are informed enough to set privacy settings correctly, they are still trusting the storage site itself to keep their data safe. Some recent network breaches show that this trust may be misplaced.

Network Attached Storage (NAS) devices have become widely available in recent years, but are difficult to configure and use, requiring knowledge of routers, firewalls, and DNS. To get around issues such as firewalls, some manufacturers of these devices provide a centralised access facility, so that files can be accessed via their site[1], having been uploaded from the NAS (since egress from home networks is usually straightforward). However, while easier to configure, these give rise to at least the same security problems as the network-hosted approach, if not considerably more in some cases.

Corporates are also now taking more note of the anytime, anywhere Internet access and have ever increasing requirements to allow secured datasets internally and externally. As communications technologies improved many corporates did move significantly towards various schemes for sharing data, particularly supply chain management, where suppliers would allow limited access to internal systems for stock checking, ordering, etc.

As the Internet grew, this should have become very widespread, but in fact it actually decreased as corporations and suppliers faced many difficulties, until at least the advent of 'cloud computing' which promises to extend this ability. Ensuring secure access or data sharing have become significant problems, as the ease of access to the Internet opened systems to attack from across the world. It is interesting to note that as communications capability has grown, companies are struggling to make full use of this, and in many cases resort to siloed approaches through filtering, firewalling and many

[1] For example MioNet (http://www.mionet.com/) from Western Digital, GoFlex Home from Seagate, Verbatim Mediashare, etc

similar techniques. The underlying contradiction of this approach has been recogised by, for example, the Jericho Forum[2], who promote a de-perimeterised model and security attributes within the data itself [3], but this has been slow to gain acceptance.

This paper will focus on two main issues: firstly, securing corporate data internally, and then secondly sharing corporate data externally to trusted parties. Achieving these inherently contradictory requirements simultaneously is a significant advantage of the proposed solution.

## 2. Sigmoid(x)

The proposed solution, Sigmoid(x), employs similar technologies to peer-to-peer file-sharing networks. The data is stored in the network and is therefore accessible anywhere, but the data is split into chunks and encrypted so that only owners or those explicitly authorised can access it.

Crucially, by encrypting it at source, the network entities which store the data cannot access the contents as anything remotely readable. Indeed, they have no knowledge whatsoever regarding the contents, which could be parts of files or pointers to other file chunks. This removes the requirement of trust that exists in the network-hosted approach. From a user point of view, the system is as simple to use as a file-sharing system. The encrypted files appear as a virtual file system accessed by the OS through a standard USB flash drive interface, and so file access is application independent.

This approach also allows data to be integrity checked without knowing what it actually is. Each secure chunk stored on the network is named with the hash of its contents. Nodes on the network can check the validity of data at any time with reference to this hash.

The file-sharing approach has some other significant advantages. Firstly, since files are no longer stored in a single point, either in the home network or in a network server, the data can be replicated, significantly enhancing robustness. In fact, the system monitors the location of data chunks, and ensures that these are geographically dispersed within the network. Secondly, while chunk encryption is dependent on other chunks in the file, encryption over a file is self-contained and independent of a user's credentials. This means that two identical files belonging to different users on different parts of the network are encrypted into identical secure data chunks.

The result is that common files, for example operating system files, do not need to be stored more than once (except for additional replications for robustness). Studies based on corporate backup suggest that between 75% and 90%[3] of data within a corporate network is duplicated. This means that significant savings in storage capacity can be achieved, even allowing for replication of chunks to ensure robustness and availability.

The removal of the requirement of trust in the file stores – termed 'vaults' in Sigmoid – has another significant advantage. While vaults need to be identifiable, that identity does not need to be bound to a trust value, and so identity management is greatly simplified and the overhead of a Public Key Infrastructure (PKI) is avoided. Also, since the system is peer to peer, all users of the system must make storage available and therefore provide a vault. As a result, a user's identity can be bound to a vault. I user may have multiple accounts, and multiple vaults.

When a user joins the system, they create a vault. As part of this process, two RSA public-private key pairs are created. The first pair is used to sign material from that vault, for example deletion requests. The second is used to allow for key revocation. The identity of a vault is the hash of the first public key, signed with the second private key. Identities are bound to vaults and can be used by other nodes as part of a ranking process. A user common name can also be created, and this is bound to the user's vault. This identity is signed by the vault, and is used for sharing data. It is possible to change this identity.

