5G-Enabled Decentralised Services

Greig Paul
Department of Electronic & Electrical Engineering
University of Strathclyde
Glasgow, UK
Email: greig.paul@strath.ac.uk

James Irvine
Department of Electronic & Electrical Engineering
University of Strathclyde
Glasgow, UK
Email: j.m.irvine@strath.ac.uk

***Abstract*—Increasing numbers of connectivity-dependent services are entering the market, many of which require users to entrust their data to the provider of the service. In exchange for having more convenient access to data, users must upload and entrust their data to these services. Companies are constantly launching and offering new products and services, to which users are submitting important data. A significant future challenge for users and service providers alike is retaining user trust, as services close and data is no longer available. This is increasingly important as mobile device connectivity improves to the point where cloud-based services are the norm, and local storage of data is less popular. We present a model demonstrating how viable, user-friendly services (not limited to data storage services) can be based upon a simple, decentralised storage-based back-end.**

I. Introduction

With the rise of modern, cloud-based services, users of these services are often no longer in possession of client software capable of accessing their data. Instead, users typically access their data via a vendor-provided web interface. Increases in the use of mobile computing devices to access data, such as smartphones, tablets, and browser-based systems (such as Chromebooks), mean services typically present a web-based API, which can be queried by the application running on the client system. The client is therefore simply a platform-native version of the web application, which is fully dependent on the operation of the underlying web service.

Web-based services offer a number of advantages, typically centred around user convenience. For example, most cloud services offer universal access from any device equipped with a web browser. The trade-off here, however, is that for data to be accessible via the web-browser, it is typically not encrypted in a manner which prevents the service operator from accessing the data. For the service provider to be able to display the data within the user's browser, the data is available to the web server process (and other processing services) which run on the service provider's infrastructure. A typical conventional service architecture is shown in Figure 1.

Even where a user accepts this, and trusts the service provider with their data, they are exposed to a number of risks. Firstly, there is the risk of server compromise, where a malicious attacker was able to gain unauthorised access to data held by the service [1]. This could (in some cases) be through no fault of the service provider itself, especially for services which use third party hosted infrastructure (such as "cloud" services or datastores, or virtual private servers which are hosted by another company) [2]. Another risk is that, in the event of the company no longer being able to provide services

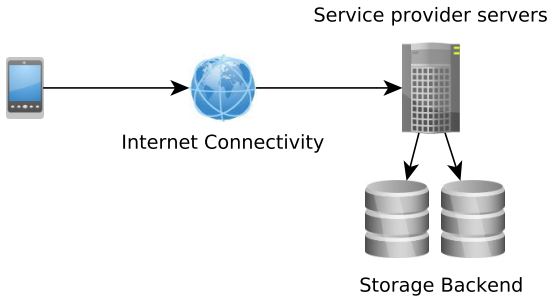

Fig. 1. A conventional cloud-based service architecture on a mobile device

(or of the service provider's infrastructure suppliers ceasing to trade), user data may not be accessible. A less extreme example was seen with the Google Reader service, which Google ceased to provide, meaning users could no longer access their data, or use the service after its termination [3], which is an inherent risk of relying upon externally hosted, centralised services. While user data backups could be migrated to a new service (if interoperable), this may leave users without access to their data for a period of time during the migration. Indeed, the utility of a remote service is the outsourcing of the processing itself, often in removing the need for in-depth or inefficient processing to be carried out on a mobile device (with battery constraints), rather than the data itself. If the company providing the service were to no longer trade, or there were to be a more major failure of hardware or software, the data held by the service may also be permanently unavailable [4].

This problem highlights the motivation for this work — to demonstrate that it is possible to provide services to users, which are user-friendly and simple to use, but are not reliant upon any single entity to provide server hosting. Through this decentralised service architecture, users retain access to their own data at all times, and cannot lose access to the access interface to their data. In this case, we demonstrate that it is possible to implement a wide variety of services using a simple storage-based backend, such as that of a MaidSafe-like decentralised storage network. [5] Such a storage network is based upon a Distributed Hash Table (DHT) for key-value pair based data storage.