## 3. Implementation

The system is implemented in two main components.

The client software is the human interface and provides the encryption and authentication capabilities for data and access to data. It is implemented in C++ for performance. The services of Sigmoid(x) are implemented on the top of the MaidSafe open source Distributed Hash Table (DHT), which is based on the Kademlia DHT [4].

---

[2] http://www.jerichoforum.org

[3] http://www.snia.org/sites/default/education/tutorials/2011/spring/data/FreemanPearce_Advanced_Dedupe_Concepts_FINAL.pdf, or http://m.softchoice.com/files/pdf/brands/acronis/Storage_Savings_with_Acronis_Deduplication_white_paper.pdf

Nodes in the DHT contain data vaults and are responsible for the storage and validity (via a validity checking algorithm) of data on the network. These nodes have a form of intelligence that is loosely based on an ant colony. This in essence means they co-operate with each other for the good of the network. The nodes are programmed to act in their own interest and carry out tasks that help them specifically. The order of preference for actions is 1: the network, 2: the data, 3: the individual vault. This is achieved by creating a system that actively ignores and in some cases will respond very negatively to human interaction on them directly. In this manner the validity of action can be logical and therefore programmable.

All vault actions are managed and authenticated. There is no network logging, although incremental counters and last seen times may be maintained for ranking purposes. All such actions are transparent and more importantly uniform (all ants are equal, there is no queen ant).

The three main requirements are now discussed and implementation overviews given.

### 3.1 Secure Data

Data encryption in the prototype implementation uses a combination of cryptographically secure hashing and AES256 symmetric encryption. Files are chunked, hashed, and then the hash of each chunk is combined with the hashes of the preceding two chunks (cyclically) to produce keying material to encrypt that chunk. This chaining obfuscates common file fragments. The resulting encrypted chunk is the data stored in the network, and its hash stored in the data map. The system uses an underlying peer to peer network based on a Kademlia DHT. This is a very efficient hash table implementation that has been widely deployed in many networks of millions of nodes. Unlike standard peer to peer networks, data can be deleted from the network under instruction from authorised clients with the appropriate digital signature and validated ID.

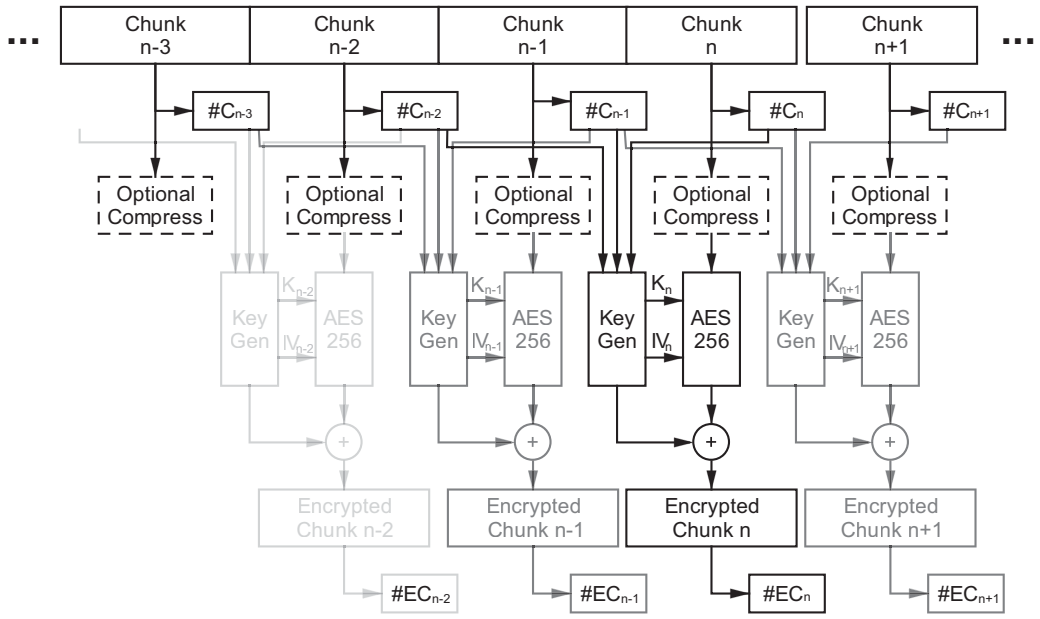


**Figure 1**: Encryption of file chunks

Encryption is based on the file content and so known plaintext attacks are possible. However, this is a requirement of deduplication, and is only possible when the adversary already has a copy of the file. It is possible to recognise chunks belonging to particular files, and therefore the presence of particular known files on the system, but it is not possible to tie those chunks to a particular user.

Deduplication and hash collisions are handled by checking whether data chucks exist within the network which the same hash values for all the chunks of the file being stored. If this is the case, then it is assumed that the file already exists on the network. If only a subset of the hash values exist on the network, it is assumed that hash collisions have occurred, the chunk size is modified, and the file recoded. Files are always split into a minimum of three chunks (smaller files are encoded within the data map directly for efficiency), so the probability of simultaneous collisions on all file chuck hashes are small enough to be ignored.

### 3.2 Autonomous Network

As fundamentally a peer to peer network, each new user of the system has to provide a data vault, with a capacity at least equal to the data they will be storing within the network. Data vaults can be authenticated using the public key generated when the vault was created. A maintenance process periodically validates the integrity of chunks using their hash values, replacing corrupted chunks or chunks lost when a vault went off line.

The peer to peer network operates a reputation system. This allows the number of replications of the data to be controlled based on the availability and otherwise of the nodes. Data is moved between nodes based on reputation and also to ensure the geographic spread to enhance robustness and accessibility. It can be seen from this that robustness is being traded off against network load, but such maintenance access can be timed for when the network is lightly loaded.

By changing the authentication and data mapping functionality the underlying data storage system can be implemented in different ways from a distributed peer to peer system, to one with a centralised user database for enterprise environments. Implementation results show that speeds of near to 80% of local access can be maintained with suitable network hardware, since data chunks are effectively being accessed in parallel which avoids hard drive access bottlenecks.

As previously noted, a vault's ID is the hash of its (self) signed public key. This allows communications to this node to be encrypted, and communications from this node can be signed and validated by the recipient. This further allows reliable ranking of the node.

File access to chunks uses iterative queries on the DHT. When data is retrieved from a vault, a cache instruction is sent to the neighbour of that vault nearest to the requesting node. This means that if data is requested again, it can be provided by the neighbouring node, and will be cached in the next nearest node. Caches time out after a period. As well as providing improved performance to legitimate queries (studies show that for 60% of data, 95% of accesses occur within a 30 minute window), this caching strategy makes the network highly resistant to Distributed Denial Of Service (DDOS) attacks since as data is requested, it will be cached more frequently and at points in the network nearer to the requestor.

### 3.3 Distributed Authentication

Access to data within the network is via a data atlas. The data atlas contains a number of data maps, and each data map contains pointers, in the form of hash values, to individual chunks of files. The data atlas, as a file itself, is also stored on the peer-to-peer network, and like other files is duplicated for redundancy. In order to access data, the user enters credentials into client software on any device within the peer-to-peer network. This client software uses the credentials to form a pointer to a data map for the data atlas. If these credentials are valid, the data atlas can be retrieved. If the wrong credentials are entered, then the data retrieved will be random chunks of encrypted data within the peer-to-peer network. Since it is not possible to identify within the network which data chunks belong to data maps and which data chunks belong to file data, it is not possible to use brute force attacks against the user's credentials. This authentication mechanism means that the user can retrieve the data atlas from any device, and can therefore access their data from any connected device.

Since the key pairs and user credentials used in the system are self-generated, no central authority is required, although in a corporate environment an option would be to have a central authority generating and signing the credentials. Credentials need never be transmitted over the network, and are never validated by a third party. This significantly enhances system security.

### 3.4 File Sharing

File sharing is facilitated through a system of distributed directories. A distributed directory is a group of zero or more data maps or distributed directories, identified by a Maidsafe Sharing ID (MSID). A user's data atlas is therefore a collection of data maps and distributed directories. In order to create a distributed directory, the user creates the MSID, and a public/private key pair. This key pair is used to sign the distributed directory, which as a group of data maps is stored as an encrypted file on the system like other data maps and user files.

To grant read only access to files, a user would distribute the MSID and the hash of the encrypted distributed directory file. Other uses can therefore find the data maps contained in the distributed directory, and access the files themselves. They can use the public key contained in the MSID to check the validity of the distributed directory.