In future, with 5th Generation connectivity to mobile devices, the push towards cloud-based services will most likely continue, particularly towards users of mobile devices. Faster connectivity such as 5G means the perceived need to store data locally is further reduced, and cloud-based offerings

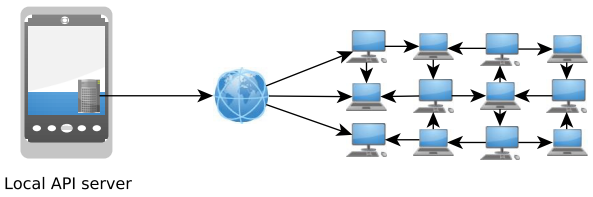

Fig. 2. The proposed decentralised storage-based architecture on a mobile device

can be pushed towards mobile users. We demonstrate here a concept to offer mobile-compatible services, which work with existing web-style APIs, removing the reliance of users upon centralised cloud-based architectures. At the same time, all data on the network is encrypted and unreadable to other parties, including the service provider. An example of our proposed service architecture is shown in Figure 2.

II. Decentralised Storage-Based Networks

A decentralised, storage-based network, such as MaidSafe [6] or Storj [7], is an example of a service, without fixed infrastructure under the control of a single entity. This lack of overall centralised control gives decentralised networks security against a number of attacks which are difficult to prevent in a centralised system. For example, if a web-based email provider's servers are compromised by attackers, or the operator of the service proves to be untrustworthy, the contents of any unencrypted content on the service is vulnerable to theft. With the recent drive towards cloud-based service adoption, users are increasingly being asked to entrust their private information (such as personal or confidential business correspondence and financial details) to the operators of online services [8]. This poses a security and privacy risk for service users.

Given the ease with which a domain name and servers can be purchased, it is eminently feasible and practical for *fake* cloud-based services to be launched, solely with the goal of encouraging users to upload data which could be abused or stolen by the operators of the service. Indeed, the honesty of a service provider is not guaranteed, and the ability for future network services to not require users to trust the provider of the service to behave ethically, legally, and honestly with their data, is clearly advantageous.

The MaidSafe network is an example of decentralised storage-based network, built around a Kademlia-like distributed hash table (DHT) implementation [9]. This work shall use the MaidSafe network as a platform upon which decentralised services can be created, although these techniques should be generic enough that they may be applied to any similar network. By breaking down the required operations of a service, it is possible to implement complex, interactive services, upon the storage layer, without users seeing the service simply as a storage system. This could be considered analogous to the use of centralised storage-only based cloud services, such as Amazon S3, although such a single point of failure [4] is desirable to mitigate. Previous work has focused on securing the data held within Amazon S3 [10].

III. User Authentication

User authentication is critical to the security of any network-based service, to ensure that only legitimate users gain access to data held on the service. Within a decentralised network, standard user identification and authentication procedures cannot be easily applied.

A. Standard Centralised Authentication

In a conventional centralised service (such as a website or other web-based service), each user of the service has an account, which is defined by a unique identifier (such as a username). A central authentication server has a record of all valid user accounts, and also contains a record of the authentication credentials required for a given account. This could include cryptographic hashes of passwords, or the public key identifier of a client-side certificate. When a user attempts to authenticate to the service, they are challenged by the authentication server, and prove they are authorised to log in using the requested account, by disclosing the account password, or decrypting and replying to a message using their client-side key. The basic principles of an authentication service are described by Gong in [11].

The authentication server can carry out other kinds of verification in the background at this point (such as using IP address geolocation to identify geographically unusual login requests based on previous account logins). To grant access to the user in question, the authentication server returns the user a token (valid for a session, or a finite period of time), which can be used to communicate directly with the service in question. In the case of a web service, this token would be stored by a web browser as a cookie, and sent with future requests to the web service.

The access points to the service itself carry out a verification process on every request received, whereby the supplied token is validated. A revoked (the user had logged out, or it has expired) or invalid (it was forged or non-authentic) token would result in the refusal of the request. Various strategies for the validation of such tokens exist (such as the service making an internal request to verify the token, or the token being signed, allowing it to be verified directly by the server providing the service).