To grant read/write access to the distributed files, a user also distributes the private key created for the MSID. Having the private key allows another user to sign a new copy of the distributed directory, corresponding to altered files. If some files in the directory are unchanged, they will not be duplicated but the original chunks will be referenced as per the normal de-duplication process.

While the second user has the private key of the MSID, and can therefore sign, and so recreate, the distributed directory, only the original user has the private key for the original file chunks. Since only a user with that private key can delete chunks, the original unaltered file will remain on the peer to peer network along with any new copies created by a sharing user. This offers considerable security, especially in the case of a shared user passing the MSID to a third party.

A distributed directory can itself contain distributed directories as well as data maps, allowing a hierarchical structure where users can share subsets of files shared to them, or write permissions can to granted to sets of files within an overall share.

Since sharing involves the passing of file pointers in the form of the IDs of the chunks through the distributed directory/data map structure, the files themselves do not need to be duplicated. File sharing is therefore quick and efficient. If sharing a file means that it is being accessed in a new location, chunks will be cached nearer to that location in the same way as if the file owner moved to a new part of the network. The system is therefore self organizing. If a file is shared to different uses in different combinations or with different permissions, only the data map need be replicated in the different distributed directories, not the file itself.

In the example in Figure 2, Alice shares files with both Bob (read/write) and David and Eric (read only). Bob in turn shares some of the files Alice has shared with Charles read only. Charles, David and Eric cannot alter files because they do not have the private key of the share. Bob, however, is given the private key corresponding to Distributed Directory 1 by Alice. He can therefore alter the file addressed by Data Map 1, or the files in Distributed Directory 2. Should he amend any files, the data maps will change, which means that the ID of Distributed Directory 1 will change. Since he has the private key he can sign the new ID. However, without Alice's private key he cannot delete the original file chunks and so the previous version of the file and directory will remain available to Alice.

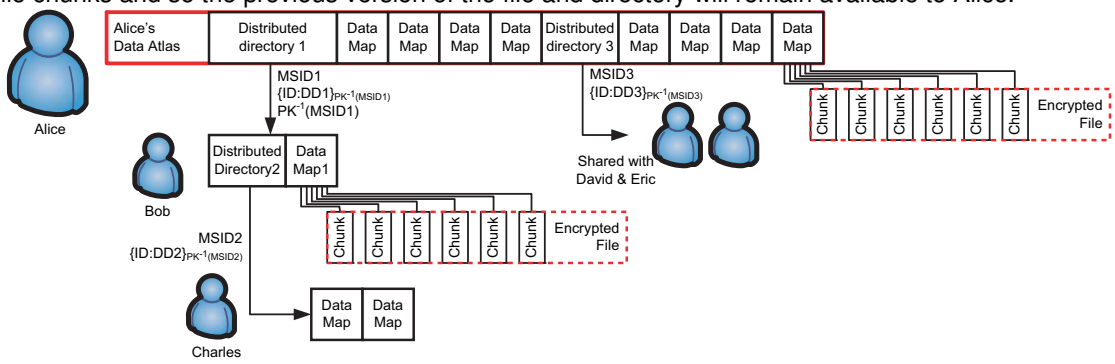


**Figure 2**: Distributed Directory Structure

## 4. Conclusions

Sigmoid(x) delivers a novel solution to the secure storage of data across a network. By distributing the data, anytime, anywhere access is maintained, with significantly enhanced resilience and security compared to a home server approach. The solution is also highly scalable – as more users join the scheme, they bring more storage, and the de-duplication of common files means more space is gained than is required to store replications. This also enhances security as the set of possible chunks that may be related to any given chunk grows in direct proportion to the amount of data stored.

## References


[1] Maidsafe Ltd, Distributed File System, UK patent GB1021312.2, 2010

[2] David Irvine, File System Authentication, UK patent GB 2453077, 2009.

[3] Marco Plas, The Book of Jericho 2.0: Security architecture from old to new, Capgemini Group, Available from http://www.domustechnica.com/images/stories/files/Book_of_Jericho.pdf

[4] Petar Maymounkov , David Mazières, Kademlia: A Peer-to-Peer Information System Based on the XOR Metric, First International Workshop on Peer-to-Peer Systems, Cambridge, MA, March 2002, p.53-65