The obvious weakness to this model for user authentication is that it places full trust on the authentication server. In the event of the authentication server being compromised (or a rogue server operator at the service provider), the service is no longer secure, since the attacker may generate legitimate authentication tokens (via the server), and use these to access any user's account. Centralised authentication remains the most commonly used method, and it is seen across almost all websites featuring some kind of login-based account, despite these weaknesses and limitations.

B. Decentralised Authentication

In contrast, authentication within a decentralised network is a much more difficult task. A decentralised network inherently has no trusted entity to act as the authentication server. An improved model for authentication is clearly necessary, since it is obviously unacceptable for any user of the network to

be able to pose as another network user in an undetectable manner, and this would be possible if the authentication-server model was used in a decentralised network [12].

In the absence of a single centralised entity to carry out user authentication, it is therefore necessary for every member of the network to be able to carry out authentication itself, at the time it is presented with a request. It is necessary to ensure that the authentication process is not critical to the confidentiality of data stored within the network. This removes the ability for other users to compromise the data of any individual user. This can be achieved by ensuring that each user encrypts their own data held on the network, using a key known only to them. The process of authenticating a user is therefore simply a means to prevent vandalism of data (and authorise the modification of data), since the underlying data would not be readable to an imposter.

By separating the authentication and security layers, authentication can be carried out with every request. To preserve the security of this authentication process, however, it must use strong authentication (where others can verify a user's identity, without gaining any information as to the authentication secret held by the client). Users can authenticate themselves to other nodes by signing data with a private key, corresponding to a verifiable public key. To ensure public keys can always be identified, the user identity can be defined as the cryptographic hash of the user's public key.

IV. Service API and Storage Model

We now consider the operations available on such a network, which can be used to form services, having established the possibility of using decentralised authentication techniques.

A. Available Primitives

In order to implement features which users expect from a service, such as the ability to store data (and later retrieve it), as well as inter-user sharing, it is necessary to consider the primitives necessary to achieve this. These form the basis of the API-style requests which are required by applications to implement functionality.

- GET(chunkID) - the `GET` operation retrieves, from the decentralised storage network, a given chunk of data, by the given address.
- PUT(data, chunkID) - the `PUT` operation will request a given block of data is stored on the network (with optional Chunk ID for mutable data).
- UPDATE(data, chunkID) - the `UPDATE` operation modifies a given block of data (provided it was stored as mutable data).
- DELETE(chunkID) - the `DELETE` operation, which can only be carried out by the owner of a chunk, may be used to remove it from the network.
- SHARE(chunkID, recipientID) - the `SHARE` operation shares a given data map with another user on the network, based on their public key.
- GETSHARES(senderID) - the `GETSHARES` operation locates any data shared with the current user by another user, as discussed in Section IV-B
- GETADDRESS(friendlyName) - the `GETADDRESS` operation retrieves the DHT-based address for a given user account, based on their friendly-name or other human-readable identifier (such as email address).

With these basic primitives, it is possible for a variety of complex, synchronous and asynchronous services to be implemented. The `GET`, `PUT` and `UPDATE` operations are relatively simple DHT operations, as described in [5].

The `SHARE` operation takes a given datamap (passed in as a chunk address), by encrypting it with the public key of the given recipient. This data must then be shared with the user by the application, although this sharing would typically be carried out asynchronously, as shown below in Section IV-B.

B. Inter-User Data Sharing

In order for collaboration and inter-user sharing to be possible, it is necessary for some coordination data to be securely exchanged between users, to notify them of such a file share. The MaidSafe protocol describes how inter-user sharing can be carried out, based upon the use of asymmetric encryption to create an additional encrypted data map, which would be communicated to the recipient of the shared data. [13] It does not specify the process through which this would be shared. To resolve this, we initially propose the use of a form of shared inbox, where a set of inbox addresses on the DHT are regularly polled by each user. These inbox addresses are derived from the cryptographic hash of both corresponding users' DHT addresses. There will therefore be 2 such inbox addresses for each pair of users. Data placed in the inbox from node A to node B would therefore be submitted to inbox $H(A|B)$, and data from node B to node A would be submitted to the inbox $H(B|A)$.

Data submitted to an inbox is signed by the sender (to prevent tampering or forged messages), and is encrypted by the public key of the recipient, to preserve the confidentiality of the data contained within the inbox. The inbox data simply consists of a data-map structure (from the storage network), which gives the recipient the keys needed to retrieve the data contained within those addresses on the network. When the recipient has added this data to their node, they update the content of the inbox to contain a signed (and encrypted with the sender's public key) message, stating that the inbox is empty, and to update with future messages.

In this way, it is possible for 2 users to asynchronously exchange data in a secure fashion. This permits users to share data or transmit messages between accounts on the network, which is a key part of any kind of collaborative protocol.

V. Example Service Implementations

A. User Identity and Discovery

The process of user discovery is an essential, but often overlooked, aspect of service implementation. While popular centralised services such as Whatsapp use a trusted-server approach (where other service users are discovered by sharing a list of existing contacts, and having the server carry out a lookup of existing service users), this approach is not suitable on a decentralised network. While it is not strictly necessary to

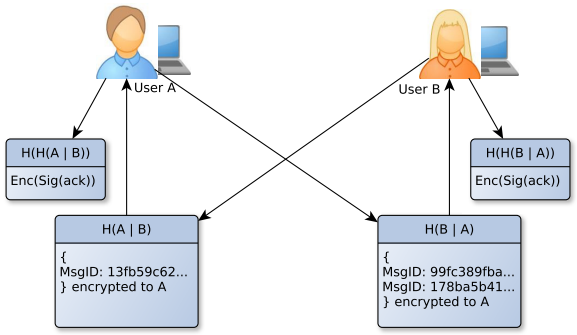

Fig. 3. The proposed inbox-based asynchronous data exchange protocol

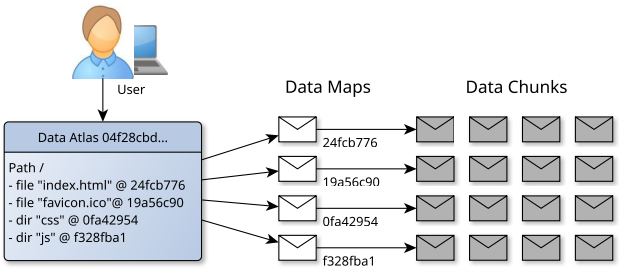

Fig. 4. File Storage Service Implementation

use user-friendly identifiers if a user's DHT address is known, this address is long and inconvenient to use.

For this reason, a decentralised model of usable identifiers (akin to a username) are desirable — they are easy to exchange and communicate with others. The standard means of ensuring uniqueness of user identifiers (by using a central server to allocate and allow or deny the addition of a new identifier) is naturally not possible in a decentralised network.

To register a new friendly name on the network, a user simply selects their desired identifier, and initiates a `PUT` to the network, to store a mutable chunk containing the user's address and friendly name, signed by their private key. This chunk is stored at the address of the hash of the user's desired friendly name. To retrieve the identity of a user on the network, based on their friendly name, another user would carry out a `GET` request against the hash of the friendly name, which returns the signed DHT address of the user owning that address (provided one exists).

Modification of these identity chunks is prevented by the nature of mutable data on the network, which cannot be modified without a valid (signed) request being received by the holder of each copy of a chunk. The nodes holding the chunk monitor each other to prevent invalid modifications from being accepted on the network, using the reputation system to remove an untrustworthy node from the network.

This technique is, however, vulnerable to flooding attacks, whereby all desirable user identities could be occupied by a small number of malicious users, to make the friendly name system less useful. Techniques such as the Hashcash proof of work mechanism [14] could potentially be used in future to deter such attacks.

B. File Storage

The first, and most simple, service which could be implemented would be a file storage system. The user runs a client, which connects to the network, and presents a virtual filesystem, mounted via FUSE or an equivalent library [15]. Filesystem metadata (including directory listings and hierarchies) is stored within a data atlas, which is an encrypted directory of the files available to the user, containing pointers to the address of the corresponding file's data map entry. Subdirectories are links to other data atlases, which themselves contain links to data atlases and maps. This data map entry itself is encrypted, holding pointers to the actual data chunks, as well as the keying data necessary to decrpyt these chunks. Figure 4 shows the relationship between the data atlas, data maps, and data chunks.

The process of retrieving a file from the network involves making a `GET` request for the user's encrypted root data atlas chunk (the location and key of which are deterministically derived from the user's identity). Following decryption (and provision of the root directory contents to FUSE), a file or directory will be selected, and the data atlas pointers will determine the next `GET` request to be made.

When a read request is made against a file, its data map is loaded, and a `GET` request is made to the network, to retrieve the chunks containing the requested file. The data map contains the keying information necessary to decrypt the file, and the file can thus be accessed by the user and their local applications, through FUSE layer. The process of updating a file varies, depending upon the use-case the user's client is configured for — the MaidSafe network supports both mutable and immutable data, meaning that it is possible to either preserve a full version history of a file indefinitely, or to simply overwrite previous copies.

For the case of preserving version history data, the file is stored in immutable form, meaning the data map (and chunks) are unable to be modified in future. By simply recording the address (and decryption key) for the relevant datamap in the user's data atlas as a past version, it can be accessed in future if necessary. If the user wishes to overwrite files, a new data map entry would be created, with a new pointer to a new set of chunks for the file. A (signed) request would then be made to the network, by the client, to remove the existing chunks and data map.

C. Asynchronous Messaging (Email-like)

In order to send asynchronous messages (where both parties may not be online simultaneously), the sender composes a message, addressed to a given user on the network (by their node ID). Note that friendly names can be resolved into node IDs, via the process described in Section IV-A.

The composed message is signed by the sender (to prove authenticity), and is encrypted with the recipient's public key. The content of the message (which can include text, and other content like attachments) is then uploaded to the network as regular data via a `PUT` request. The data maps pointing to this data (and holding the requisite keys) are then encrypted by the recipient's public key, and appended (via the *UPDATE* operation) to the mutually agreed inbox address on the network

(as mutable data). This inbox address is deterministic, and derived from the sender and recipient identifiers (meaning the two nodes need never directly communicate prior to being able to exchange messages). Since the inbox address is unique for each sender-receiver pair, the sender retains their own copy of the inbox contents (allowing them to append by mutating the contents of the inbox address).

To signify acknowledgement of the contents of the inbox, the recipient uses `PUT` (and subsequently `UPDATE`) on their own mutable acknowledgement chunk (stored at the hash of the inbox address), to place a signed (and encrypted) message for the sender, indicating which messages have been retrieved. This allows the sender to remove their records of sent messages, and clear the inbox contents.

In this case, it is not possible for unsolicited messages to be received, as the recipient must be aware of the identity of the sender (to be able to derive the inbox address). This means that a user will only receive messages from senders they have agreed to receive messages from.

VI. Security Considerations

Some security considerations applicable to networks such as this are discussed in [5]. The primary concern with regard to any network such as this is data confidentiality. In a decentralised network, users are no longer reliant upon the trustworthiness of the service provider to ensure their data confidentiality, instead making use of standard, proven, cryptographic protocols. A secondary concern is over the ease with which users could potentially abuse such a network through the creation of many accounts [16].

The principle security assumptions made are that the cryptographic hash algorithm used by the network remains secure (the MaidSafe network uses SHA512), and that symmetric AES encryption remains unbroken. The need for the hash algorithm to remain secure against partial pre-image attacks is to ensure that network nodes are added to the network in a uniform manner, thus preventing one attacker from flooding a portion of the network, and using their localised majority of untrusted nodes to take control and carry out malicious actions.

VII. Future Considerations

A DHT-based push-messaging based system may remove the need for regular polling of addresses on the network. By allowing a network user to subscribe to changes made to a given chunk of (mutable) data, it would be possible for a data manager to alert client software to the change of contents, with this message being routed through the network as a synchronous DHT-based message. These notifications could alternatively be carried over websockets (or a similar protocol), although this would potentially allow storage nodes to directly identify the node making the request. Nonetheless, supporting push-based updates to mobile clients would reduce power usage, and allow decentralised DHT-based networks to offer more user-friendly and mobile-optimised services.

VIII. Conclusion

We demonstrate a model for DHT-based distributed storage to be used as the back-end architecture for modern, user-friendly mobile services, removing the dependency upon a single service operator. By implementing the core application functionality locally, users can take advantage of increasing connection speeds and cheap online storage as part of a decentralised storage network, safe in the knowledge all their data is securely encrypted. Complex client applications can be built upon a simple back-end based solely around storage, offering new possibilities for users wishing to ensure their data remains available in the future, in the event of the developer of one of their applications ceasing to trade, or discontinuing the service.

Acknowledgment

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

References

[1] D. McCullagh, "Dropbox confirms security glitch–no password required," *CNET*, June 20 2011. [Online]. Available: http://www.cnet.com/news/dropbox-confirms-security-glitch-no-password-required

[2] S. Gallagher. (2014, October) Security bug in Xen may have exposed Amazon, other cloud services. arstechnica. [Online]. Available: http://arstechnica.com/security/2014/10/security-bug-in-xen-may-have-exposed-amazon-other-cloud-services/

[3] F. Manjoo. (2013, March) Why did Google Reader die? Slate. [Online]. Available: http://www.slate.com/articles/technology/technology/2013/03/google_reader_why_did_everyone_s_favorite_rss_program_die_what_free_web.html

[4] H. Blodget. (2011, April) Amazon's cloud crash disaster permanently destroyed many customers' data. Business Insider. [Online]. Available: http://www.businessinsider.com/amazon-lost-data-2011-4

[5] G. Paul and J. Irvine, "Security of the MaidSafe Network," Wireless World Research Forum. WWRF32, May 2014. [Online]. Available: http://strathprints.strath.ac.uk/48569/

[6] D. Irvine, J. Irvine, and S. K. Goo, "Sigmoid (x): Secure distributed network storage," *WWRF*, 2011. [Online]. Available: http://pure.strath.ac.uk/portal/files/34898763/Paul_etal_wwrf32_vault_network.pdf

[7] Storj - the future of cloud storage. Storj.io. Retrieved 16 February 2015. [Online]. Available: http://storj.io/

[8] W. K. Hon, C. Millard, and I. Walden, "The problem of personal datain cloud computing: what information is regulated?the cloud of unknowing," *International Data Privacy Law*, vol. 1, no. 4, pp. 211–228, 2011.

[9] 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.

[10] J. Yao, S. Chen, S. Nepal, D. Levy, and J. Zic, "Truststore: Making Amazon S3 trustworthy with services composition," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 600–605.

[11] L. Gong, "Increasing availability and security of an authentication service," *Selected Areas in Communications, IEEE Journal on*, vol. 11, no. 5, pp. 657–662, Jun 1993.

[12] D. Irvine, "Self-authentication," pp. 2–4, 2010. [Online]. Available: http://maidsafe.net/Whitepapers/pdf/SelfAuthentication.pdf

[13] ——, "Maidsafe distributed file system," pp. 1–4, 2010. [Online]. Available: http://maidsafe.net/Whitepapers/pdf/MaidSafeDistributedFileSystem.pdf

[14] A. Back *et al.*, "Hashcash-a denial of service counter-measure," 2002. [Online]. Available: http://hashcash.org/papers/hashcash.pdf

[15] M. Szeredi. (2005) Fuse: Filesystem in userspace. [Online]. Available: http://fuse.sourceforge.net

[16] G. Paul and J. Irvine, "A protocol for storage limitations and upgrades in decentralised networks," in *Proceedings of the 7th International Conference on Security of Information and Networks*, ser. SIN '14. New York, NY, USA: ACM, 2014, pp. 69:69–69:72. [Online]. Available: http://doi.acm.org/10.1145/2659651.2659724